آموزش مبانی کامپیوتر و برنامه نویسی به زبان C++

www.naghoospress.ir

نویسندگان :
دکتر ناصر قاسم آقائی
مهدی جابرزاده انصاری
علی دهقان

آموزش مبانی کامپیوتر و برنامه‌نویسی به زبان

C++

دکتر ناصر قاسم‌آقائی

مهندس مهدی جابرزاده انصاری

مهندس علی دهقان

شرکت ناقوس اندیشه
«سهامی خاص»

سرشناسه : قاسم‌آقائی، ناصر، ۱۳۳۰ -
عنوان و نام پدیدآور : آموزش مبانی کامپیوتر و برنامه‌نویسی به زبان C++/ ناصر قاسم‌آقائی، مهدی جابرزاده انصاری، علی دهقان.
مشخصات نشر : تهران: شرکت ناقوس اندیشه: زانیس، ۱۳۸۶.
مشخصات ظاهری : [۵۲۸] ص. : مصور، جدول.
شابک : ۸۵۰۰۰ ریال: ۰-۳۱۲-۳۷۷-۹۶۴-۹۷۸
وضعیت فهرست‌نویسی : فیپا.
یادداشت : واژه‌نامه.
یادداشت : کتابنامه: ص. [۵۲۸].
موضوع : سی ++C (زبان برنامه‌نویسی کامپیوتر).
موضوع : کامپیوترها— راهنمای آموزشی.
شناسه افزوده : جابرزاده انصاری، مهدی، ۱۳۶۴ -
شناسه افزوده : دهقان، علی، ۱۳۶۴ -
رده بندی کنگره : ۲ق۹۳س/QA۷۶/۷۳
رده بندی دیویی : ۰۰۵/۱۳۳
شمارهٔ کتابشناسی ملی : ۱۱۰۷۱۹۹

www.naghoospress.ir

چاپ اول

نام کتاب : آموزش مبانی کامپیوتر و برنامه‌نویسی به زبان C++
مؤلفان : ناصر قاسم‌آقائی، مهدی جابرزاده انصاری، علی دهقان
ناشر : شرکت ناقوس اندیشه
ناشر همکار : انتشارات زانیس
چاپ اول : ۱۳۸۶
تیراژ : ۲۰۰۰ جلد
لیتوگرافی : فرازنگر
چاپ : سعدی
صحافی : صفحه‌پرداز
حروفچینی و صفحه‌آرایی : واحد تولید
حروف‌نگار : مریم رضائی
قیمت با CD : ۸۵۰۰۰ ریال
شابک : ۰-۳۱۲-۳۷۷-۹۶۴-۹۷۸
ISBN : 978-964-377-312-0

مرکز پخش: شرکت ناقوس اندیشه
تهران: خیابان انقلاب، خیابان ۱۶ آذر، خیابان نصرت، پلاک ۱۵
تلفن: ۵-۸۸۹۸۴۶۹۴

فهرست

مقدمه

هم اکنون که در حال نگاشتن مقدّمهٔ کتاب هستیم، ماه‌هاست که کار نگارش متن کتاب پایان یافته و مراحل تایپ، صفحه‌آرایی و ویرایش آن، همگی به اتمام رسیده‌اند. لذا بسیاری از مباحث و موضوعات مهم در خصوص نحوهٔ مطالعهٔ کتاب، ویژگی‌های آن و بسیاری از مطالب دیگر پیشتر در خلال سطور کتاب عنوان شده است. اما مقدمهٔ هر کتاب جایگاه و اهمیت خاص خود را دارد، چرا که بسیاری از افراد قبل از آنکه تصمیم به مطالعهٔ یک کتاب بگیرند، مقدمهٔ آن را بررسی می‌کنند تا از ویژگی‌های کتاب مذکور مطلع گردند و درصورتی که آن کتاب را مفید یافتند، سعی در مطالعهٔ آن می‌نمایند؛ لذا پوزش ما را به جهت مطرح کردن مباحثی که اندکی بعد در داخل متن کتاب خواهید یافت پذیرا باشید.

درس مبانی کامپیوتر نخستین درس تخصصی است که دانشجویان رشته‌های کامپیوتر و فناوری اطلاعات، پشت سر می‌گذارند. متأسفانه در برخی از مراکز آموزش عالی، برخلاف سر فصلی که وزارت علوم برای این درس مشخص کرده، و بر بررسی الگوریتم و آموزش زبان روندنما (فلوچارت) تأکید کرده است، دانشجو را به یکباره و مستقیم وارد مباحث برنامه نویسی می‌کنند، و سعی بر آن دارند که مفاهیم مرتبط با شیوهٔ حل مسائل به روش الگوریتمی را به واسطهٔ یک زبان برنامه‌نویسی به دانشجویان آموزش دهند. البته می‌توان این رفتار را ناشی از خلأی دانست که نبود کتب مشخصی برای این واحد درسی به وجود آورده است. از طرفی داشجویان نیز تنها مبادرت به تهیهٔ کتب آموزش برنامه-نویسی می‌کنند و در نتیجه اثرات ناشی از انتقال نیافتن یک دید الگوریتمی به دانشجو در سال‌های بالاتر و در واحدهای بعدی خود را نشان می‌دهد. با توجه به این اوصاف نیاز به تهیهٔ کتابی جامع که هم مباحث علم کامپیوتر و طراحی الگوریتم و هم برنامه‌نویسی به یک زبان قدرتمند و قابل قبول در حد مبانی کامپیوتر را شامل شود، به شدّت احساس می‌شد. لذا عمده‌ترین هدف در تهیهٔ این کتاب بر طرف کردن نیاز دانشجویان و اساتید به یک کتاب واحد جهت تدریس در درس مبانی کامپیوتر است. در این کتاب به ازای هر یک فصل که در خصوص مباحث علم کامپیوتر و الگوریتم بحث شده است بلافاصله یک فصل نیز به بیان مطالب برنامه‌نویسی به روش ساخت‌یافته در ارتباط با همان مفاهیم الگوریتم-نویسی، و تحت زبان C++ پرداخته شده است. بدین ترتیب دانشجو در طول یک ترم به مرور طی یک برنامهٔ جامع و هماهنگ هم مباحث مربوط به رسم روندنما و حل مسائل به شیوهٔ الگوریتمی را فرا می‌گیرند و هم مباحث برنامه‌نویسی معادل آن‌ها را. با عنایت به آنکه ارزش کتب درسی به میزان تمرینات موجود در آن است در فصول مختلف تمرینات بسیاری را به فراخور بحث و با ترتیب خاصِ از آسان به دشوار تنظیم نموده ایم.

متن کتاب بسیار ساده و روان و همراه با توضیح ریزترین نکات می‌باشد، که قضاوت در این زمینه را به عهدهٔ خوانندگان عزیز می‌گذاریم. کتاب حاضر با این دیدگاه نگاشته شده است که خواننده هیچ آشنایی قبلی با کامپیوتر ندارد و باید او را قدم به قدم تا مرز یک برنامه‌نویس نیمه حرفه‌ای کامپیوتر پیش برد. لذا افراد مبتدی هیچ نگرانی در خصوص فهم مطالب کتاب نداشته باشند چرا که این کتاب تا حد بسیار زیادی به شکل خود آموز طراحی شده است. از نظر شکل و ساختار کتاب و دیگر ویژگی‌های آن در قسمت پیشگفتار مطالبی را عنوان خواهیم کرد و تنها این نکته را می‌افزایم که در حدود ۵۰ نفر ساعت کارِ کارشناسی از سوی دوست عزیز آقای مهندس محمد حقایق که از دانشجویان کارشناسی ارشد در رشتهٔ روانشناسی در دانشگاه اصفهان هستند صورت پذیرفته، تا به شکل و قالب فعلی کتاب رسیده‌ایم با این امید که بتوانیم میل خوانندگان را برای مطالعهٔ صفحات بعدی کتاب هر چه بیشتر کنیم، و صفحات کتاب شکل خسته-

کننده‌ای را نداشته باشد. چه بسا کتبی وجود دارند که مطالب بسیار مفیدی را دربرمی‌گیرند اما به جهت صورت نا خوشایند صفحات و عدم تنظیم صحیح مطالب با اقبال عمومی روبرو نشده‌اند. به عنوان مثال در این کتاب قسمت‌های برنامه‌نویسی را به صورت رنگی وارد کرده‌ایم و به گونه‌ای رنگ کلمات انتخاب شده که در کامپایلرها انجام می‌گیرد تا بر یادگیری دانشجویان بیفزاید.

همچنین در ابتدای فصل سوم مطالب مفید و قانع‌کننده‌ای را در خصوص دلایل انتخاب زبان C++ (در برابر زبان جاوا) برای آموزش در این کتاب و برای درس مبانی عنوان نموده‌ایم و تنها این نکته را اضافه می‌کنم که اگر زبانی همچون جاوا را برای آموزش به افراد مبتدی در درس مبانی انتخاب کنیم، یک اشتباه برگشت‌ناپذیر خواهد بود. چرا که در آموزش جاوا از ابتدا باید مفاهیم شئ‌گرایی را مطرح کرد همچون classها، درحالی که دانشجوی درس مبانی هنوز با برنامه‌نویسی ساخت یافته نیز آشنایی ندارد، از طرفی این زبان برخی مفاهیم مهم شئ‌گرایی همچون سربارگذاری عملگرها را شامل نمی‌شود که خود نقص بسیار بزرگی برای یک زبان آموزشی در درس برنامه نویسی پیشرفته نیز محسوب می‌شود. لذا در جلد دوم این مجموعه نیز به بیان مفاهیم شئ‌گرایی و برنامه‌نویسی شئ‌گرا به واسطهٔ زبان C++ خواهیم پرداخت و در کنار آن زبان C# که دقیقاً دارای ساختاری مشابه زبان جاوا ولی تکامل یافته‌تر و نزدیک‌تر به زبان C++ است، مورد بررسی قرار می‌گیرد. بد نیست بدانید C++ مادر زبان جاوا و هر دوی این زبان‌ها مادر زبان C# هستند.

همچنین جهت تهیهٔ لوح‌های فشردهٔ شماره ۲ و ۳ و ۴ که شامل کامپایلر Visual Studio 6.0 و MSDN مربوطه است به وب سایت کتاب مراجعه کنید. شایان ذکر است تمامی برنامه‌های موجود در کتاب ابتدا بر روی این کامپایلر تست و اجرا شده و به درستی وارد کتاب گردیده است و همچنین برروی لوح فشردهٔ شماره ۱ که همراه کتاب عرضه گردیده تمامی برنامه‌ها همراه با فایل های اجرایی آن‌ها موجود می‌باشد. احتمالاً تا یکی دو ماه پس از چاپ کتاب اسلایدهای لازم برای تدریس کتاب با فرمت PPS. آماده می‌شود که برای دریافت آن‌ها می‌توانید به سایت کتاب با آدرس زیر مراجعه فرمایید:

http://www.programmingbook . 4t . com/

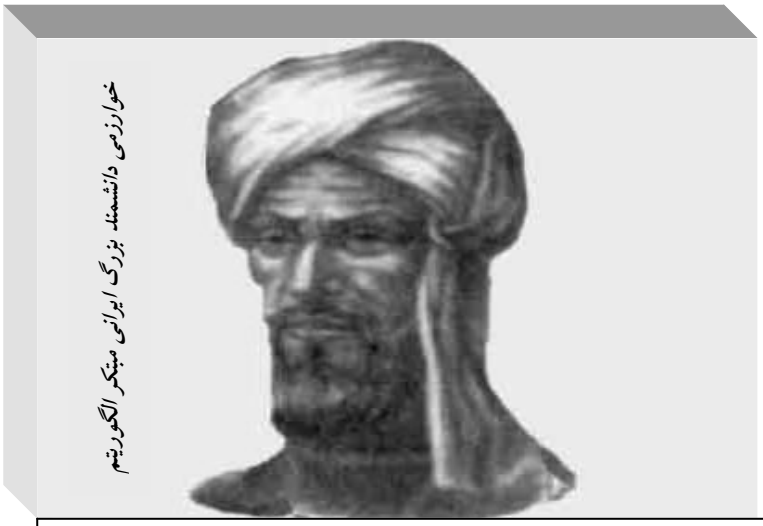
در پایان وظیفهٔ خود می‌دانیم از زحمات همهٔ کسانی که ما را در این راه به هر شکلی یاری رساندند تشکر کنیم. در این بین از دوست عزیز آقای رضا چنگیز که انصافاً مردانگی به خرج دادند و کار تایپ قسمت اعضمی از کتاب و ویرایش اولیهٔ آن را به عهده گرفت از صمیم قلبم تشکر می‌کنیم. همچنین از دانشجویان عزیز خانم‌ها سمیرا پویان فر و صالحه روانبخش که باز در زمینهٔ تایپ فصول برنامه‌نویسی نه تنها خودشان بلکه برخی از اعضای خانوادهٔ محترم‌شان را نیز درگیر کرده بودند صمیمانه تشکر می‌کنیم. همچنین از دوست عزیز آقای احسان حنیف پور و خانم لیلا زاهدی که در صفحه‌آرایی ما را یاری رساندند نیز باید تشکر کنیم.

موفق و مؤید باشید!

دکتر ناصر قاسم آقائی

مهندس مهدی جابر زاده انصاری

مهندس علی دهقان



خوارزمی دانشمند بزرگ ایرانی مبتکر الگوریتم

فصل اول

آشنایی با علم کامپیوتر و ویژگی‌های الگوریتم

اهداف فصل و چکیده مطالب :

۱. آشنایی با ویژگی‌های این کتاب و روش مطالعهٔ آن
۲. آشنایی با تاریخچهٔ کامپیوتر
۳. آشنایی با نسل‌ها و انواع کامپیوترها
۴. شناختی کلی از طریقهٔ کار و ساختمان کامپیوترهای امروزی
۵. تفاوت حل مسائل به شیوهٔ الگوریتمی و شیوهٔ مکاشفه‌ای
۶. تعریف الگوریتم
۷. بررسی ویژگی‌های الگوریتم
۸. آشنایی با حل مسائل به شیوهٔ الگوریتمی

۱-۱: پیشگفتار

شاید آخرین مطلبی که در هر کتابی نگاشته می‌شود مقدمهٔ آن کتاب باشد، و غالب خوانندگان نیز بدون توجه به نکاتی که معمولاً نویسنده در خصوص ویژگی‌ها و روش مطالعهٔ کتاب در مقدمهٔ کتاب بیان می‌کند، از نخستین صفحاتِ کتاب، مطالعهٔ خود را آغاز می‌کنند؛ و در نهایت امر، به جای کسب حداکثر بهرهٔ لازم، به اندک مایه‌ای دست می یابند و همهٔ تقصیرات را بر دوش نویسنده می‌اندازند که قلم بدی داشته و... .

لذا تصمیم برآن گرفتیم که در نخستین صفحات این کتاب به‌طور بسیار مختصر و در قالب یک نشست دوستانه و صمیمی سخنانی در خصوص ویژگی‌های این کتاب و طریقهٔ تدریس و مطالعهٔ آن بیان کنیم، باشد که خوانندگان عزیز التفات لازم را بفرمایند و این چند صفحه را به دقت مطالعه نمایند. امید است که با به کار بستن نکات مطرح شده در خلال این صفحات، حداکثر بهرهٔ لازم را از این کتاب ببرید.

دوست عزیز! شاید شما یک دانشجوی رشتهٔ کامپیوتر یا یک دانش‌پژوه آزاد و علاقه‌مند به علم کامپیوتر باشید، در هر دو صورت شما در آغاز راه طویل و بی‌انتهایی قرارگرفته‌اید که برای آن انتهایی را نمی‌توان متصور شد. امروز که نگارش این کتاب را شروع کرده‌ایم، با اطمینان به شما می‌گوییم که، اگر روزگاری افرادی وجود داشتند که از یک‌تا صد این علم را آموخته بودند و می‌توانستند ادعا کنند که در علم کامپیوتر به حد نهایی رسیده‌اند دیگر وجود ندارند. اگر نیم نگاهی به علوم دیگری همچون ریاضیات، فیزیک، و... بیندازید متوجه خواهید شد که این علوم هر چند سابقه‌ای چند صد ساله دارند ولی دارای رشد علمی چندان شتابانی نیستند. و هنوز هم می‌توان افرادی را یافت که حداقل در یک بخش از این علوم، [هرچند برای‌دوره‌ای کوتاه از زمان] به حد نهایی دانش‌های موجود در رشتهٔ کاری خود رسیده باشند. اما علم کامپیوتر در شکل امروزی خود که شاید سابقهٔ آن به یک قرن هم نمی‌رسد دارای چنان حجم گسترده و رشد فزاینده‌ای است که به واقع نمی‌توان انتهایی برای آن متصور شد.

لذا در این راه طویل و بی‌انتها از یک طرف باید عزم و اراده‌ای محکم و پولادین داشته باشید و از طرف دیگر می‌بایستی به درستی و با بینش قدم بردارید. با آنکه در عرصهٔ علم هیچ راهی به بی‌راهه نمی‌رود و تمامی راه‌ها شما را به مقصد و مقصود نزدیک‌تر می‌گرداند، اما نکتهٔ بسیار مهم این است که، در هر حالتی باید نزدیکترین راه را انتخاب کنید، و از مهمترین دلایل نگارش این کتاب هم، همین *است که راه را برای شما نزدیک‌تر گرداند*.

در اکثر دانشگاه‌ها کتب مختلفی برای درس *"مبانی کامپیوتر"* مورد استفاده قرار می‌گیرد و در کنار بیان اصول الگوریتم‌نویسی یک زبان ساخت‌یافته نیز تدریس می‌شود. لذا کتابی که پیش روی شماست در جلد نخست نه تنها اصول الگوریتم‌نویسی را به شما آموزش می‌دهد، بلکه آموزش برنامه‌نویسی ساخت‌یافته را نیز به واسطهٔ زبان C++ تحت کامپایلر Microsoft Visual Studio 6.0 دربر دارد.

البته در خصوص دلیل انتخاب زبان C++ برای آموزش برنامه‌نویسی می‌توان گفت که، در اکثر دانشگاه‌های دنیا به دلیل قابلیت‌های این زبان و در دسترس عموم بودن کامپایلر آن، همین زبان تدریس می‌شود. در C++ گرچه اصل بر برنامه نویسی شئ‌گرا است، ولی می‌توان به‌راحتی برنامه‌هایی مبتنی بر برنامه‌نویسی ساخت‌یافته را نیز در این زبان نوشت؛ که از مهمترین ویژگی‌های C++ به شمار می‌آید. سال‌های متمادی اساتید دانشگاه‌ها برای آموزش برنامه‌نویسی ساخت‌یافته از زبان پاسکال استفاده می‌کردند که مسلماً برای آموزش سیستم شئ‌گرا مجبور بودند به زبانی همچون C++ رجوع

کنند، اما رفته رفته با منسوخ شدن زبان پاسکال، زبان C++ جایگزین آن شد و با توجه به آنکه زبان‌های برنامه‌نویسی که بعد از C++ آمدند مثل Java و C# از برنامه‌نویسی ساخت‌یافته به‌طور کامل پشتیبانی نمی‌کردند، زبان C++ همچنان به‌عنوان مهمترین و مناسب‌ترین زبان آموزش برنامه‌نویسی برای افراد مبتدی باقی ماند. البته در جلد دوم این مجموعه در خصوص برنامه‌نویسی شئ‌گرا و تجزیه و تحلیل شئ‌گرا که در کمتر کتاب برنامه‌نویسی به آن اشاره شده صحبت خواهیم کرد و مطالب مهمی را در این زمینه عنوان خواهیم نمود. همچنین در بخش ضمیمه ها تفاوت های برنامه‌نویسی در کامپایلرهای مختلف C++ تحت دو سیستم عامل ویندوز و لینوکس را مورد بررسی قرار خواهیم داد تا شما خود را مقید به کامپایلر خاصی نبینید.

از دیگر ویژگی‌های مهم این کتاب ترتیب خاص فصول آن است، که پس از بیان مطالبی در خصوص الگوریتم و رسم کارنمای چندین مثال مفید، بلافاصله در فصل بعدی آن، به بیان مطالب برنامه‌نویسی معادل الگوریتم‌ها می‌پردازد، لذا رعایت این ترتیب فصول در خصوص افراد مبتدی، چه از سوی اساتید و چه از سوی دانشجویان بسیار حائز اهمیت است. البته امکان دارد مثال های صفحات نخستین چندان مورد توجه افراد غیرمبتدی قرار نگیرد اما رفته رفته مثال های مفیدتر و جذاب‌تری را مخصوصاً در فصول برنامه‌نویسی خواهند یافت.

از دیگر ویژگی‌های این کتاب این است که، در فصول مختلف با توجه به مطالب بیان شده تمریناتی تحت‌عنوان " *کار در کلاس* " بیان گشته است. *کار در کلاس‌ها* باید بلافاصله پس از یادگیری مطالب مورد نظر، توسط دانشجویان حل گردد تا در صورت نیاموختن قسمتی از درس فوراً از استاد خود کمک بگیرند. لذا توجه به این تمرینات و حل به موقع آنها بسیار مهم است.

همراه با کتاب یک مجموعه لوح فشرده‌ای ارائه گردیده که دانشجویان می‌توانند با توجه به نکات مطرح شده در ضمیمه شماره ۱ محیط Visual Studio 6.0را بر روی کامپیوتر خود نصب کنند و برنامه‌های موردنظر را در آن بنویسند و اجرا کنند. همچنین بر روی لوح فشردهٔ شماره ۱ کد مربوط به کلیهٔ مثال های کتاب قرار گرفته است. لوح شمارهٔ ۱ همراه با کتاب و لوح‌های شمارهٔ ۲ تا ۴ به‌صورت اختیاری عرضه شده است.

در پایان برخی از فصول، که نکاتی در خصوص برنامه‌نویسی بیان شده، صورت یک پروژه نیز بیان گردیده، که انجام به موقع این پروژه‌ها می‌تواند بسیار به فهم مطالب کمک کند. یادتان باشد که تمرینات برنامه‌نویسی این کتاب را باید به‌طور کاملاً انفرادی انجام دهید و از همکاری با دیگران در ترم نخست جداً اجتناب کنید.!؟

آشنایی مقدماتی با سخت افزار کامپیوتر نیز، از مباحث درس مبانی کامپیوتر به‌شمار می‌رود، اما به دلیل حجم مطالبی که در زمینهٔ الگوریتم و برنامه‌نویسی در این درس باید مطرح شود، اکثر اساتید خبره بحث تفصیلی در این زمینه را به واحد درسی آزمایشگاه کامپیوتر محول می‌کنند و در مبانی کامپیوتر تنها به شرح مختصری در این زمینه می‌پردازند. لذا در ادامهٔ این فصل به‌طور بسیار مختصر به بیان مطالبی در خصوص ساختار کامپیوترهای امروزی می‌پردازیم. سپس مفهوم الگوریتم را در خصوص چند مثال ساده و مقدماتی بررسی می‌کنیم. و ویژگی‌های الگوریتم ها را بر می‌شمریم، اما بحث تفصیلی در خصوص رسم کارنما را به فصل دوم واگذار می‌کنیم.

۲-۱ : تاریخچه و نسل های کامپیوتر

احتمالاً نخستین سؤالی که از خود می‌پرسید این است که چه لزومی دارد با تاریخچهٔ کامپیوتر آشنا شویم؟

جواب این سؤال بسیار روشن است! چرا که چگونه یک فرد می‌تواند ادعا کند که کارشناس کامپیوتر می‌باشد در حالی که نمی‌داند کامپیوتر از کجا آمده و چه سرگذشتی داشته!؟ به همین جهت برای شمایی که قرار است در رشتهٔ کامپیوتر کارشناس بشوید آشنایی با تاریخچهٔ کامپیوتر بسیار ضروری به نظر می‌رسد. با توجه به این مقدمهٔ کوتاه به بیان تاریخچه کامپیوتر می‌پردازیم، امید است که شرینی این مطلب در خاطر شما بماند.

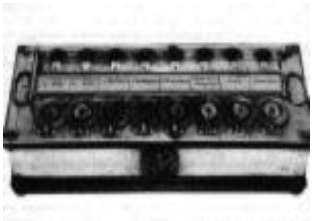
اگر به کتب فرهنگ لغت مراجعه کنید و به دنبال معنای کلمهٔ رایانه یا کامپیوتر بگردید به اولین تعریفی که برمی‌خورید چنین است که ؛

" کامپیوتر ماشینی است که قادر به انجام برخی محاسبات ریاضی و منطقی می‌باشد"

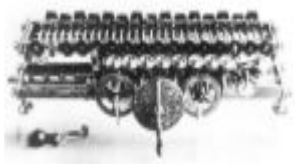
لذا می‌توان در یک کلمه معنای کامپیوتر را معادل عبارت *ماشین محاسبه‌گر* یا در معنی صحیح‌تر *حسابگر* دانست. بنابراین برای دنبال کردن تاریخچهٔ کامپیوتر باید به دنبال وسایلی باشیم که انسان در طول تاریخ برای محاسبات ساخته و از آن بهره گرفته است.

از اولین وسایلی که انسان برای انجام محاسبات از آن بهره گرفت انگشتان دست خود بود و دقیقاً به همین دلیل است که انسان‌ها برای محاسبات از مبنای ده استفاده می‌کنند. اما تلاش انسان همواره در جهت ساخت وسایلی بوده که او را در زمینه شمارش یاری کند و کامپیوتر آخرین دستاورد این تلاش است، تاریخچه کامپیوتر مراحل طولانی را طی کرده است که چرتکه اولین مرحله آن بوده بنابراین به قولی می‌توان کشور چین را پیش‌گام در زمینه ساخت کامپیوتر دانست. اما محاسبات انسان تنها به شمارش ختم نشد و وسایلی در جهت محاسبات گوناگون همچون محاسبات نجومی، محاسبات دریانوردی و... ساخته شدند. از جملهٔ این وسایل می‌توان به دو وسیله به نام‌های «لوح اتصالات» و «طبق المناطق» اشاره کرد که توسط دانشمند ایرانی «غیاث الدین جمشید کاشانی» جهت محاسبات نجومی و دریانوردی ساخته شدند.

اما تلاش‌های جدی در جهت ساخت ماشین‌های حساب به قرن هفدهم میلادی بازمی‌گردد که «ویلهلم شیکهارد» ریاضیدان آلمانی در سال ۱۶۲۳ طرح یک ماشین حساب را ارائه داد. پس از وی این تلاش‌ها ادامه یافت تا آنکه در سال ۱۶۴۲ «بلز پاسکال» که ریاضیدانی فرانسوی بود یک ماشین حساب مکانیکی(چرخ دنده‌ای) برای انجام عمل جمع و تفریق اختراع کرد که تصویر دو نمونه از آن را در زیر می‌بینید.



و بالاخره آخرین تلاش در جهت ساخت ماشین حساب مکانیکی منصوب به ریاضیدان آلمانی «لایب نیتز» می‌باشد که در سال ۱۶۹۴ توانست ماشین حسابی بسازد که چهار عمل اصلی را انجام می‌داد. تصویر این ماشین را نیز در زیر می‌بینید.

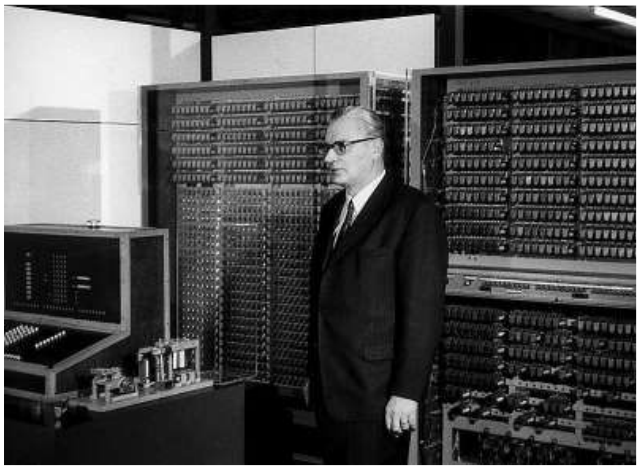


پس از آن، طرح ساخت ماشین‌های برنامه‌ریزی شده (خودکار) مـطرح شد که اولین تلاش موفق در این زمینه مربوط به «ژوزف ژاکار» فرانسوی می‌شود. وی توانست در سال ۱۸۰۸ یک چرخ بافندگی خودکار بسازد که توسط کارت‌های مشبک طرح های متنوعی روی پارچه ایجاد می‌کرد.

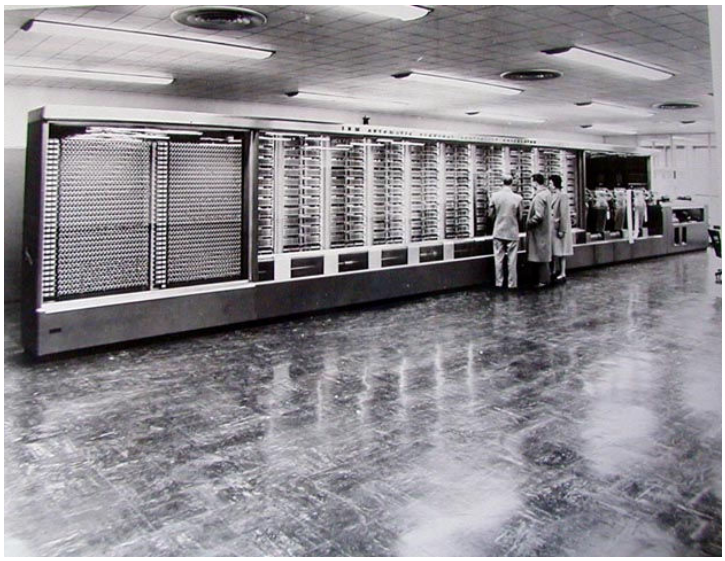
اما این موفقیت زمینه ساز این بود که با گسترش صنعت و تجارت و نیاز روز افزون به ماشین‌های محاسبه‌گر و ماشین‌های قابل برنامه‌ریزی جهت اداره کردن کارخانجات، ایدهٔ ساخت اینگونه ماشین‌ها در اذهان دو ابر قدرت صنعتی آن زمان یعنی آلمان نازی و ایالات متحده مطرح گردد.

در سال ۱۸۳۰ «چارلز بابیج» انگلیسی اولین نفری بود که طرح ساخت ماشینی با قابلیت *برنامه‌پذیری* را داد. وی وسیله‌ای با نام موتور تحلیل یا (Analytical) را طراحی کرد، هر چند که او به دلیل محدودیت امکانات نتوانست این دستگاه را بسازد ولی او را به پاس زحماتش پدر کامپیوتر می‌نامند.

اما نخستین تلاش در جهت ساخت کامپیوتر در مفهوم امروزی خود مربوط به سال ۱۹۳۹ می‌شود که مهندس «کنراد سوزه» از طرف سازمان تحقیقات فضایی آلمان مأموریت یافت یک ماشین محاسباتی بزرگ و پرتوان بسازد. تلاش‌وی در سال ۱۹۴۱ به ثمر رسید و ماشینی به نام Z_3 اختراع شد که براساس سیستم دودویی کار می‌کرد. دلیل اصلی موفقیت سوزه نسبت به طرح‌های ناموفق قبلی ابتکار وی در استفاده از همین سیستم دودویی بود. البته این ماشین در جنگ جهانی دوم به کلی نابود شد. در زیر تصویر مهندس سوزه را در کنار Z_3 مشاهده می‌کنید.



در امریکا نیز همزمان با تلاش‌های سوزه «مهندس هوارد آیکن» که از اساتید برجسته دانشگاه هاروارد بود در سال ۱۹۳۷ یک ماشین‌حساب الکترومکانیکی ساخت. اما اندکی بعد، با موفقیت مهندس سوزه و بهره‌گیری آیکن از ایدهٔ منطق دودویی بکار رفته در Z_3 وی نیز توانست در سال ۱۹۴۲ ماشین حساب تماماً الکترونیکی به نام ASCC بسازد. اما به جهت تأثیری که این ماشین بر پیشبرد تکنولوژی گذاشت پس از مدتی آن را Mark I نام نهادند. این ماشین اولین کامپیوتر دیجیتالی محسوب می شود و قادر به ذخیرهٔ ۷۲ عدد ۲۳ رقمی در حین محاسبات بود و در هدایت موشک‌ها در جنگ جهانی دوم بکار گرفته شد. تصویر این ماشین را نیز در زیر مشاهده کنید.



آیکن و همکارانش پس از مدتی Mark II را ساختند که در آن ضمن آنکه به‌طور کامل از تئوری دودویی استفاده کردند بیش از ۱۳۰۰ رله به‌کار گرفتند. این ماشین قادر بود عمل وقت‌گیر ضرب را در ۰/۲۵ ثانیه انجام دهد که خود رکورد بزرگی محسوب می شد.

در دانشگاه پنسیلوانیا نیز دو دانشمند جوان به نام‌های «پراسپراکرت» و «جان ماچلی» در سال ۱۹۴۴ موفق به ساخت نخستین کامپیوتر نسل اول که از لامپ خلاء استفاده می‌کرد، شدند. کامپیوتر آنها که به سفارش مرکز آمار آمریکا ساخته شده بود و اینیاک (ENIAC) نام گرفت و تا سال ۱۹۵۵ از آن استفاده شد. آن دو به دلیل قرضی که جهت پرداخت جریمهٔ تأخیر در تحویل ماشین اینیاک به سازمان آمار امریکا متحمل شدند، مجبور به گرفتن قرض‌های متوالی و ساخت ماشین‌های دیگری شدند، و متأسفانه تا پایان عمر نه تنها لذت زحمات خود را نبردند بلکه در قرض و بدبختی و در سنین میانسالی به جهت کار طاقت فرسای جان دادند.

در سال ۱۹۴۷ «جان فون نویمان» دانشمند آلمانی مقیم امریکا پیشنهاد کرد که کامپیوترها علاوه بر داده‌ها برنامهٔ کار محاسبه را نیز در خود ذخیره سازند و قابلیت برنامه‌ریزی کردن را به کامپیوتر بافزایند.

در سال ۱۹۵۲ دو کامپیوتر به نام‌های EDVAC[1] و EDSAC[2] که براساس نظریه نویمان کار می‌کردند ساخته شدند. این ماشین‌ها قابلیت‌های زیر را داشتند، که به عبارتی می‌توان تحت‌عنوان فواید نظریه نویمان به آنها نگریست:

۱. سرعت ماشین‌هایی که منطبق با نظریه نویمان بودند صدها برابر شده بود.

۲. امکان ایجاد تغییرات لازم در برنامه‌هایی که در حین اجرا دچار اشکال شده بودند به وجود آمد. به بیان دیگر برنامه نویسی در این زمان متولد شد. چراکه تا قبل از این تمامی عملیاتی که کامپیوترها انجام می‌داند با طراحی مدارات الکترونیکی برای آنها میسر شده بود. لذا کامپیوترهای آن زمان به نوعی تک منظوره بودند، و یا اگر هم چند منظوره به شمار می‌آمدند، کارهای ثابتی را می توانستند انجام دهند. با ظهور کامپیوترهای منطبق با نظریهٔ نویمان دیگر نیازی نبود برای تغییرات در کاربری کامپیوتر برای آن مدارات جدیدی را طراحی کنند بلکه برنامهٔ جدیدی را طراحی می‌کردند و کامپیوتر وظیفهٔ اجرای آن برنامه را به عهده داشت.

از حدود سال ۱۹۵۰ کامپیوتر به‌عنوان یک کالای تجاری توسط شرکت UNIVAC به بازار معرفی شد، پس از آن شرکت IBM که سازندهٔ ماشین‌های تحریر و لوازمات اداری بود در سال ۱۹۵۴ (معادل ۱۳۳۰ شمسی) کامپیوترهای نسل اول را وارد بازار کرد که مدل ۷۰۱ و ۶۵۰ نام داشتند و به‌عنوان مهمترین سازندهٔ کامپیوتر مطرح شد. در زیر تصویر مدل ۷۰۱ از سری کامپیوترهای IBM را مشاهده می‌کنید.

در ادامه به بیان نسل‌های کامپیوترها خواهیم پرداخت.

1. Electronic Discrete Variable Automatic Calculator
2. Electronic Delay Storage Automatic Calculator

نسل اول (لامپ خلاء Vacume Tube)

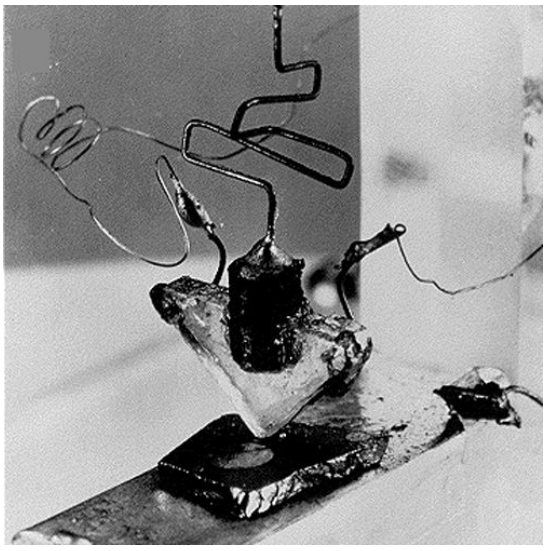
همان‌طور که پیش‌تر بیان شد در ساخت این کامپیوترها از لامپ خلاء استفاده شد و از سال ۱۹۴۴ تا ۱۹۵۵ مورد استفاده قرار می‌گرفتند. یعنی قبل از دهۀ ۱۳۳۰ شمسی ساخته شدند. اما به دلایل زیر این کامپیوترها به سرعت کنار گذاشته شدند:

۱. اندازۀ این کامپیوترها بسیار بزرگ بود و فضای زیادی را اشغال می‌کردند.
۲. به جهت استفاده از لامپ خلاء به نیروی برق زیادی نیاز داشتند.
۳. به علت گرمای زیادی که در لامپ‌های خلاء ایجاد می‌شد به وسایل خنک کنندۀ قوی نیاز داشتند.
۴. به علت گرمایی که در لامپ‌های خلاء ایجاد می‌شد امکان داشت در حین محاسبات این لامپ‌ها بسوزد و در نتیجه می‌بایستی تمامی محاسبات از سرگرفته می‌شد.
۵. مسلماً با استفاده از وسیلۀ گران قیمتی مثل لامپ خلاء قیمت این محصول بسیار بالا تمام می شد.

چنان که ذکر رفت یکی از اولین کامپیوترهای این نسل اِنیاک بود که دارای ۳۰ تن وزن، ۱۷۰ متر مربع مساحت بود و ۱۸۰۰۰ لامپ خلاء در آن به‌کار رفته بود و به ۱۵۰ کیلو وات انرژی الکتریکی نیاز داشت.

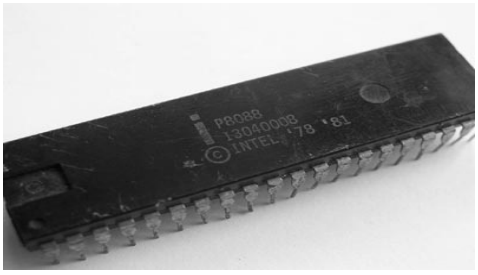
نسل دوم (ترانزیستور Transistor)

با اختراع ترانزیستور در سال ۱۹۵۰ تحول عظیمی در صنایع الکترونیکی رخ داد و در کامپیوترها نیز از ترانزیستورها به جای لامپ خلاء استفاده شد. این کامپیوترها دارای حجم کمتر و مصرف انرژی پایین‌تر و سرعت بالاتر بودند. در این کامپیوترها برای اولین‌بار از حلقه‌های مغناطیسی به‌عنوان حافظه‌اصلی(RAM) استفاده شد. گفتنی است این کامپیوترها در سال های ۱۳۳۰ تا ۱۳۴۰ شمسی مورد استفاده قرار گرفتند. در زیر تصویر نخستین ترانزیستور دنیا را مشاهده می کنید. آیا ترانزیستورهای کنونی را تا به حال دیده‌اید؟



نسل سوم (مدارات مجتمع IC)

با اختراع مدارهای مجتمع (Integrated Circuit) حجم کامپیوترها کاهش یافت و سرعت آنها افزایش یافت. چرا که این مدارات شامل بیش از صد عنصر منطقی بوده که هر عنصر منطقی خو شامل چندین عنصر الکترونیکی مثل دیود و ترانزیستور هستند و به روش خاصی برروی صفحاتی از جنس سیلیکون در چند سانتی‌مترمربع کنار یک دیگر چیده شده‌اند. و این خود نشان‌دهندهٔ میزان کوچک شدن کامپیوترها می‌باشد. در زیر تصویر یک نمونه IC را می‌توانید مشاهده کنید.



همچنین گسترش نرم افزارهای ساخت‌یافته نیز موجب افزایش سرعت این نسل و به طبع گسترش آنها شد. اولین کامپیوتر از این نسل را شرکت IBM در سال ۱۹۶۰ با نام IBM360 به بازار ارائه کرد و تا سال ۱۹۷۰ در مراکز تجاری مورد استفاده قرار می‌گرفت.

نسل چهارم (ریز پردازنده ها CPU)

در سال ۱۹۷۰ با متراکم تر شدن مدارات مجتمع به واسطهٔ فن‌آوری VLSI ساخت ریزپردازنده ها آغاز شد و کامپیوتر به‌عنوان یک کالای خانگی راهی بازارها شد. از جمله ریزپردازنده‌ها می توان به خانوادهٔ ۸۰۸۶ ، ۸۰۲۸۶ ، ۸۰۳۸۶ ، ۸۰۴۸۶ و... اشاره کرد.

نسل پنجم (کامپیوترهای هوشمند)

از سال ۱۹۸۰ با گسترش نظریهٔ منطق فازی ایدهٔ ساخت این کامپیوترها به‌طور جدی توسط ژاپن مطرح شد، که تلاشی در جهت ساخت کامپیوترهایی با ویژگی های انسانی است. براین‌اساس در تکنولوژی ساخت این کامپیوترها به‌گونه‌ای باید عمل شود که کامپیوتر قادر به اعمالی چون استنباط و استدلال‌کردن باشد. گرچه هنوز کامپیوتری با این ویژگی‌ها ساخته نشده است ولی ساخت آن دور از دسترس نیست؛ چنانکه شرکت ها و مؤسسات زیادی در حال کارکردن بر روی آنها هستند.

نسل ششم (کامپیوترهای انسان‌نما)

در این نسل هدف بر آن است که در ساخت کامپیوتر فعّالیت‌های مغز انسان به‌طور کامل کپی برداری شود. چه بسا کامپیوترهای این نسل دارای قدرت درک حواس پنج‌گانهٔ انسان و اَعمالی از این دست باشند. مسلماً سرعت، دقت و هوشمندیِ این نسل از کامپیوترها قابل باور نبوده و دارای ساختمان و مدارات پیچیده‌تری نسبت به نسل‌های قبل از خود هستند.

۱–۳ :سخت افزار و ساختمان کامپیوتر های امروزی

در این بخش هدف آن است که خوانندگان عزیز را با کلیاتی در خصوص ساختمان و نحوهٔ کارکرد کامپیوترهای شخصی (Personal Computers) آشنا سازیم. البته به دلیل پیشرفت های مداوم در ساخت کامپیوترهای PC، از ذکر جزئیات و ارائهٔ اعداد و ارقام در این مبحث معذوریم و تنها به ذکر مباحث کلیدی و بیان مطالب به‌صورت انتزاعی می پردازیم. سعی خواهیم کرد مدلی برای کامپیوتر ارائه کنیم تا ما را در درک چگونگی عملکرد کامپیوتر در خصوص اجرای برنامه‌هایی که به زبان C++ می‌نویسیم یاری دهد. و همان‌طور که پیشتر گفته شد، ذکر جزئیات در این خصوص را به واحد درسی *"آزمایشگاه کامپیوتر"* واگذار می‌کنیم.

مسلماً هر فرد مطلعی بر این مطلب صحه می‌گذارد که مغز انسان کامپیوتری بسیار پیچیده‌تر و به مراتب قوی‌تر از کامپیوترهای امروزی است. گرچه مغز انسان از نظر سرعت انجام محاسبات بسیار ضعیف تر از کامپیوتر عمل می‌کند، اما امکاناتی که مـغـز انسان در زمینهٔ تشخیص هوشمندانهٔ مطالب دارد، بزرگترین کامپیوترها را نیز پشت سر می‌گذارد. کامپیوترها به خودی خود فاقد هرگونه هوش هستند. کامپیوتر ماشینی است که صرفاً قادر به انجام عملیات ریاضی و منطقی است، و این متخصصین کامپیوتر هستند که با برنامه‌ریزی کامپیوترها، آنها را قادر به درک هوشمندانهٔ مطالب پیچیده می‌کنند. البته این مطلب را بدان جهت مطرح کردیم که برخی افراد به اشتباه فکر می‌کنندکامپیوتر ماشینی هوشمند است که قادر است به تنهایی جواب همهٔ سؤالات را بیابد.!؟

آن چیزی که در این خصوص مهم به نظر می‌رسد این است که، اگر به نحوهٔ انجام محاسبات توسط انسان و پردازش اطلاعات در مغز او توجهی داشته باشیم، می‌توانیم ادعا کنیم که بسیاری از قسمت‌های اصلی کامپیوترها که در پردازش اطلاعات درگیر هستند مشابه عملکرد انسان و مغز او [در انجام محاسبات] عمل می‌کنند.

هر انسانی برای انجام محاسبات بر روی تعدادی دادهٔ خام در ابتدا نیازمند آن است، که داده‌های خود را برروی یکسری برگه نوشته و برروی میزکار خود قرار دهد. سپس با رجوع به حافظهٔ خود یا منابعی که در اختیار دارد، راه‌حل مناسب را پیدا کرده، و با به‌کارگیری توانایی مغز خود راه‌حل مذکور را برروی داده‌های خام به‌طور مکرر اعمال می‌کند تا آنکه همهٔ داده‌ها پردازش گردد و نتایج لازم حاصل آید. همچنین انسان قادر است نتایج به‌دست آمده یا هر مطلب دیگری را در حافظهٔ کوتاه مدت و یا در صورت لزوم در حافظهٔ بلند مدت خود به خاطر بسپارد. از طرفی انسان در حین محاسبات مدام عملیات انجام شده را چک می‌کند تا از صحت نتایج به‌دست آمده اطمینان حاصل نماید. حال تصمیم داریم با توجه به مدلی که در مورد انسان ارائه کردیم مدلی برای کامپیوتر ترسیم کنیم تا با نحوهٔ عملکرد آن بهتر آشنا شویم.

۱. هر کامپیوتر در درجه اول نیازمند بخشی است که داده های خام را از کاربر خود دریافت کند. لذا در کامپیوتر اصلی ترین دستگاه ورود اطلاعات صفحه کلید است، که با آن اطلاعات عددی و کاراکتری را وارد کامپیوتر می‌کنند.
۲. همچنین هر کامپیوتر نیازمند بخشی است که نتایج به‌دست آمده از پردازش‌های خود را نشان دهد. از این رو صفحه نمایش(مانیتور) از متداول‌ترین وسایل جهت نمایش اطلاعات می‌باشد.

۳. از طرفی کامپیوتر نیازمند بخشی است که بتواند توانایی مغز انسان در انجام محاسبات ریاضی و اَعمال منطقی را بر روی داده‌ها میسر سازد. برای این منظور یک واحد پردازشگر مرکزی به نام CPU در نظر گرفته شده است. این واحد قادر است عملیات چهارگانهٔ ریاضی و همچنین اَعمال منطقی را انجام دهد. در فصول بعدی با این عملیات بیشتر آشنا خواهید شد.

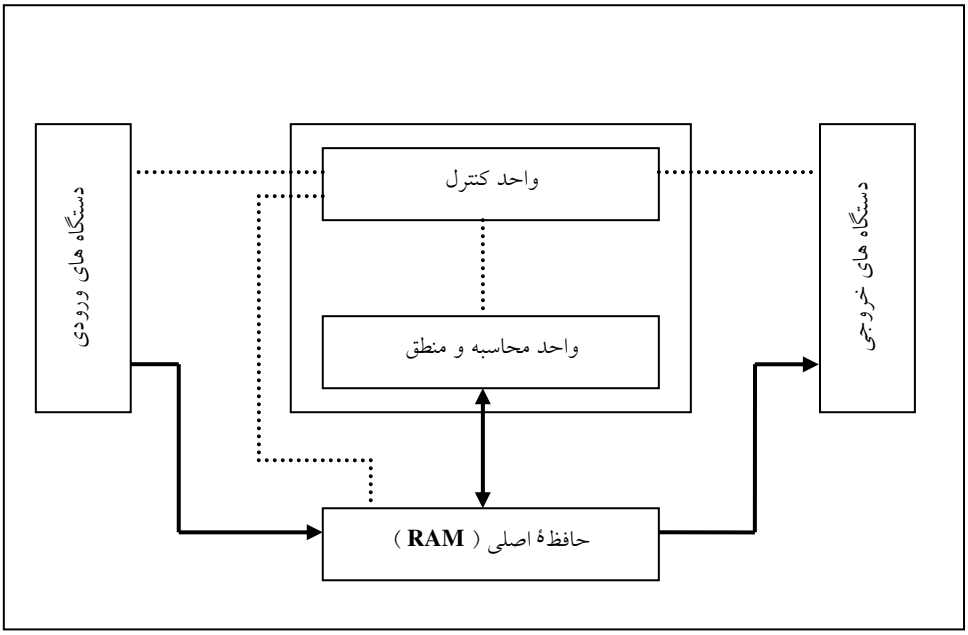
۴. همچنین هر کامپیوتر باید قادر باشد الگوها و راه حل‌ها و اطلاعات دیگر را در حافظه‌ای به‌صورت دائمی ذخیره سازد. لذا کامپیوتر دارای یک حافظهٔ ثانویه مثل هارد دیسک یا فلاپی و امثال آن است که می‌توان مطالبی را بر روی آن ذخیره کرد تا در دفعات بعدی که کامپیوتر شروع به‌کار می کند، بتوان آن اطلاعات را بازیابی نمود و مورد استفاده قرار داد.

۵. *اما حافظهٔ اصلی چیست؟* کامپیوتر نیازمند داشتن مکانی مشابه میزکار انسان، جهت قرار دادن اطلاعات و داده‌های در حال پردازش برروی آن، است. همچنین داده‌هایی که از صفحه کلید وارد می‌شود باید در حافظهٔ موقت ذخیره گردد و داده‌هایی که بر روی مانیتور نمایش داده می‌شود ابتدا باید در مکانی قرار داشته باشد تا سپس توسط تراشه‌های تصویری به نمایش در آیند. همهٔ این اطلاعات در حافظهٔ RAM قرار می‌گیرند. این حافظه دائمی نبوده، و با خاموش شدن کامپیوتر تمامی اطلاعات آن پاک می‌گردد. به RAM حافظهٔ اولیه نیز گفته می‌شود.

۶. از سوی دیگر در کامپیوتر باید مکانیسمی مشابه شبکه‌های عصبی انسان جهت کنترل صحت عملکرد قسمت‌های مختلف سیستم تعبیه گردد. همچنین فرمان شروع کار یا باز ایستادن از کاری باید توسط همین ارگان صادر گردد. لذا واحد پردازشگر مرکزی یعنیCPU از دو قسمت متفاوت به نام‌های ALU وCU تشکیل شده است که واحد محاسبه و منطق یا همانALU عملیات ریاضی و منطقی را انجام می‌دهد و واحد کنترل یا همانCU عملیات کنترل دستگاه‌ها و فرماندهی را انجام می‌دهد.

۷. بنابراین با توجه به مطالبی که بیان شد هر کامپیوتر را می‌توان به‌صورت طرح وار همانند شکل ۱-۱ در نظر گرفت. چنانکه در این شکل می‌بینید، برخی از قسمت‌ها دارای یک ارتباط یک طرفه و برخی دیگر دارای ارتباطی دو طرفه با یکدیگر هستند، که نشان دهندهٔ شکل جریان اطلاعات می‌باشد. از طرفی تمامی قسمت‌ها با خطوط نقطه‌چین با واحد CU ارتباط دارند. عملکرد این کامپیوتر طرح‌وار بدین‌صورت است که، داده‌های خام و دستورالعمل محاسبات از واحد ورودی با فرمان CPU به حافظهٔ اصلی وارد می‌شود. سپسCPU با انجام محاسبات لازم، مرحله به مرحله طبق دستورالعملِ از پیش تعیین شده جلو می‌رود تا در نهایت به حالت توقف برسد، و نتایج به‌دست آمده را در قسمت دیگری از حافظه قرار می‌دهد. در نهایتCPU نتایج به‌دست آمده را به واحد خروجی می‌فرستد تا برروی مانیتور نمایش داده شوند. در حین انجام تمامی این مراحل واحد CU نه تنها بر درست کارکردن تمامی دستگاه‌ها نظارت دارد، بلکه این واحدCU است که طبق دستورالعمل داده شده به کامپیوتر، به بخش‌های مختلف

می‌گوید که چه کاری را و در چه زمانی انجام دهند، کدام داده را مورد پردازش قرار دهند و یا نتایج را در کجای حافظه ذخیره کنند، و... . اما یک تفاوت اساسی میان عملکرد کامپیوتر و عملکرد انسان وجود دارد، به‌طوری که برای کارکردن کامپیوتر باید، راه‌حل مسئله از قبل توسط کاربر(انسان) به کامپیوتر داده شود و درکنار داده‌ها در RAM قرارگیرد تا CU بتواند از آن راه‌حل استفاده کند. بنابراین کامپیوترهای امروزی قادر نیستند به تنهایی راه حلی را بیابند بلکه کامپیوتر به یک انسان به‌عنوان کاربر نیازمند است تا نیازمندی‌های آن را برطرف سازد. کامپیوتر بدون حضور کاربر، همانند انسان با استعدادی است که سواد نداشته باشد، و لذا از استعداد خود هیچ بهره‌ای نمی‌تواند ببرد. بنابراین گرفتن دستورالعمل برای کامپیوتر همانند سوادمند شدن یک انسان با استعداد می‌ماند.



شکل ۱-۱ : چگونگی ارتباط بین اجزای اصلی کامپیوتر

۸. **اما مسلماً یک سؤال اساسی برای شما پیش آمده و آن اینکه چرا حافظهٔ ثانویه در شکل فوق ترسیم نشده است؟ جواب این سؤال بسیار روشن است، چرا که حافظهٔ ثانویه جزء واحدهای اصلی یک کامپیوتر به حساب نمی‌آید؛ چنانکه اگر ماشین حساب را به‌عنوان یک کامپیوتر کوچک در نظر بگیرید قسمتی مطابق با حافظهٔ ثانویه در آن نمی‌یابید؛ ولی ماشین حساب بدون حافظهٔ ثانویه به‌درستی کار می‌کند و این نشان‌دهندهٔ عدم نیاز مدل کامپیوتری ما به حافظهٔ ثانویه است. اما شما می‌توانید این حافظه را خودتان به شکل۱-۱ اضافه کنید. فکر می‌کنید حافظهٔ ثانویه چگونه رابطه‌هایی با اجزای شکل فوق باید داشته باشد؟**

۱-۴ : انواع کامپیوتر و کاربرد آنها

کامپیوترهای امروزی از نظر بزرگی، سرعت، دقت و کاربرد به دسته‌های مختلفی تقسیم می‌شوند:

۱. **سوپر کامپیوترها** (Super Computer)
۲. **کامپیوترهای بزرگ** (Main Frame)
۳. **کامپیوترهای متوسط** (Mini Computer)
۴. **کامپیوترهای کوچک یا رایانه شخصی** (Micro Computer or Personal Computer)

سوپر کامپیوترها بیشتر در مراکز نظامی و فضایی ابرقدرت‌ها مورد استفاده قرار می‌گیرند. قدرت محاسبات این کامپیوترها چنان گسترده و وسیع است که کشور فرانسه آزمایش‌های اتمی خود را به واسطهٔ این کامپیوترها شبیه‌سازی کرده و در محیط کاملاً مجازی شرایط انفجار و تخریب بمب‌های اتمی خود را محاسبه و بررسی می‌کند. همچنین در کشور امریکا در برخی مراکز بزرگ اطلاعاتی این کشور مثل NSA با این کامپیوترها، کوچکترین اطلاعاتی که توسط دستگاه‌های مخابراتی داخلی و خارجی آن کشور رد و بدل شود را مورد تجزیه و تحلیل قرار می‌دهد و در صورت لزوم رمزگشایی می‌کنند چنانکه ادعا می‌شود تا به حال هیچگونه پیام رمزی نبوده که توسط این ماشین رمزگشایی نشده باشد. لذا با توجه به کاربرد استراتژیک این کامپیوترها قدرت‌های بزرگ از دسترسی دیگر کشورها به این تکنولوژی جلوگیری می‌کنند.

کامپیوترهای بزرگ در مراکزی که با حجم اطلاعات بسیار زیاد سروکار دارند، به کار می‌روند، مثل دانشگاه‌ها و بانک‌ها و شرکت های هواپیمایی مادر و ... . کامپیوترهای متوسط در مراکز آموزشی و تجاری کوچکتر و کامپیوترهای شخصی در منازل، ادارات، سازمان ها و شرکت‌های دولتی و غیردولتی کاربرد وسیعی دارند. البته شاید نتوان یک مرز بندی دقیق بین کامپیوترهای متوسط و کامپیوترهای شخصی قائل شد.

کار در کلاس ۱-۱ :

در خصوص موارد زیر تحقیق کنید و در صورت امکان با تهیهٔ اسلاید، گزارشی از نتایج تحقیق خود را به کلاس ارائه دهید. مواردی که در زیر ذکر شده‌اند حتماً باید در درس آزمایشگاه کامپیوتر پوشش داده شوند، و یا آن که خود دانشجو به تحقیق در خصوص آن‌ها بپردازد.

۱. *قطعات یک کامپیوتر امروزی کدام ها هستند؟ نام قطعات لازم برای مونتاژ یک کامپیوتر را ذکر کنید و مختصری در خصوص کارکرد هر یک شرح دهید.*
۲. *در مورد انواع لوازم جانبی کامپیوتر و کاربرد آن ها شرح دهید.*
۳. *ساختار داخلی یک پردازنده اعم از رجیستر ها، انباره، واحد محاسبه و منطق و... شرح دهید.*
۴. *در خصوص نحوه ی انجام عملیات در پردازنده ها تحقیق کنید.*
۵. *اندکی در مورد برنامه های* **setup** *و سیستم عامل شرح دهید.*

۱–۵ : بررسی مفهوم الگوریتم

اولین مقوله‌ای که در عرصهٔ علم کامپیوتر با آن برخورد می‌کنید " *الگوریتم* " می‌باشد. اما برای آنکه مفهوم الگوریتم را خوب و دقیق به شما انتقال دهیم و بتوانیم تعریفی دقیق از آن ارائه کنیم نیازمند پاسخگویی به یک سؤال مهم هستیم.!

فرض کنید رباتی در اختیار داریم که به‌وسیلهٔ دستورات خاصی که برای آن قابل‌فهم است، قادر است دستورالعمل انجام کاری را از ما دریافت کند و آن کار را مطابق دستورالعمل داده شده انجام دهد. حال سؤال این است که چگونه می‌توانیم با بهره‌گیری از امکانات این ربات مسائلی که در روزمره با آنها روبرو می‌شویم را انجام دهیم؟ و یا به عبارت دیگر با چه روشی به ربات خود بفهمانیم که کار مشخصی را چگونه باید انجام دهد!؟

این سؤال را می‌توانیم در حالت کلی به‌صورت زیر بیان کنیم :

چگونه مسائل را به واسطهٔ کامپیوتر حل کنیم ؟

در جواب به این سؤال باید اذعان داشت هنگامی که مسئله‌ای را به ما می‌دهند مراحل زیر را در جهت حل مسئله باید انجام دهیم:

۱. *شناخت مسئله* : منظور آنکه، بـفـهمیم مسئله چه داده‌هایی را در اختیار ما قرار داده است و چه مجهول‌هایی را از ما می‌خواهد!؟

۲. *تحلیل راه‌حل* : منظور یافتن ارتباطی بین داده‌های مسئله و مجهول‌های آن است برای این منظور دو روش کلی برای عموم مسائل به کار گرفته می‌شود:

- *شیوهٔ مکاشفه‌ای*
- *شیوهٔ الگوریتمی*

مقصود از شیوهٔ مکاشفه‌ای این است، که به جای تفکر منطقی و سیستماتیک، به تفکرات جانبی روی آورده و به کوتاه‌ترین و عملی‌ترین راه‌حل برای حل مسئله دست پیدا کنید. به‌عنوان مثال هنگامی که به‌واسطهٔ فرمول‌گرایی در حل برخی مسائل ریاضی به جای طی‌کردن مراحل علمی و دقیق ریاضی آن، مسئله را با برخی ترفندها و به قول معروف از راه‌حل‌های تستی حل می‌کنید از این روش بهره گرفته‌اید. به‌عنوان مثال تمرین زیر را ابتدا خودتان سعی کنید حل کنید و سپس پاسخ کتاب و تحلیل آن را به دقت مطالعه نمایید. هدف از حل این تمرین این است، که شما دریابید مسائلی را که با آن روبرو می‌شوید اصولاً به چه روشی حل می‌کنید. چه بسا به‌طور ناخودآگاه روش علمی معتبری را برای حل مسائل به‌کار می‌برید.!؟

کار در کلاس ۱-۲ :

تعداد سی کشتی گیر در یک دوره مسابقات یک حذفی شرکت دارند. معین کنید در این دوره مسابقات چند مسابقهٔ کشتی انجام می‌شود؟ راه‌حل خود را شرح دهید.

راهنمایی : (منظور از یک حذفی بودن مسابقات این است که دو نفر کشتی می گیرند و بازنده از دور مسابقات حذف می شود.)

یک راه‌حل منطقی این مسئله می‌تواند این باشد که از روش مسائل ترسیمی هندسه جلو برویم، بدان معنا که ابتدا فرض کنیم مسئله حل شده است، بنابراین با توجه به آنکه فرضیات زیر را نیز داریم:

الف- فقط یک برنده و ۲۹ بازنده داریم.

ب- هر مسابقه فقط یک بازنده دارد.

ج- هر بازنده نیز فقط یک باخت دارد.

مشخص می‌شود که تعداد مسابقات با تعداد بازنده‌ها برابر است بنابراین ۲۹ مسابقه انجام شده است. به حل کردن مسائل بدین روش، شیوهٔ مکاشفه‌ای گفته می‌شود. حتی حل این مسئله می‌تواند بدین‌صورت تعمیم پیدا کند که تعداد مسابقات یک حذفی برای n کشتی گیر برابر n-۱ مسابقه می‌باشد.(فرمول گرایی)

اما شاید راه حل شما به این صورت باشد که گفته باشید: ابتدا ۱۵ مسابقه به‌صورت دوبه‌دو انجام شده و ۱۴ نفر از ۱۵ نفر باقی‌مانده در ۷ مسابقه شرکت می‌کنند و یک نفر نیز بدون مسابقه دادن، به قید قرعه بالا می‌آید، که در نهایت هشت نفر باقی می‌مانند. این ۸ نفر در ۴ مسابقه شرکت کرده و ۴ نفر بالا می‌آیند و برنده‌ها در ۲ مسابقه دو به دو شرکت کرده و فینالیست‌ها را مشخص می‌کنند و در نهایت ۱ بازی فینال انجام خواهد شد. بنابراین :

۱۵ + ۷ + ۴ + ۲ + ۱ = ۲۹

جمعاً ۲۹ مسابقه انجام شده است.

به شیوهٔ حل اخیر شیوهٔ الگوریتمی گفته می‌شود که در ادامه به اصول آن و مزایای استفاده از این روش بیشتر خواهیم پرداخت. فعلاً در همین حد بدانید که با شیوهٔ الگوریتمی می‌توان بسیاری از مسائل پیچیده و بزرگ را به‌راحتی به واسطهٔ کامپیوتر پیادسازی و حل کرد که شاید راه‌حل فرمولی و مکاشفه‌ای هم نداشته باشند. در کار در کلاس بعدی هدف بر آن است، که شما را بیشتر با شیوهٔ الگوریتمی آشنا سازیم.

کار در کلاس ۱-۳ :

فرض کنید مدلی که از کامپیوتر به‌عنوان ربات در ابتدای بخش ۱-۳ مطرح کردیم در اختیار شما قرار داده شده است. با صرف‌نظر از اینکه چگونه مفاهیم و دستورات را به ربات انتقال می‌دهیم، نحوهٔ استفاده از تلفن عمومی (فقط گرفتن یک شماره تلفن به صورت موفقیت آمیز) را برای ربات شرح دهید.

راهنمایی : برای این منظور یک سلسله دستوراتی را به زبان فارسی برای کامپیوتر بنویسید.

شاید راه‌حلی که شما به‌صورت تعدادی دستورالعمل ارائه نموده‌اید به‌صورت زیر باشد:

۱. *شروع کن.*
۲. *گوشی تلفن را بردار.*
۳. *یک سکه در تلفن بینداز.*
۴. *صبر کن تا صدای بوق آزاد را بشنوی.*
۵. *شماره مورد نظر را بگیر.*
۶. *پایان عملیات.*

در تهیهٔ یک دستورالعمل باید به نحوهٔ اجرای آن توجه خاص داشت. منظور آنکه، در اجرای دستورالعمل هیچگونه سؤال و ابهامی برای مجری آن نباید پیش‌آید، و اجرای مراحل آن نباید بستگی به برداشت‌های متفاوت مجری داشته باشد. ضمناً ترتیب عملیات باید معیّن باشد، و دارای شروع و خاتمهٔ مشخصی باشد. چرا که دستورات یک دستورالعمل توسط کامپیوترهای سریال که در اختیار ما قرار دارند، دستور به دستور به ترتیب از نقطهٔ آغاز تا پایان اجرا می‌گردد. لذا نباید انتظار داشته باشید که کامپیوترهای سریال بتوانند چندین عمل را به‌طور همزمان انجام دهند. البته کامپیوترهای موازی وجود دارند که قادرند با چندین پردازنده به‌طور همزمان کارهای متفاوتی را به‌صورت همزمان انجام دهند که از بحث ما به‌کلی خارج است. فراموش نکنید که مبحث کامپیوترهای موازی با مقولهٔ Multi Tasking بودن سیستم‌عامل‌هایی چون ویندوز و لینوکس به‌کلی متفاوت است در این خصوص در کتاب *"آزمایشگاه کامپیوتر"* مطالب روشنگری بیان شده است.

دستورالعملی با ویژگی‌های فوق یک الگوریتم نامیده می‌شود. توجه داشته باشید که الگوریتم مقوله‌ای خاص علم کامپیوتر نیست و در بسیاری از علوم دیگر نیز کاربرد دارد. اما گفته می‌شود که علم کامپیوتر، علم الگوریتم‌ها است. الگوریتم به معنای شیوه و روش حل یک مسأله است و از نام دانشمند بزرگ و گرانقدر ایرانی *"محمد بن موسی خوارزمی"* گرفته شده است، چرا که وی در آغاز قرن سوم هجری در کتاب *"جبر و مقابله"* برای حل مسائل با ابداع معادلات جبری شیوه و روش حل جدیدی ارائه کرد و علم جبر را پایه‌گذاری نمود. از آنجایی که در پردازش داده‌ها و محاسبات ریاضی ابداع شیوه و روش حل مناسب، و یا به عبارت دیگر طراحی یک الگوریتم خوب، نقش محوری و کلیدی در حل آن مسأله دارد، علم کامپیوتر را علم الگوریتم‌ها نامیده‌اند.

حال به ارائهٔ تعریف دقیقی از الگوریتم می‌پردازیم.

تعریف الگوریتم:

الگوریتم عبارت است از تعدادی دستورالعمل پشت سرهم که مراحل مختلف یک کار را به زبان دقیق و با جزئیات کافی بیان نماید و در آن ترتیب مراحل و خاتمه پذیر بودن عملیات کاملاً مشخص باشد.

البته در فصول بعدی ویژگی‌های دیگری را برای الگوریتم بیان خواهیم کرد و آن را در دو نوع الگوریتم‌های ساخت یافته و غیر ساخت یافته دسته‌بندی خواهیم نمود، اما فعلاً ویژگی‌هایی را که برای الگوریتم در تعریف فوق برشمردیم، مورد تحلیل قرار می‌دهیم.

۱ – زبان دقیق با جزئیات کافی

اگر از یک جمله، که در مرحله‌ای از یک الگوریتم آمده است، برداشت‌های متفاوتی شود و یا به سبب مهم بودن، سؤال برانگیز باشد گوییم، بیان آن جمله دقیق نیست. مثلاً در الگوریتم بالا گفته نشده چه نوع سکه‌ای را در دستگاه تلفن بیندازا؟ بنابراین از آنجایی که بیان جملات به‌صورت دقیق، به واسطهٔ زبان فارسی یا انگلیسی بسیار مشکل است؛ از زبان‌های به اصطلاح صوری مثل زبان‌های برنامه‌نویسی استفاده می‌شود. همچنین جهت بیان الگوریتم‌ها، تعدادی قوانین و قواعد را به‌صورت استاندارد در آورده‌اند که به واسطهٔ آن‌ها می‌توان الگوریتم‌ها را به‌صورتی فراگیر و همه فهم بیان نمود، به‌طوری که الگوریتم‌ها را می‌توان با رسم کارنما (فلوچارت) نمایش داد.

به‌طور کلی روش‌های زیر برای بیان الگوریتم وجود دارد:

۱. استفاده از زبان‌های طبیعی مثل فارسی و انگلیسی و ...
۲. استفاده از نمودار
۳. استفاده از زبان‌های برنامه‌نویسی
۴. استفاده از شبه‌کد (Pseudo code)

در این فصل از روش اول و در فصول آتی از روش‌های دوم و سوم استفاده می‌کنیم و در آنجا با این روش‌ها بیشتر آشنا خواهید شد. اما در خصوص روش شبه‌کد باید گفت؛ در این روش با کمی دخل وتصرف در دستورات یک زبان برنامه‌نویسی الگوریتم را با آزادی عمل بیشتری بیان می‌کنند. چنانکه ممکن است دستوراتی را در شبه‌کد به‌کاربرند که اصلاً در آن زبان وجود نداشته باشد ولی منظورشان را به روشنی برساند، و چگونگی پیاده‌سازی آن را به عهدهٔ فرد برنامه‌نویس می‌گذارند. در این خصوص در کتب طراحی الگوریتم مطالب مفیدی یافت می‌شود.

۲ – ترتیب مراحل

مسلماً ترتیب انجام کارها در هر الگوریتمی مهم است. همان‌طور که پیشتر گفته شد کامپیوترهایی که شما با آنها سروکار دارید به‌صورت سریال عمل می‌کنند. یعنی تنها قادر به انجام یک عمل ریاضی یا منطقی در یک سیکل زمانی خود هستند، بنابراین کامپیوتر می‌تواند عملیات یک الگوریتم را پشت‌سرهم و به ترتیب انجام دهد.

۳ – خاتمه پذیر بودن عملیات

شرط یا شروط خاتمهٔ عملیات باید در الگوریتم به‌طور صریح یا ضمنی بیان شود. به‌خصوص وقتی که با انجام یکسری عملیات تکراری روبرو هستیم حتماً باید شرط پایان حلقهٔ تکرار به‌طور صریح بیان شود. مثلاً در مثال قبلی اگر تلفن خراب باشد و بوق آزاد نزند ربات ما باید تا ابد صبر کند!!!

حال با توجه به توضیحات بالا *الگوریتم تلفن زدن* را یک بار دیگر بازنویسی می‌کنیم. برای اینکه الگوریتم بدون ابهام باشد، فرض می‌کنیم که حداقل یک تلفن سالم در شهر وجود دارد.

" **الگوریتم اصلاح شدهٔ تلفن زدن** "

۱. *شروع کن.*
۲. *گوشی تلفن را بردار و یک سکهٔ سالم در تلفن بینداز.*
۳. *حداکثر ۳۰ ثانیه صبر کن تا صدای آزاد شدن خط را بشنوی اگر خط آزاد شد به مرحلهٔ ۵ برو در غیر این صورت به مرحلهٔ ۴ برو.*
۴. *گوشی را سرجایش بگذار، و اگر سکه پس داده شد آن را بردار. اگر دفعه سومی است که این مرحله را اجرا می کنی (منطقی است که نتیجه بگیرید تلفن خراب است) به مرحلهٔ ۶ برو و در غیر این صورت به مرحلهٔ ۲ باز گرد و مراحل را تکرار کن.*
۵. *شماره موردنظر را بگیر و به مرحلهٔ ۷ برو.*
۶. *تلفن دیگری در شهر پیدا کن و اجرای الگوریتم را از مرحله ۲ تکرار کن.*
۷. *پایان.*

در ادامه به بیان یک الگوریتم عددی توسط زبان طبیعی می‌پردازیم و بحث این فصل را خاتمه می‌دهیم. در فصل آتی با رسم کارنما آشنا شده و به بیان ترسیمی الگوریتم‌هایی، یعنی کارنما می‌پردازیم.

کار در کلاس ۱-۴ :

الگوریتمی بنویسید که معدل سه عدد ۱۵ و ۱۷/۴۵ و ۱۹/۱۰ را حساب کند.

ما الگوریتم را در حالت کلی‌تری حل می‌کنیم. فرض کنید نمرات درسی یک دانش‌آموز را در متغیرهای x و y و z و معدل آنها را در متغیر M قرار می‌دهیم. !؟ فقط به این نکته توجه داشته باشید که منظور از U ← V این است، که مقدار متغیر U را در متغیر V قرار بده.

مثال ۱-۱ : " الگوریتم محاسبهٔ معدل سه عدد "

۱. *شروع کن.*

۲. *مقدار متغیرهای* x *و* y *و* z *را از کاربر در یافت کن.*

۳. *مجموع آن ها را به‌صورت زیر حساب کن و در* S *قرار بده :*

(S ← x + y + z)

۴. *مقدار معدل را به‌صورت زیر محاسبه کن و در* M *قرار بده :*

(M ← S / 3)

۵. *مقدار معدل که در متغیر* M *قرار دارد را چاپ کن.*

۶. *پایان.*

با توجه به معماری وان نویمان که در کامپیوترهای امروزی لحاظ شده است، ویژگی مهم زبان های برنامه‌نویسی دستوری، ***متغیرها، دستورالعمل‌های انتساب، ساختارهای تصمیم گیری و حلقه‌های تکرار*** است که در فصول آتی به تک تک این موارد می‌پردازیم.

۱-۶: تمرین

۱. شکل کلی از کامپیوتر رسم کرده و کار هر یک از بخش‌های آن را شرح دهید؟
۲. الگوریتم را تعریف کنید و ویژگی‌های آن را برشمرید؟
۳. الگوریتمی جهت تعویض چرخ پنچر یک خودرو بنویسید؟
۴. الگوریتمی جهت طرز تهیهٔ کیک زیر بنویسید؟

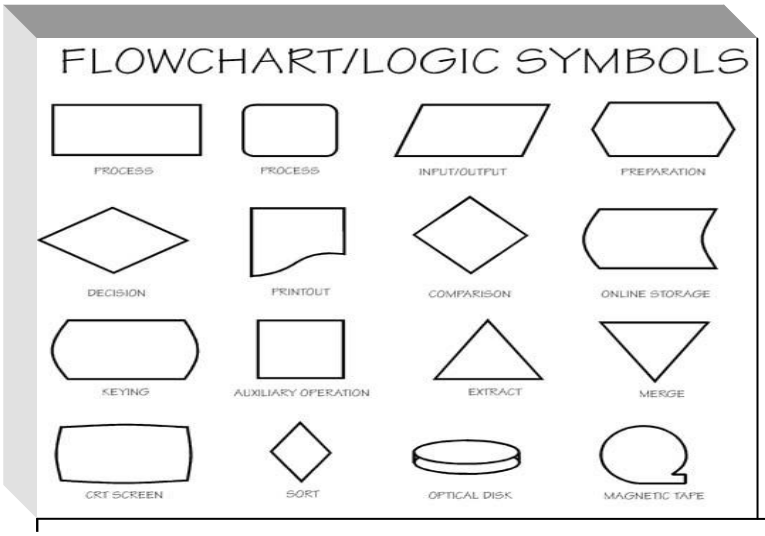
مواد لازم عبارت است از :

- روغن دو فنجان،
- آرد شش فنجان،
- شکر دو فنجان،
- شیر دو فنجان،
- تخم مرغ شش عدد،
- وانیل نصف قاشق،

دستورالعمل: روغن را در ظرفی بریزید و شکر را به آن اضافه کنید، تخـم‌مرغ‌ها را در آن بشکنید و خوب مخلوط کنید، سپس آرد و شکر را به آن اضافه کنید و به هم بزنید تا حالت کشش پیدا نماید. وانیل را اضافه کنید و مجدّداً به هم بزنید. ظرف مخصوص کیک را چرب کنید کمی آرد به آن بپاشید و مـخلوط را در آن هم بریزید. سپس ظرف را به مدت ۳۰ دقیقه در حرارت ۱۲۰ درجه قرار دهید.

آیا می‌توانید الگوریتم فوق را به ابتکار خود به‌صورت یک نمودار نمایش دهید؟

۵. الگوریتمی برای تهیهٔ چای به‌وسیلهٔ یک سماور برقی بنویسید؟
۶. الگوریتمی بنویسید که شعاع دایره‌ای را بگیرد و محیط و مساحت آن را چاپ کند؟
۷. الگوریتمی بنویسید که درجهٔ حرارت را برحسب سانتی‌گراد (C) بگیرد و به فارنهایت (F) تبدیل کند؟ برای حل الگوریتم از فرمول زیر استفاده کنید: $F = 9.5*C + 32$
۸. الگوریتمی بنویسید که طول و عرض مستطیلی را دریافت کرده و محیط و مساحت آن را چاپ کند؟
۹. الگوریتمی بنویسید که قاعده و ارتفاع مثلثی را دریافت کرده و مساحت آن را چاپ کند؟
۱۰. الگوریتمی بنویسید که دو مقدار را بخواند و سپس مقدار بزرگتر را چاپ کند؟
۱۱. الگوریتمی بنویسید که عددی را خوانده و قدرمطلق آن را چاپ کند؟
۱۲. الگوریتمی برای پیدا کردن اولین جمله از دنبالهٔ فیبوناچی که از ۱۰۰۰ بزرگتر است به‌دست آورید؟ آیا می‌توانید این مسئله را از طریق مکاشفه‌ای حل کنید و برای آن یک فرمول عمومی ارائه کنید!؟ پاسخ خود را توضیح دهید.



فصل دوم

آشنایی با مقدّمات زبان روندنما

اهداف فصل و چکیده مطالب :

۱. آشنایی با قواعد مقدماتی رسم کارنما
۲. آشنایی با تعریف متغیر و انواع عبارات در الگوریتم
۳. به‌کارگیری عبارات رابطه‌ای (شرطی) در الگوریتم‌ها
۴. آشنایی با مکانیسم‌های تکرار و تصمیم در الگوریتم
۵. آشنایی با تقدم عملگرها
۶. ارزیابی عبارات محاسباتی(جبری) بدون پرانتز
۷. خوانا ساختن عبارات محاسباتی به وسیلهٔ پرانتزگزاری
۸. آشنایی با چند تابع ریاضی پرکاربرد
۹. یافتن مهارت در چگونگی ساخت الگوریتم

۲-۱ : آشنایی با قواعد مقدماتی رسم کارنما

در فصل گذشته با کلّیّاتی در خصوص ویژگی‌های الگوریتم آشنا شدیم، و الگوریتم‌هایی را توسط زبان فارسی نوشتیم. در این فصل به دنبال یافتن روش استانداردی برای بیان الگوریتم‌ها هستیم. معمولاً الگوریتم‌ها را توسط نمودارهای خاصی به نام کارنما (یا فلوچارت[1]) تحت یکسری قوانین استاندارد نشان می‌دهند. این بدان علت است که با رسم کارنما، هم می‌توان با به‌کارگیری قوهٔ بصری افراد مفاهیم را به سرعت انتقال داد و در نتیجه بهره‌وری مخاطب را در یادگیری مطالب بالا برد؛ و هم اینکه با بهره‌گیری از قوانین استاندارد برای رسم کارنما، آن را برای هر فردی که با قوانین کارنما آشنایی داشته باشد قابل فهم کنیم. حتی برای شخص نویسندهٔ الگوریتم نیز پیگیری الگوریتم و بررسی درستی آن به‌واسطهٔ نمودار به مراتب راحت‌تر است. البته ما در این کتاب به دلایلی برخی از استانداردهای رسم کارنما را رعایت نخواهیم‌کرد که در جای خود شرح تک‌تک آنها داده خواهدشد. با توجه به نکاتی که در این فصل بیان‌می‌شود شما قادر خواهید بود بسیاری از الگوریتم‌های ساده را طراحی‌کنید و برای آنها کارنما رسم نمایید. اما در فصول آتی که مسائل پیچیده‌تری را مطرح می‌سازیم رفته‌رفته با مباحث پیشرفته‌تری از الگوریتم‌ها روبرو خواهید شد، که برای سهولت‌کار در ترسیم کارنما برای اینگونه الگوریتم‌ها، عناصری را به‌صورت یک قراردادِ منحصر به این کتاب [و نه به صورت استاندارد] بیان می‌کنیم. به‌عنوان مثال در حلقه‌های تکرار از یک سری جعبه‌های فشنگی شکل استفاده خواهیم‌کرد، و این بدان جهت است که بتوانیم شرایط ساخت‌یافتگی را در مورد یک الگوریتم خاص مورد بحث و بررسی قرار دهیم.

خوشبختانه مجموعهٔ قواعد زبان کارنما بسیار ساده و پیش پا افتاده است چنان که ظرف کمتر از یک ساعت می‌توان تمامی آنها را فراگرفت. اما آنچه که مشکل می‌نماید یادگیری چگونگی به‌کاربردن این قواعد برای ساخت یک الگوریتم است که تنها و تنها با تمرین و ممارست در این زمینه به‌دست می‌آید. الگوریتم نویسی و یافتن راه‌حل برای یک مسئله بیش از آنکه به آشنایی فرد با قواعد این زبان بستگی داشته باشد به هنر و قریحهٔ فردی افراد بستگی دارد. چنانکه می‌توان گفت علم کامپیوتر و طراحی الگوریتم یک هنر است. البته این بدین‌معنا نیست که برخی از افراد به کلّی در الگوریتم نویسی ناتوان هستند و برخی دیگر به‌طور ذاتی از آن برخوردار، بلکه منظور این است که، ممکن است راه‌حل‌های متفاوتی برای یک مسئله از سوی افراد مختلف ارائه شود ولی فقط یکی از آن راه‌حل‌ها بهترین راه‌حل به حساب می‌آید و بیشترین ارزش را دارد. مسلماً چنین راه‌حلی را فردی ارائه خواهد کرد که نهایت ذوق و قریحه را در طراحی الگوریتم خود به‌کار برده و به همهٔ جوانب مسئله اندیشیده باشد.

بنابراین در این فصل با در نظر گرفتن آنچه که در مورد ویژگی‌های الگوریتم در فصل قبل یادگرفتیم، قدم به قدم، برای قسمت‌های متفاوت الگوریتم‌هایی که تا اینجا نوشتیم، نمادهایی را در نظر می‌گیریم تا به کمک آن نمادها بتوانیم الگوریتم‌های خود را به‌صورت نموداری نشان دهیم. دوباره متذکّر می‌شویم که حل تمامی تمرینات فصول مربوط به الگوریتم نویسی بسیار مهم است، چرا که بسیاری از نکات مهم در تمرینات آموزش داده‌شده است، تا خوانندگان با تلاش خود و به واسطهٔ راهنمایی‌های‌کتاب به این نکات دست پیدا کنند و در نتیجه تأثیر مطالب در ذهن آنها ماندگارتر باشد. همچنین تمریناتی که دارای پیوستگی مطلب هستند با دقت مرتب شده‌اند.

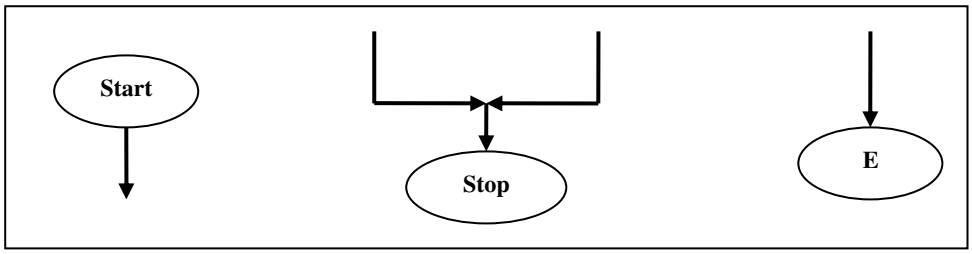
---

1) flowchart

جعبه‌های شروع و پایان

همانگونه که دیدید در هر الگوریتم یک نقطهٔشروع و یک نقطهٔپایان وجوددارد. در استانداردهای انستیتوی ملی استاندارد امریکا (H.N.S.I) که تنها مرجع قانون‌گذار در این زمینه است پیشنهاد شده از نماد بیضی برای نشان دادن نقاط شروع و پایان الگوریتم‌ها استفاده شود، ما نیز از همین روش تبعیت می‌کنیم.

برای دکمهٔ شروع الگوریتم از یک *بیضی مدور* با متن **Start** یا (**S**) و برای دکمهٔپایان نیز از یک بیضی به همان شکل اما با متن**Stop** یا **End** یا (**E**) استفاده می‌شود.



شکل ۲-۱ : دکمه‌های شروع و پایان

نکات قابل توجه

۱. در هر الگوریتم تنها یک نقطهٔ شروع وجود دارد، چرا که اگر نقطهٔ شروع بیش از یکی باشد الگوریتم دارای ابهام خواهد بود. ولی ممکن است در الگوریتم چندین نقطهٔ پایان وجود داشته باشد، در نتیجه می‌توان در هر کجا که نیاز باشد یک نقطهٔ پایان در نظر گرفت. همچنین می‌توانید تنها یک نقطهٔ پایان در نظر بگیرید و چندین پیکان ورودی به آن وارد کنید.

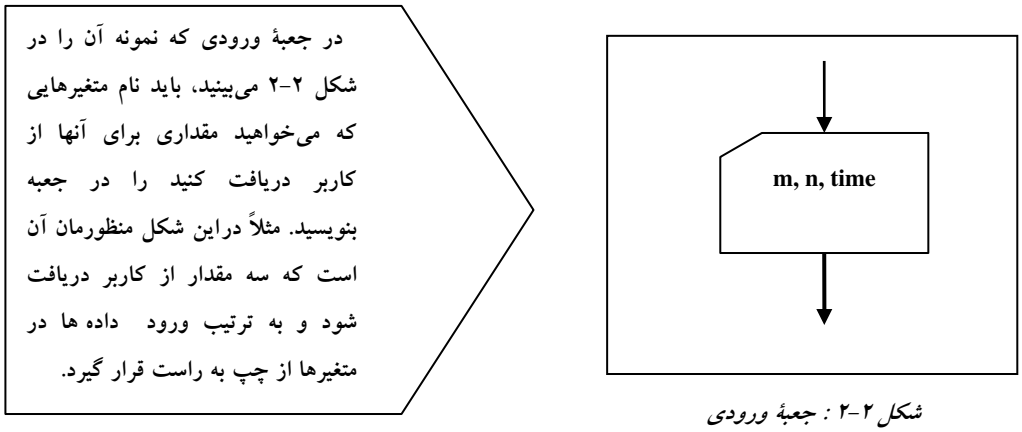
۲. در برخی از کتب به‌جای شکل بیضی برای دکمه‌های شروع و پایان از دایره استفاده می‌شود که مسلماً تفاوت چندانی در اصل موضوع ندارند. ما نیز در برخی از مثال‌ها از دایره استفاده کرده‌ایم. چنانکه بعداً خواهید دید در این کتاب از شکل *بیضی تخت*، برای جعبه‌های تصمیم استفاده شده است. لذا خوانندگان عزیز نباید شکل بیضی مدور که برای دکمه‌های شروع و پایان استفاده شده را با شکل بیضی تخت که برای جعبه‌های تصمیم استفاده شده است، اشتباه بگیرند.

۳. از هر مرحله می‌توان با یک پیکان خروجی به هر یک از مراحل قبلی کارنما، به غیر از دکمهٔ شروع بازگشت. اما از دکمه‌های پایانی نمی‌توان به هیچ مرحلهٔ دیگری رفت.

۴. مقصود از پیگیری (یا آزمایش) الگوریتم آن است که از دکمهٔ شروع، اجرای الگوریتم را آغاز کرده و مطابق با جهت پیکان خروجی از هر مرحله به مرحلهٔ بعدی بروید. تا اینکه در نهایت به یک دکمهٔ پایانی برسید و اجرای الگوریتم متوقف گردد. در این حالت اگر الگوریتم نتایج مورد نظر را به‌دست داده باشد الگوریتم صحیح است در غیر این صورت باید مراحل الگوریتم را مورد بازبینی قراردهید و دوباره آن را پی‌گیری نمود. به این عمل *آزمایش الگوریتم* یا *اثبات درستی الگوریتم* نیز گفته‌می‌شود.

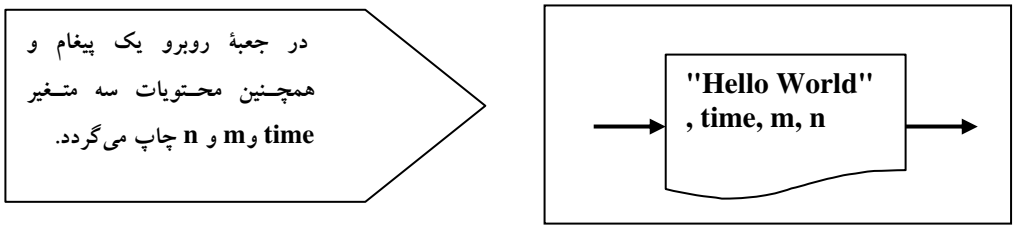
ورود و خروج اطلاعات

فرض کنید می‌خواهید از کاربر داده‌هایی را دریافت کنید و در متغیرهایی ذخیره نمایید. در این صورت از جعبه‌هایی مانند شکل ۲-۲ باید استفاده کنید. این جعبه ما را به یاد کارت منگنه که یکی از وسایل متداول ورود اطلاعات در گذشته به شمار می‌رفته می‌اندازد.



شکل ۲-۲ : جعبهٔ ورودی

همچنین جهت نمایش پیام‌ها و نتایج به‌دست آمده؛ از جعبه‌هایی مانند شکل ۲-۳ استفاده می‌کنیم که شبیه یک تکه کاغذ می‌باشد. نکته قابل توجه آنکه اگر می‌خواهید پیامی چاپ گردد باید آن را در بین دو علامت نقل‌قول به شکل " " قرار دهید. همچنین اگر متغیری را در داخل جعبهٔ خروجی قرار دهید محتویات آن متغیر چاپ می‌شود نه نام آن متغیر.



شکل ۲-۳ : جعبه خروجی

در جعبه‌های خروجی نیز اطلاعات به ترتیب از چپ به راست چاپ می‌گردد. به‌عنوان مثال در شکل ۲-۳ ابتدا پیام "Hello World" چاپ می‌گردد و سپس محتویات سه متغیر time و m و n چاپ می‌شود.

تذکر: در برخی از کتب؛ هم برای جعبه‌های ورودی و هم برای جعبه‌های خروجی، از جعبه‌هایی به شکل متوازی الاضلاع استفاده می‌کنند و در آنها از لفظ "بخوان" و "بنویس" برای نشان دادن عمل خواندن (یا نوشتن) استفاده می‌شود.

متغیرها و جعبهٔ انتساب

همانگونه که در جبر از متغیرها در عبارات محاسباتی(جبری) استفاده می‌کنید در الگوریتم‌ها نیز می‌توانید از متغیرها برای این منظور استفاده کنید. هر متغیر با یک یا چند حرف لاتین به‌عنوان نام متغیر شناسایی می‌شود، و در مکان‌های مختلف می‌توان با استفاده از نام متغیر به محتویات آن دست یافت. می‌توان چنین فرض کرد که متغیر دارای مخزنی است که قادریم در آن مخزن، مقادیری از اعداد حقیقی را جای دهیم، و با داشتن نام متغیر می‌توانیم به سراغ مخزن آن برویم و به محتویات داخل مخزن دست پیدا کنیم. به عبارت دیگر متغیر نامی است برای محتویات داخل مخزن خود. لذا ممکن است در طول الگوریتم محتویات مخزن به دفعات تغییر کند اما نام متغیر از اول تا پایان ثابت می‌ماند. اولین نکته‌ای که باید در این خصوص به خاطر بسپارید روش نامگذاری صحیح متغیرها است، به‌طوری که در نامگذاری متغیرها باید قواعد زیر را رعایت کنید:

۱. در نامگذاری متغیرها می‌توانید از حروف کوچک و بزرگ لاتین و ارقام و خطِ زیر استفاده کنید. بنابراین کاراکترهای مجاز در نامگذاری متغیرها عبارتند از : **a,b,c,…,z,A,B,C,…,Z,0,1,2,…9,_**

۲. حرف اول نام یک متغیر نمی‌تواند رقم باشد. گرچه استفاده از خط زیر در نخستین کاراکتر نام متغیر بلامانع است ولی پیشنهاد می‌شود که حرف اول نام یک متغیر حتماً یک حرف لاتین باشد. این مسئله در برخی زبان‌های برنامه‌نویسی ضروری است.

۳. بهتر است نام متغیرها با معنا و توصیف‌گر محتویات خود باشند تا در پیگیری الگوریتم با مشکلی برخورد نکنید.

۴. طول نام یک متغیر می‌تواند از یک حرف آغاز شود و با ترکیب حروف و ارقام مختلف گسترش یابد. اما در زبان‌های برنامه‌نویسی اصولاً طول نام متغیرها از یک مقدار حداکثر نمی‌تواند بیشتر باشد. لذا ما نیز به تبعیت از این قاعده در الگوریتم‌های خود طول نام متغیرها را بیش از ۳۲ کاراکتر نمی‌گیریم.

۵. با توجه به آنچه گفته شد نام‌های زیر برای متغیرها مجاز شمرده می‌شوند:

Time , m , n435p , odd_number_654j

۶. فراموش نکنید که دو متغیر زیر با یکدیگر متفاوت هستند:

M , m

پس از تعریف کردن یک متغیر می‌بایستی یا در یک جعبهٔ ورودی مقداری برای آن از کاربر دریافت کنید، و یا آنکه توسط یک دستور انتساب مقداری را به آن نسبت دهید. شکل کلی دستور انتساب به‌صورت زیر است:

(یک متغیر) یا (یک عبارت) یا (یک عدد حقیقی) ← متغیر

در سمت چپ پیکان همواره تنها یک متغیر قرار می‌گیرد، بدین‌معنا که مقدار سمت راست در متغیر سمت چپ قرار داده شود. اما در سمت راست ممکن است از یک عدد و یا نام یک متغیر دیگر استفاده کنیم. مانند نمونه‌های زیر :

Total ← 17 در این مثال مقدار ۱۷ در متغیر سمت چپ قرار می‌گیرد.

m2 ← m1 در این مثال محتویات متغیر **m1** در متغیر **m2** قرار می‌گیرد.

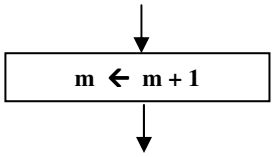
ولی گاهی هم پیش می‌آید که عبارتی را در سمت راست دستور انتساب قرار می‌دهیم. این عبارات بر دو گونه هستند. یا رشته‌ای از کاراکترها هستند، که عین عبارت رشته‌ای [که باید بین دو علامتِ " محصور شده باشد] در متغیر قرار می‌گیرد، مثل: C ← " Hello World "

و یا آنکه با یک عبارت محاسباتی(جبری) که ترکیبی از علائم ریاضی و نام تعدادی متغیر می‌باشد سروکار داریم. اگر یک عبارت جبری در سمت راست یک دستور انتساب قرار گیرد مقدار عبارت جبری محاسبه شده و حاصل آن در متغیر سمت چپ علامت پیکان قرار داده می‌شود. تنها نکتهٔ قابل توجه در مورد عبارات محاسباتی این است، که در این عبارات، به‌جای علامت ضرب (×) از علامت ستاره (*) و به‌جای علامت تقسیم (÷) از علامت ممیز (/) استفاده می‌کنیم. بنابراین باید، برای نشان دادن اعداد اعشاری، از نقطه به‌عنوان ممیز عدد استفاده کنیم، نه علامت ممیز. به‌عنوان مثال عبارت زیر را در نظر بگیرید:

$Delta \leftarrow B^2 - 4*A*C$

در دستور فوق ابتدا حاصل عبارت جبری سمت راست محاسبه شده و در متغیر **Delta** قرار داده می‌شود.

دستورات انتساب از هر کدام یک از اشکال فوق که باشند می‌بایستی در داخل جعبه‌های مستطیلی شکلی به نام جعبه‌های انتساب قرارگیرند. به‌عنوان مثال در شکل زیر در داخل جعبهٔ انتساب دستوری قرار گرفته که محتویات متغیر **m** را یک واحد افزایش می‌دهد:



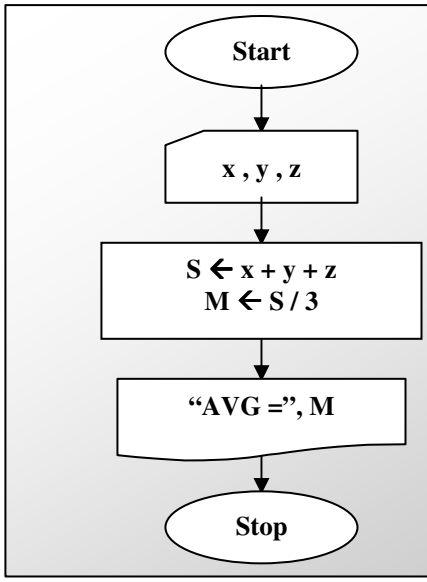
شکل ۲-۴ : جعبهٔ انتساب

رسم اولین کارنما

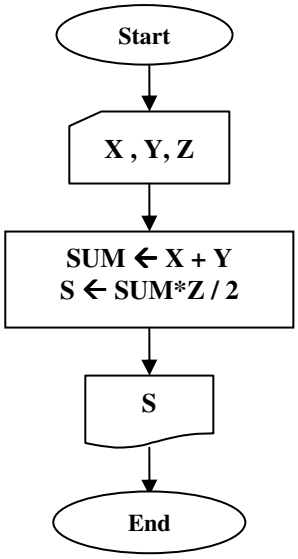
حال وقت آن رسیده است که اولین کارنما را با استفاده از آنچه که تا اینجا فراگرفتیم رسم کنیم.

کار در کلاس ۲-۱ :

کارنمای روبرو چه کاری انجام می دهد؟
تک تک مراحل آن را توضیح دهید.

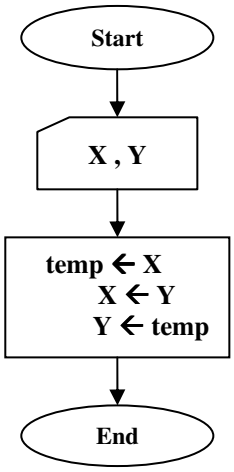


مثال ۲-۱ : کارنمایی که با دریافت سه عدد به‌عنوان قاعدهٔ کوچک، قاعدهٔ بزرگ و ارتفاع یک ذوزنقه مساحت آن را چاپ می‌کند.



شکل ۲- ۵ : کارنمای مثال ۲- ۱

مثال ۲-۲ : کارنمایی که دو مقدار را دریافت کرده و در دو متغیر **X** و **Y** ذخیره می‌کند و سپس محتویات این دو متغیر را به واسطهٔ یک متغیر کمکی به نام **temp** عوض می‌کند.



شکل ۲- ۶ : کارنمای مثال ۲- ۲

آیامی‌توانید محتویات جعبهٔ مستطیلی در کارنمای بالا را چنان تغییردهید که دیگر نیازی به استفاده از متغیر **temp** نباشد؟

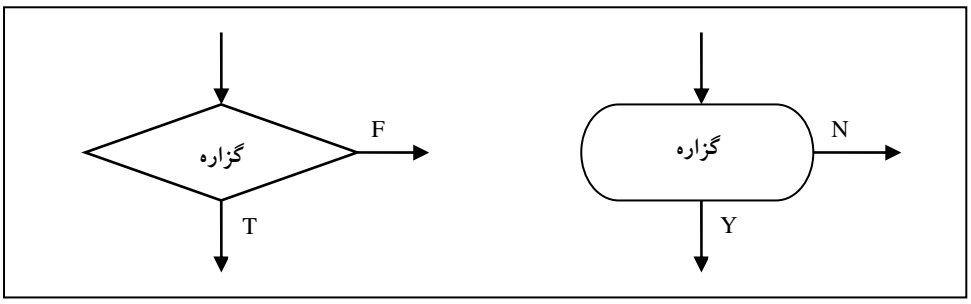
۲-۲ : تصمیم گیری در الگوریتم ها (عملیات مقایسه ای و شرطی)

جعبه های تصمیم

در بسیاری از الگوریتم‌ها پیش می‌آید که بخواهیم روند آن الگوریتم را برحسب شرایط مختلف کنترل کرده و در شرایط گوناگون اَعمال متفاوتی را انجام دهیم. به‌عنوان مثال در الگوریتم تلفن زدن دیدید که، در مرحلهٔ سوم با آوردن یک *جملهٔ شرطی* خواستار آن شدیم، که الگوریتم بر حسب درستی یا نادرستی شرط مذکور به یکی از مراحل ۴ یا ۵ رفته و روند اجرای الگوریتم از یکی از این دو مرحله از سرگرفته شود.

بر این اساس جعبه‌ای با نام *"جعبهٔ‌تصمیم"* در نظر گرفته شده است که در داخل آن یک گزارهٔ ساده یا مرکب قرار می‌گیرد؛ و دو پیکان با بر چسب درست (**T**) و نادرست (**F**) از آن خارج می‌شود. پیشتر در ریاضیات گسسته با گزاره به‌عنوان *" جمله‌ای خبری که دارای ارزشی درست یا نادرست است "* آشنا شده‌اید. بنابراین در خصوص گزاره و ویژگی‌های آن بحثی نخواهیم کرد و به بررسی ویژگی‌های جعبهٔ تصمیم می‌پردازیم. هنگامی که روند اجرای الگوریتم به جعبهٔ تصمیم برسد، با بررسی ارزش گزارهٔ داخل جعبه، براساس درست یا نادرست بودن گزارهٔ مذکور، روند الگوریتم در جهت پیکان **T** [در صورت درست بودن گزاره] و یا در جـهت پیکان **F** [در صورت نادرست بودن گزاره] حرکت می‌کند. گاهی به‌جای دو نماد فوق نمادهای **Y** و **N** به‌کار می‌رود.

در حالت استاندارد برای جعبهٔ تصمیم از لوزی استفاده می‌کنند، اما در این کتاب در بیشتر مثال‌ها از بیضی تخت با طول دلخواه استفاده شده است، تا آزادی عمل بیشتری در توضیح و تفصیل گزارهٔ جعبهٔ تصمیم برای دانشجویان فراهم کرده باشیم، و بدین‌وسیله آنها هر چه لازم می‌دانند گزارهٔ خود را شرح و تفصیل دهند. خصوصاً برای دانشجویانی که در ابتدای راه هستند شرح و تفصیلِ جعبه‌های تصمیم بسیار لازم و ضروری است. اما شما می‌توانید از هر یک از این دو جعبه استفاده کنید چنانکه ما نیز از هر دوی این جعبه‌ها در مثال‌های کتاب بهره گرفته‌ایم. شکل کلی این جعبه‌ها به‌صورت زیر است:



شکل ۲ - ۷ جعبه‌های تصمیم

از آنجا که اصولاً در اکثر الگوریتم‌ها از عبارات رابطه‌ای (شرطی) به‌عنوان گزاره استفاده می‌شود در ادامه قبل از بیان هرگونه مثالی به بررسی انواع عبارات رابطه‌ای می‌پردازیم.

بررسی عبارات رابطه‌ای (شرطی) ساده

منظور از یک عبارت رابطه‌ای (شرطی) ساده هر عبارتی به شکل کلی زیر است:

عبارت حسابی ۲ + یک علامت رابطه ای + عبارت حسابی ۱

و مقصود از علائم رابطه‌ای علامت‌های زیر است که از قبل در ریاضیات با آنها آشنا شده‌اید:

$$= ، \neq ، > ، < ، \leq ، \geq$$

یک عبارت شرطی ساده می‌تواند دارای ارزشی درست یا نادرست باشد. به‌عنوان مثال عبارات شرطی زیر را در نظر بگیرید:

۱. $۳ < ۵$: این عبارت دارای ارزش نادرستی است و اگر در داخل یک جعبهٔ تصمیم قرار بگیرد موجب می‌شود که روند اجرای الگوریتم به سمت پیکان **F** برود.

۲. $۴ \leq ۰$: این گزاره دارای ارزش درستی است و اگر در داخل یک جعبهٔ تصمیم قرار بگیرد موجب می‌شود که روند اجرای الگوریتم به سمت پیکان **T** برود.

۳. $n \neq m$: این گزاره در صورتی که دو متغیر دارای محتویات متفاوتی باشند دارای ارزش درستی خواهد بود و روند اجرای الگوریتم را به سمت پیکان **T** می‌برد. و در صورتی که دارای محتویات مشابهی باشند دارای ارزش نادرستی بوده و روند اجرای الگوریتم را به سمت پیکان **F** هدایت می‌کند.

با توجه به آنکه علائم رابطه‌ای برروی اعداد تعریف شده‌اند، اگر در طرفین علامت رابطه‌ای یک عبارت محاسباتی(جبری) آمده باشد، ابتدا حاصل آن عبارت جبری محاسبه شده و جایگزین آن عبارت می‌گردد. سپس براساس حاصل به‌دست آمده ارزش گزارهٔ شرطی معین می‌شود. به‌عنوان مثال به عبارت زیر توجه کنید:

۴. $B^2 - 4*A*C > 0$: در این عبارت شرطیِ ساده ابتدا حاصل عبارت معروف دلتا که در سمت چپ علامت رابطه ای قرار دارد محاسبه می‌شود و سپس در صورتی که مقدار آن مثبت و بزرگتر از صفر باشد روند اجرای الگوریتم به سمت پیکان **T** و در غیر این صورت به سمت پیکان **F** می‌رود.

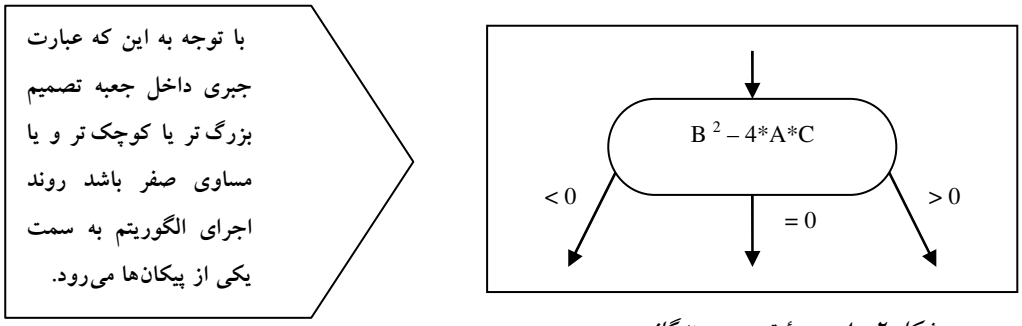
البته بدین دلیل ابتدا باید مقدار عبارت جبری محاسبه شود که در جدول تقدم عملگرها، عملگرهای رابطه‌ای پس از عملگرهای حسابی قرار دارند و بنابراین دیرتر مورد ارزیابی قرار می‌گیرند. در این خصوص در بخش ۲-۱۴ بیشتر صحبت خواهیم کرد.

بنابراین مفهوم الگوریتم شرطی را می‌توان به‌طور خلاصه چنین بیان کرد: هر گاه در الگوریتمی بخواهیم برحسب شرایطی، مجموعه‌ای از دستورات اجرا شوند و در صورت عدم تحقق آن شرایط مجموعهٔ دیگر از دستورات اجرا شوند، باید از جعبه‌های تصمیم استفاده کنیم. همچنین شکل کلی زیر را می‌توان برای آن عنوان نمود:

اگر (شرط درست است) آنگاه (مجموعه دستورات۱) وگرنه (مجموعه دستورات ۲)

تذکر: به کاربرد کلمهٔ " اگر" در عبارت فوق خوب توجه کنید. رابطه‌ای مستقیم بین این کلمه و جملهٔ شرط و به‌کارگیری جعبهٔ تصمیم وجود دارد.

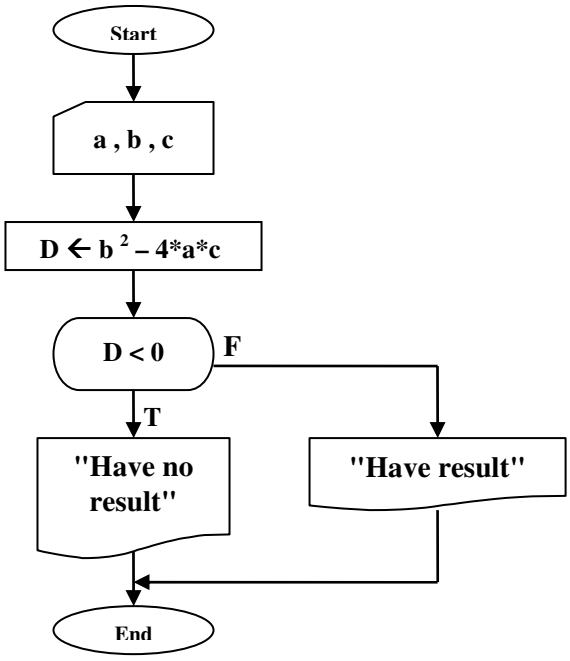
اما گاهی هم پیش می‌آید که با جعبه‌های تصمیم چندگانه (چند راهه) سروکار داشته باشیم. به‌عنوان مثال :



شکل ۲- ۸ : جعبهٔ تصمیم چندگانه

در خصوص ویژگی‌های جعبه‌های تصمیم چندگانه (چندراهه) نیز در قسمت ۲-۱۴ بیشتر صحبت خواهیم کرد.

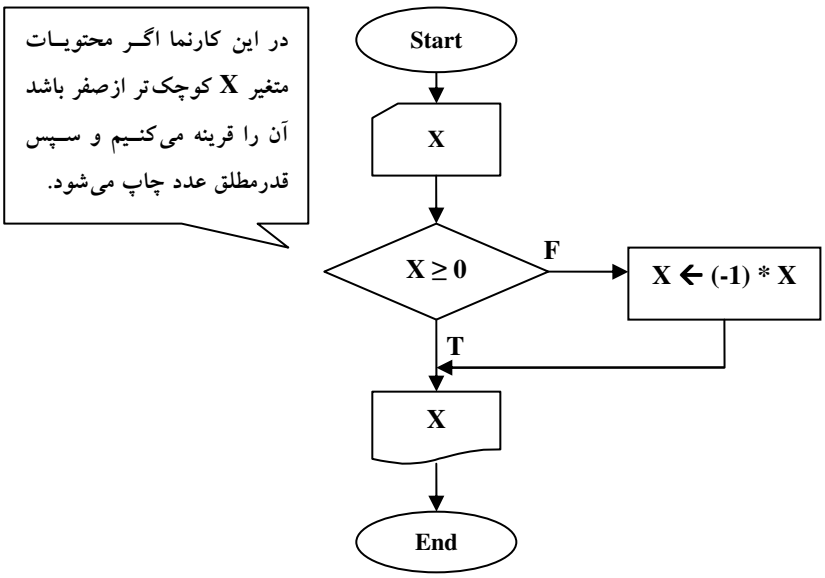
مثال ۲-۳ : کارنمایی که با دریافت سه عدد به‌عنوان ضرایب یک معادلهٔ درجهٔ دوم اعلام می‌کند که آیا این معادلهٔ درجه دوم ریشهٔ حقیقی دارد یا نه.



شکل ۲- ۹ : کارنمای مثال ۲-۳

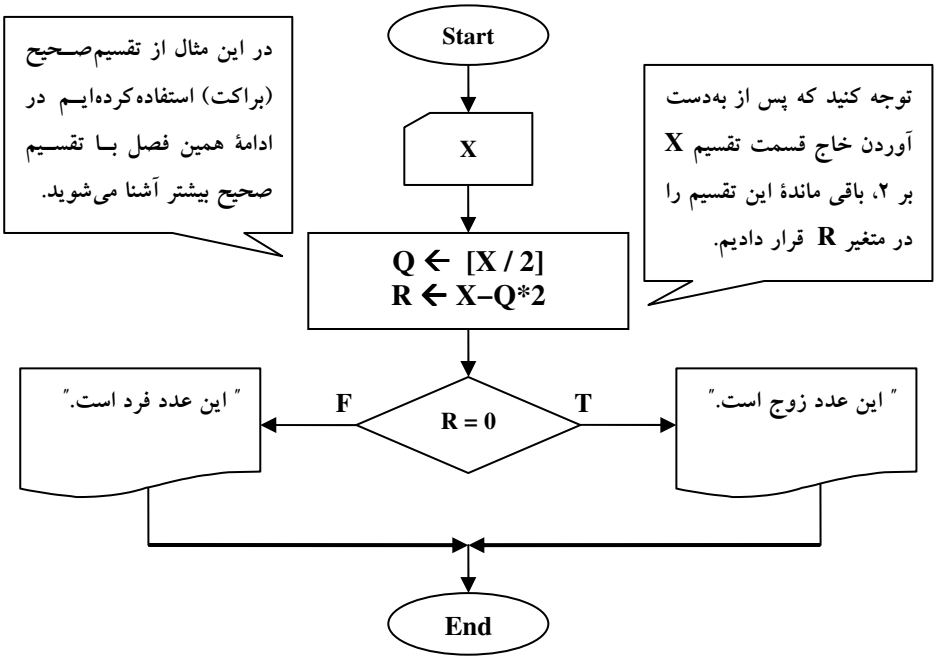
توضیح کارنما : در این کارنما پس از خواندن مقادیر سه متغیر **a** و **b** و **c** حاصل عبارت Δ محاسبه می‌شود و در صورتی که مقدار عبارت دلتا کوچکتر از صفر باشد، روند اجرای الگوریتم به سمت پیکان **T** می‌رود و پیغام عدم وجود جواب حقیقی را چاپ می‌کند و در صورتی که بزرگ‌تر یا مساوی صفر باشد، روند اجرای الگوریتم به سمت پیکان **F** خواهد رفت و پیغام وجود داشتن جواب چاپ می‌شود و در هر دو صورت در نهایت به حالت توقف می‌رود.

مثال ۲-۴ : کارنمایی که با دریافت یک عدد، قدرمطلق آن را چاپ می‌کند.



شکل ۲- ۱۰ : کارنمای مثال ۲-۴

مثال ۲-۵ : کارنمایی که با دریافت یک عدد مشخص می‌کند آن عدد زوج است یا فرد.

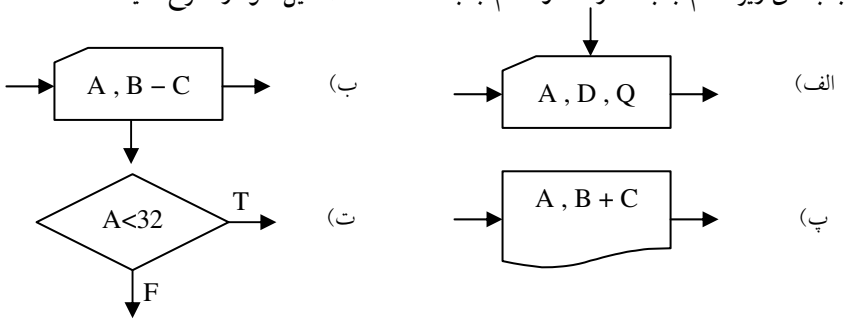


شکل ۲- ۱۱ : کارنمای مثال ۲- ۵

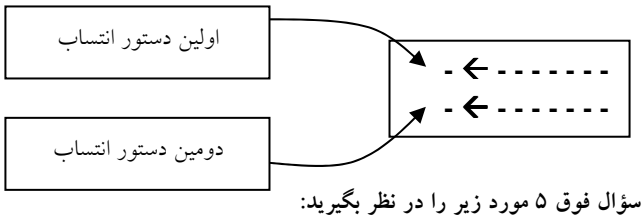
۲-۳ : تمرین

۱. مزایای استفاده از کارنما به جای زبان‌های طبیعی چیست؟

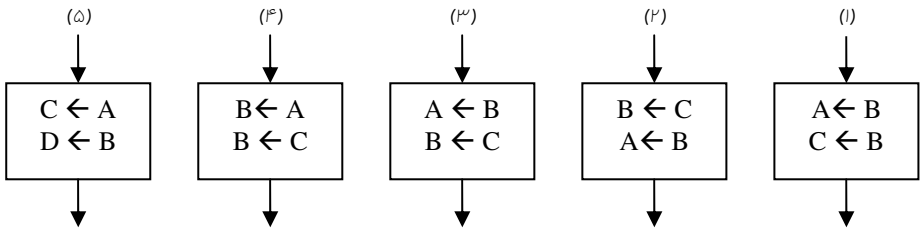
۲. از جعبه‌های زیر کدام جعبه‌ها درست و کدام جعبه‌ها غلط است؟ دلیل خود را شرح دهید.



۳. در شکل زیر، دو دستور انتساب در یک جعبهٔ انتساب قرار گرفته‌اند. یک ضابطهٔ کلی بیان کنید که تحت آن بتوان جای این دو دستورالعمل را، بدون اینکه تغییری در حاصل عملکرد جعبه ایجاد شود، عوض کرد.



راهنمایی: قبل از جواب دادن به سؤال فوق ۵ مورد زیر را در نظر بگیرید:



۴. درستی یا نادرستی هر یک از گزاره‌های موجود را با توجه به کارنمای روبرو تعیین کنید. (برای ۳ گزارهٔ نخست فرض کنید مجموعه داده‌هایی که در نتیجهٔ اجرای جعبهٔ ۱ خوانده می‌شود، به ترتیب عبارت‌اند از ("F", "L", "O", "W", "1", "0") و برای ۲ گزارهٔ آخر، فرض کنید در جعبهٔ ۱ هر مجموعهٔ دلخواهی از داده‌ها امکان دارد وارد شود.)

الف- مجموعه داده‌های فوق، اجرا فقط یکبار به جعبهٔ ۵ می‌رسد.

ب- برای مجموعه داده‌های ارائه شده، اجرا به جعبهٔ ۶ خواهد رسید.

ج- اگر در داده‌های بالا به جای صفر عدد ۶ به کامپیوتر داده شود، جعبهٔ ۲ شش مرتبه اجرا خواهد شد.

د- به ازای هر مجموعه داده‌ای، که در جعبهٔ ۱ خوانده شود، جعبهٔ ۲ همیشه دو بار اجرا می‌شود.

هـ- فقط صفر و یک به‌عنوان داده برای متغیر SWITCH قابل قبول است.

۵. کدام یک از پنج جعبهٔ خروجی که در زیر آمده، خروجی الگوریتم تمرین قبلی با همان ورودی‌ها است ؟

۶. آیا می‌توان کارنمای مثال ۲-۲ را به‌گونه‌ای تغییر داد که بدون استفاده از متغیر کمکی محتویات دو متغیر X و Y را تعویض کند؟

(1)

F L O W 1

(2)

F L O W 1 1
W O L F

(3)

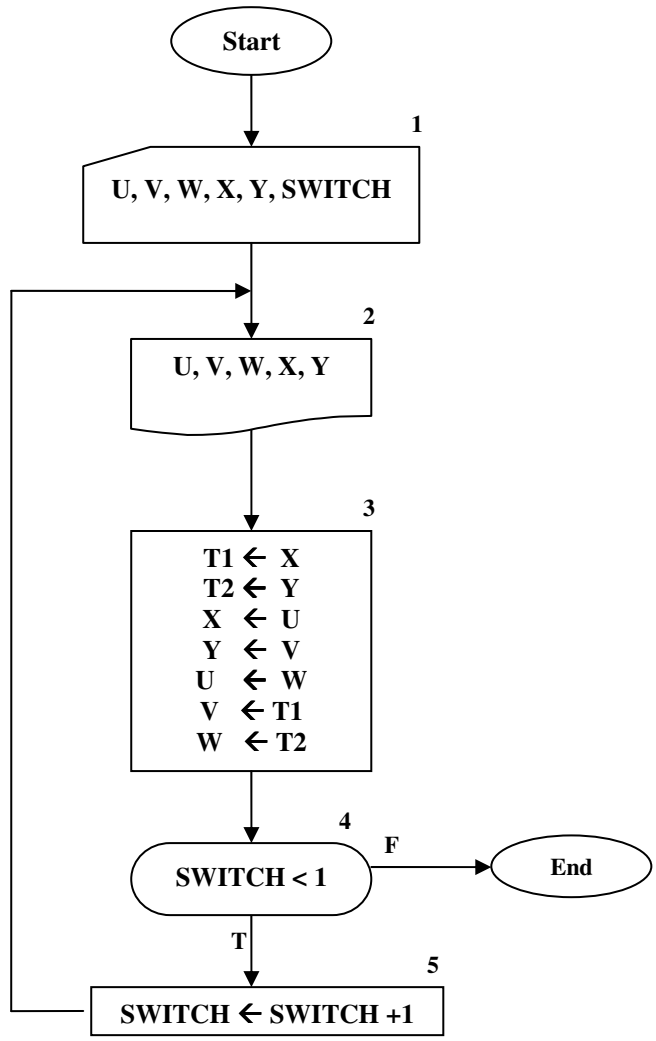
F L O W 1
O W 1 F L

(4)

W L O F 1
O W 1 F L

(5)

O W 1 F L
1 F L O W



شکل ۲- ۱۲ : کارنمای تمرین‌های ۴ و ۵

۲-۴ : تکرار در الگوریتم ها

شمارنده ها و نگهبان ها

مهمترین نقشی که ماشین‌ها از جمله کامپیوتر در زندگی روزمرهٔ انسان‌ها دارند، چنین است که انسان را از انجام امور تکراری و کسل‌کننده معاف می‌کنند. در این بین با بسیاری از مسائل برخورد می‌کنید که برای حل آنها نیازمند تکرار یک الگوی خاص هستید. برای حل اینگونه مسائل تنها کافی است مسئله را برای یک نمونه حل کنید و سپس با به‌کارگیری مکانیسم تکرار، الگوریتم را برای تعدادِ بیشتری داده گسترش دهید و بدین‌طریق قادر به حل کل مسئله خواهید بود. به‌عنوان مثال فرض کنید می‌خواهید معدل یک کلاس ۱۰۰ نفری که هر کدام از دانشجویان کلاس دارای ۵ نمره هستند را محاسبه کنید.!؟ اگر بخواهید چنین عمل خسته‌کننده‌ای را با دست انجام دهید، قطعاً در میانهٔ راه باز خواهید ماند، و مسلماً با افزایش تعداد اعداد و ارقام عرصه سخت‌تر خواهد شد. پس بهتر است این قبیل کارها را به وسیلهٔ کامپیوتر انجام دهید. اما شما چه ایده‌ای برای حل اینگونه مسائل دارید؟ یا به زبان ساده‌تر شما چه الگوریتمی را برای این مسئله می‌توانید ارائه دهید؟

کار در کلاس ۲-۲ :

به ابتکار خود کارنمایی برای حل مسئلهٔ فوق ارائه دهید؟

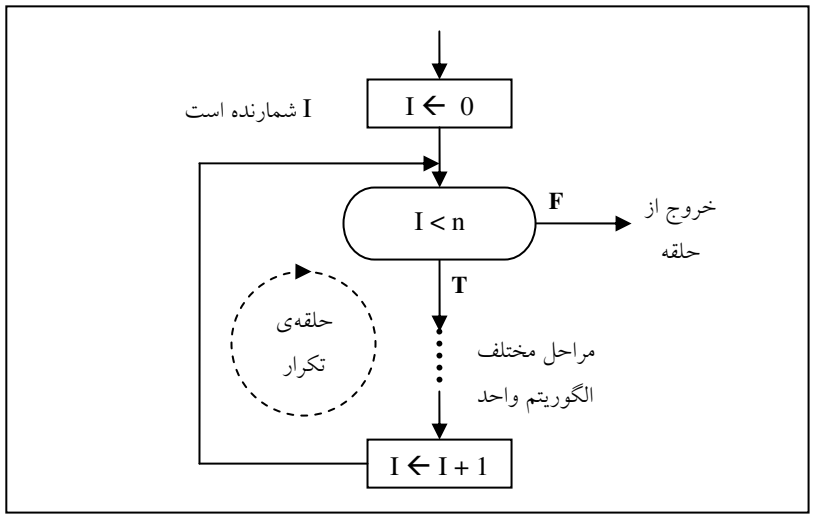
بحث خود را با این سؤال ادامه می‌دهیم که؛ برای حل این مسئله نیازمند انجام چه کارهایی هستیم.!؟

۱. ابتدا باید الگوریتمی بنویسیم که معدل یک دانشجو را بتواند حساب کند. به عبارت دیگر، باید مسئله را برای حالتی که تعداد موارد برابر یک است حل کنیم. اسم این الگوریتم را الگوریتم واحد می‌گذاریم.

۲. سپس باید با یک مکانیسم تکرار این عمل را آنقدر تکرارکنیم تا معدل همهٔ دانشجویان حساب شود، و در نهایت از مجموعهٔ معدل‌های به‌دست آمده معدل می‌گیریم تا معدل کل کلاس حساب شود.

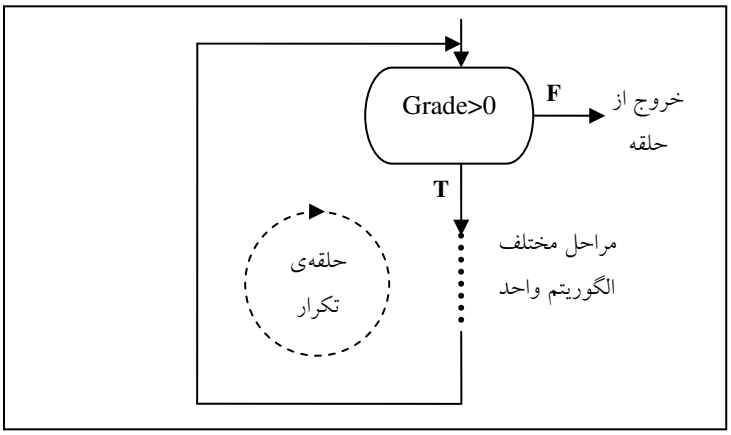
آنچه که تا اینجا مطرح شد استراتژی حل مسئله نامیده می‌شود. حال باید به دنبال روشی برای پیاده‌سازی این استراتژی باشیم. مهمترین مطلبی که می‌توانیم برای حل این مشکل از آن کمک بگیریم تعداد دانشجویان است. وقتی که ما تعداد دانشجویان را بدانیم و مسئله را برای یک دانشجو حل کرده باشیم، می‌توانیم الگوریتم را به‌گونه‌ای تغییر دهیم که الگوریتم واحد را برای n دانشجو تکرار کند؛ و در نتیجه معدل این تعداد دانشجو حساب می‌شود.

به‌این روش یعنی قراردادن یک متغیر برای محاسبهٔ تعداد دفعات گذر از حلقهٔ تکرار *"روش شمارنده"* گفته می‌شود. برای به‌کارگیری این روش ابتدا یک متغیر را به‌عنوان شمارندهٔ تعداد دفعات گذر از حلقه، به صفر مقدار اولیه می‌دهیم، سپس به ازای هر بار گذر از حلقهٔ تکرار یک واحد به آن متغیر می‌افزاییم، در نتیجه قبل از هر بار انجام دوبارهٔ الگوریتم واحد، چک می‌کنیم که آیا الگوریتم ما به تعداد دفعات موردنظر انجام شده است یا نه.؟ اگر حلقهٔ تکرار که دربردارندهٔ الگوریتم واحد است به تعداد دفعات موردنظر انجام شده بود باید از انجام دوبارهٔ آن جلوگیری کنیم وگرنه باید یکبار دیگر مراحل بالا انجام شود. بنابراین باید شرط تکرار حلقه را در یک جعبهٔ تصمیم قرار دهیم. بدین ترتیب اجرای مکرر الگوریتم واحد ادامه پیدا می‌کند تا در نهایت الگوریتم به حالتی برسد که از حلقه خارج شود. صورت کلی این روش در شکل ۲- ۱۳ نشان داده شده است:



شکل ۲- ۱۳ : نمای کلی از حلقهٔ تکرار با شمارنده

اما به روش دیگری نیز می‌توان تعداد دفعات تکرار حلقه‌های تکرار را کنترل کرد، که به *"روش نگهبان"* موسوم است. در این روش شرط حلقه را به‌گونه‌ای تنظیم می‌کنیم که به واسطهٔ دستورات داخل خود حلقه پس از چندین بار تکرار حلقه، آن شرط نقض گردد و در نتیجه حلقه متوقف شود. به‌عنوان مثال می‌توان یک ورودی نامعتبر را به‌عنوان خاتمه دهندهٔ حلقه در نظر گرفت. مثلاً از آنجایی که نمرهٔ زیرِ صفر، یک ورودی نامعتبر است، می‌تواند به‌عنوان شرط خاتمه‌دهندهٔ حلقهٔ تکرار مثال قبل به‌کار رود، بدین‌صورت که هر گاه کاربر یک نمرهٔ منفی را به‌عنوان ورودی وارد کرد، شرط حلقه دارای ارزش نادرست گردد و از حلقه خارج شود.



شکل ۲- ۱۴ : نمای کلی از حلقهٔ تکرار با نگهبان

در شکل ۲- ۱۵ کارنمای الگوریتم‌هایی را که برای مسئلهٔ محاسبهٔ معدل بیان شد، ملاحظه می‌کنید.

کار در کلاس ۲-۳ :

به سؤالات زیر در خصوص کارنماهای شکل ۲-۱۵ پاسخ دهید:

۱. *هر یک از کارنماها را شرح دهید و نام روش هر کدام را بنویسید؟*

۲. *چرا در یک کارنما متغیر* **N** *را به صفر و در دیگری به یک مقدار اولیه داده ایم؟*

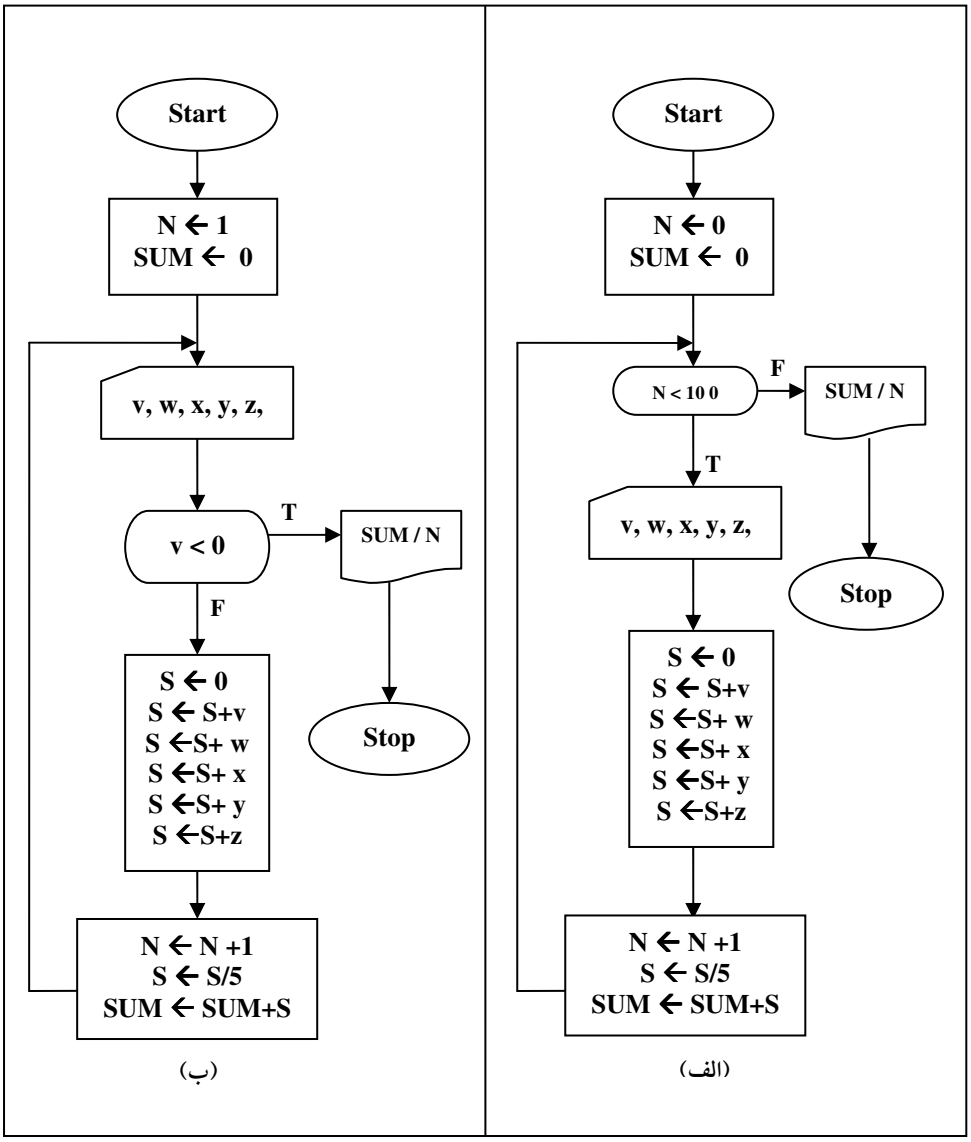
۳. *اگر در هر دو الگوریتم به‌طور مشابه با متغیر* **N** *رفتار می کردیم چه رخ می داد؟*

۴. *در مورد تعداد دفعات اجرای حلقه در هر یک از کارنماها بحث کنید؟*

۵. *اگر در کارنمای (ب) به جای شرط* **v <0** *از شروط دیگری استفاده می‌کردیم چه می شد؟ اصلاً چنین امری ممکن هست یا خیر؟*

۶. *نگهبان حلقه ی تکرار چه ویژگی‌هایی باید داشته باشد؟*

راهنمایی : فرض کنید دامنهٔ نمرات بین ۰ تا ۲۰ می باشد.



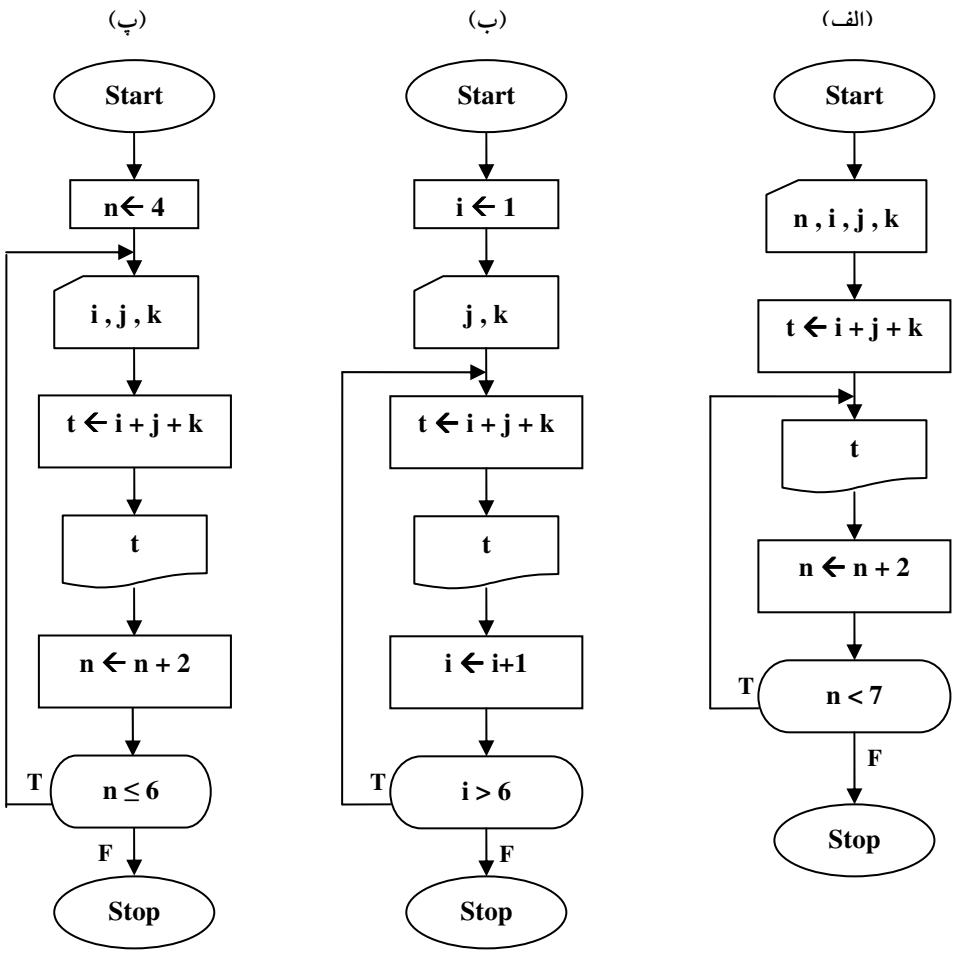
شکل ۲-۱۵ دو نوع کنترل حلقهٔ تکرار

تنها نکته‌ای که از بحث الگوریتم های دارای نگهبان ناگفته ماند این است که، اگر نگهبان حلقه به واسطهٔ دادهٔ ورودی نقض می‌شود، این ورودی نباید در دامنهٔ داده‌های قابل قبول الگوریتم قرار داشته باشد. مثلاً در کارنمای (ب) در شکل۲-۱۵ شرط نمرهٔ منفی را در نظر گرفتیم، چرا که نمرهٔ زیر صفر، معنی ندارد و کاربر با وارد کردن عددی منفی موجب ایستادن حلقهٔ تکرار می‌شود.

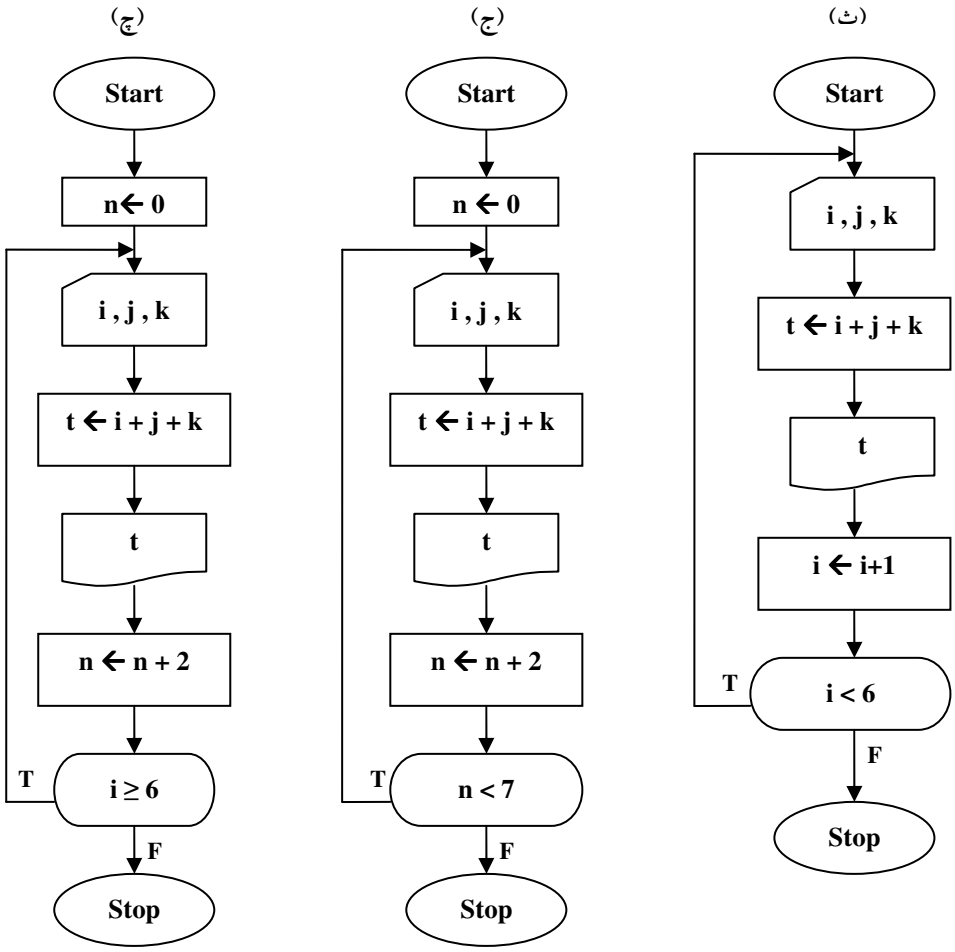
حال اگر شرط این حلقه (**v > 20**) بود چه تغییراتی باید در الگوریتم و یا نحوهٔ استفاده از آن می‌دادیم؟

۲-۵: تمرین

۱. در مورد حلقه هایی که با شمارنده کنترل می‌شدند، به‌خاطر دارید که تکرار محاسبات تعیین شده وقتی متوقف می‌شود که به تعداد دفعات موردنظر از حلقه عبور شده باشد. در هر کدام از کارنماهای زیر تعداد دفعات اجرای حلقه را تعیین کنید؟



شکل ۲- ۱۶ (الف)- کارنماهای تمرین ۱



شکل ۲- ۱۶ (ب)- کارنماهای تمرین ۱

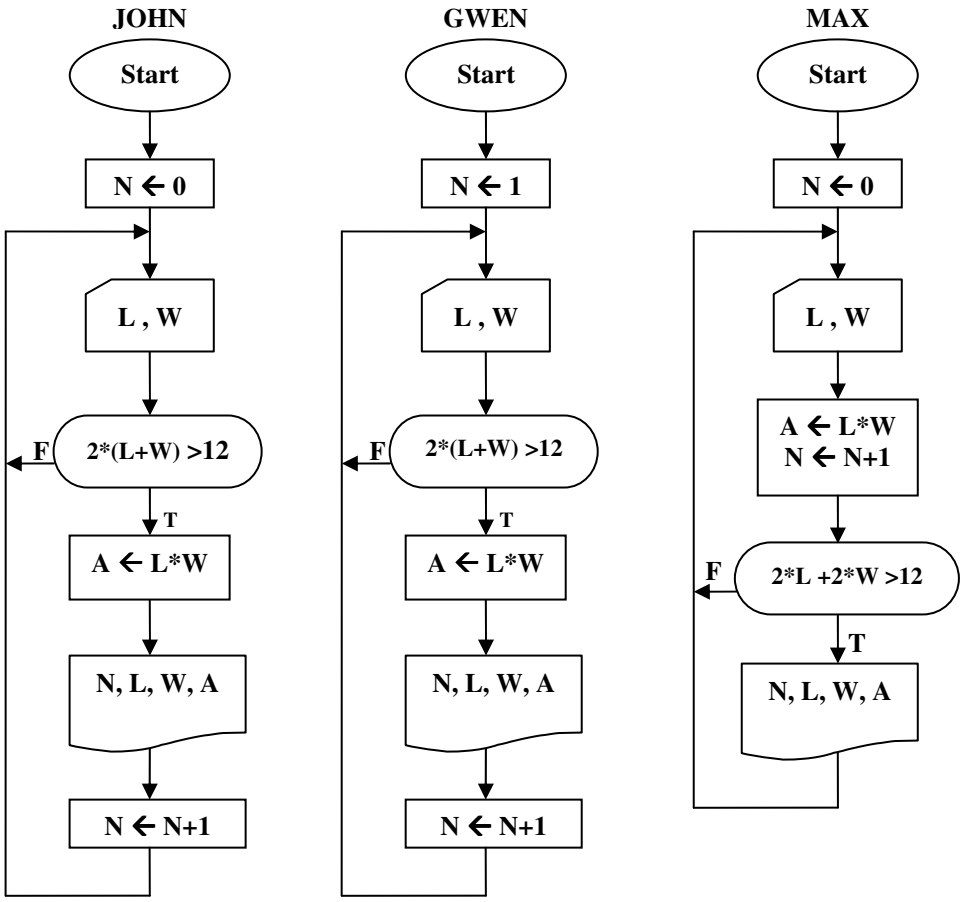
آیا کارنمایی وجود داشت که نتوانید تعداد تکرار حلقه آن را محاسبه کنید؟ در صورت مثبت بودن پاسخ چه نتیجه‌ای گرفته‌اید؟ به عبارت دیگر از این پس به چه نکته‌ای در ترسیم کارنماهای خود باید توجه داشته باشید؟

۲. استادی مسئلهٔ ساختن کارنمایی به شرح زیر را به دانشجویان کلاسی محول کرد. ورودی الگوریتم شامل طول و عرض چند مستطیل است. هدف، تهیهٔ فهرستی است از خطوطی که دارای شمارهٔ ردیف بوده و طول (L)، عرض (W)، و مساحت (A)، آن دسته از مستطیل‌هایی را که محیط آنها از ۱۲ بیشتر است به‌دست دهد. کارنماهای شکل ۲-۱۷ به‌عنوان حل این مسئله به معلم تحویل داده شد. وظیفهٔ شما به شرح زیر است:

الف- تعیین کنید کدام راه‌حل‌ها صحیح و کدام‌ها غلط هستند. (فعلاً به راه‌حلی، راه‌حل صحیح می‌گوییم که خروجی مورد نظر را تولید کند. هرچند که ممکن است کاراترین راه‌حل نباشد.)

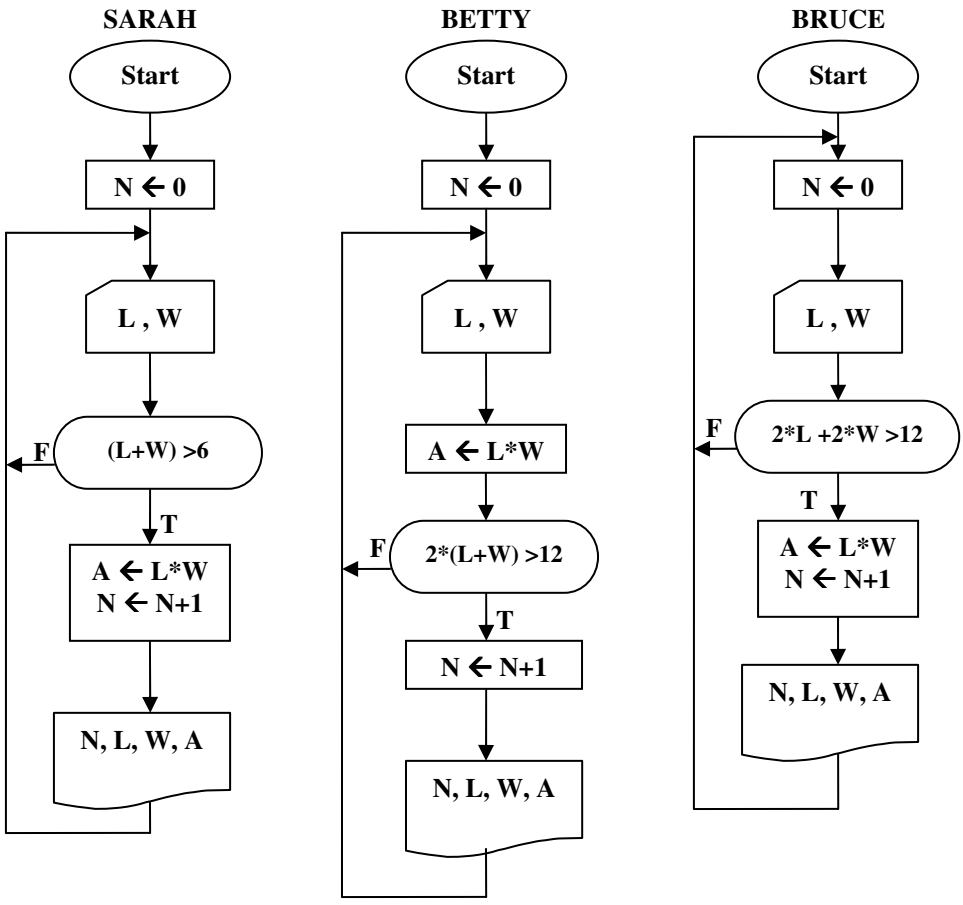
ب- در مورد راه‌حل‌های غلط، اشتباهات جواب‌هایی که به‌دست می‌آیند در چیست؟

ج- راه‌حل‌های صحیح را از نظر کارایی بررسی کنید و با استفاده از بهترین خصوصیات هر کدام، یک کارنما بسازید. (از دو برنامه، آنکه به تعداد کمتری از محاسبات نیاز دارد کاراتر است).



شکل ۲- ۱۷ (الف)- کارنماهای تمرین ۲

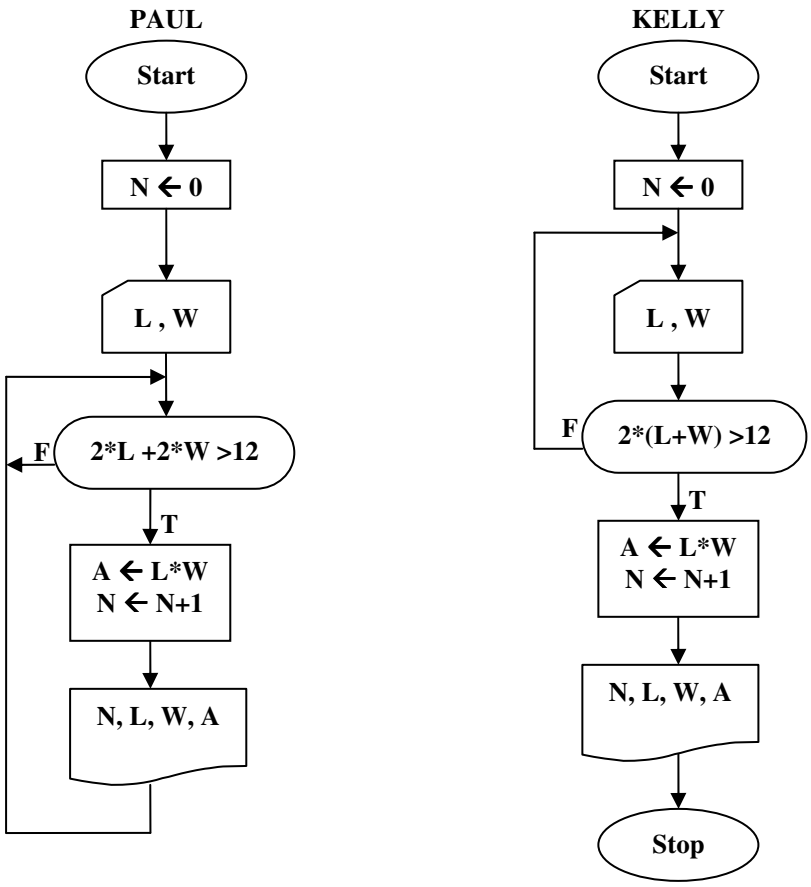
۳. مطابق مقررات پستی اگر طول جعبه‌ای به علاوهٔ اندازهٔ دور آن بیش از ۱۸۰ سانتیمتر باشد، آن را نمی‌توان به‌وسیلهٔ پست ارسال کرد.(اندازهٔ دور جعبه عبارت است از طول کوتاه‌ترین نخی که بتوان به دور جعبه بست.) سازنده‌ای تعداد زیادی جعبه دارد که می‌خواهد توسط پست ارسال کند. برای هر جعبه ابعاد سه گانهٔ آن (به ترتیب از بزرگترین به کوچکترین) و یک شمارهٔ شناسایی ذخیره شده است. کارنمایی بنویسید که با خواندن این اطلاعات، برای هر جعبه که با مقررات فوق مغایرت دارد شمارهٔ مشخصهٔ جعبه و همچنین مجموع طول و اندازهٔ دور جعبه را چاپ کند. (تعداد جعبه‌ها را نیز از کاربر دریافت کنید).



شکل ۲-۱۷ (ب)- کارنماهای تمرین ۲

۴. کارنمایی رسم کنید که تمام جملات کوچکتر از ۱۰ میلیونِ دنبالهٔ فیبوناچی را چاپ کند. همچنین ترتیبی بدهید که سطرهای خروجی، مانند شکل زیر به ترتیب شماره‌گذاری و چاپ شوند.

1	0
2	1
3	1
4	2
5	3
6	5
7	8



شکل ۲-۱۷ (پ)- کارنماهای تمرین ۲

۵. یک کارنما برای الگوریتمی بسازید که یک مجموعهٔ هزارتایی از مقادیر را یکی یکی قبول و مجموع مقادیر مثبت را حساب کند، صفرها را نادیده بگیرد و تعداد مقادیر منفی را بشمارد. وقتی محاسبات در مورد هر صد عدد انجام گرفت، مجموع مقادیر مثبت و تعداد مقادیر منفی باید چاپ شوند.

۶. کارنمایی رسم کنید که **n** عدد را به‌عنوان ورودی دریافت کند، سپس بزرگترین و کوچکترین عدد موجود در بین این اعداد را چاپ کند؟

۷. سه عدد **a** و **b** و **c** مفروض هستند. الگوریتمی بنویسید که معین کند آیا این سه عدد می‌توانند طول اضلاع یک مثلث باشند یا نه؟

۸. سه عدد **a** و **b** و **c** مفروض هستند. الگوریتمی بنویسید که معین کند آیا این سه عدد می‌توانند طول اضلاع یک مثلث قائم الزاویه باشند یا نه؟

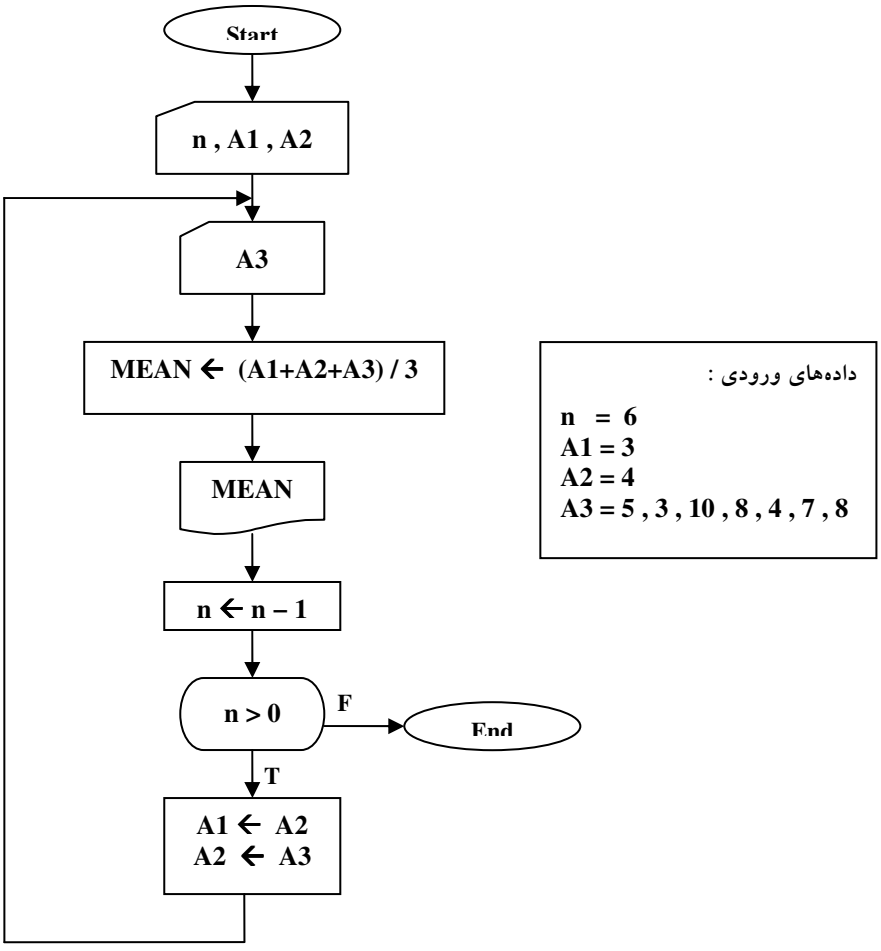
۹. کارنمای شکل ۲-۱۸ را با استفاده از مقادیر داده شده پیگیری کنید، سپس درستی یا نادرستی هر یک از گزاره‌های (الف) تا (ت) را تعیین کنید.

الف- دومین مقدار چاپ شده (در جعبهٔ ۶) هفت است.

ب- جعبهٔ ۹ چهار بار اجرا می‌شود.

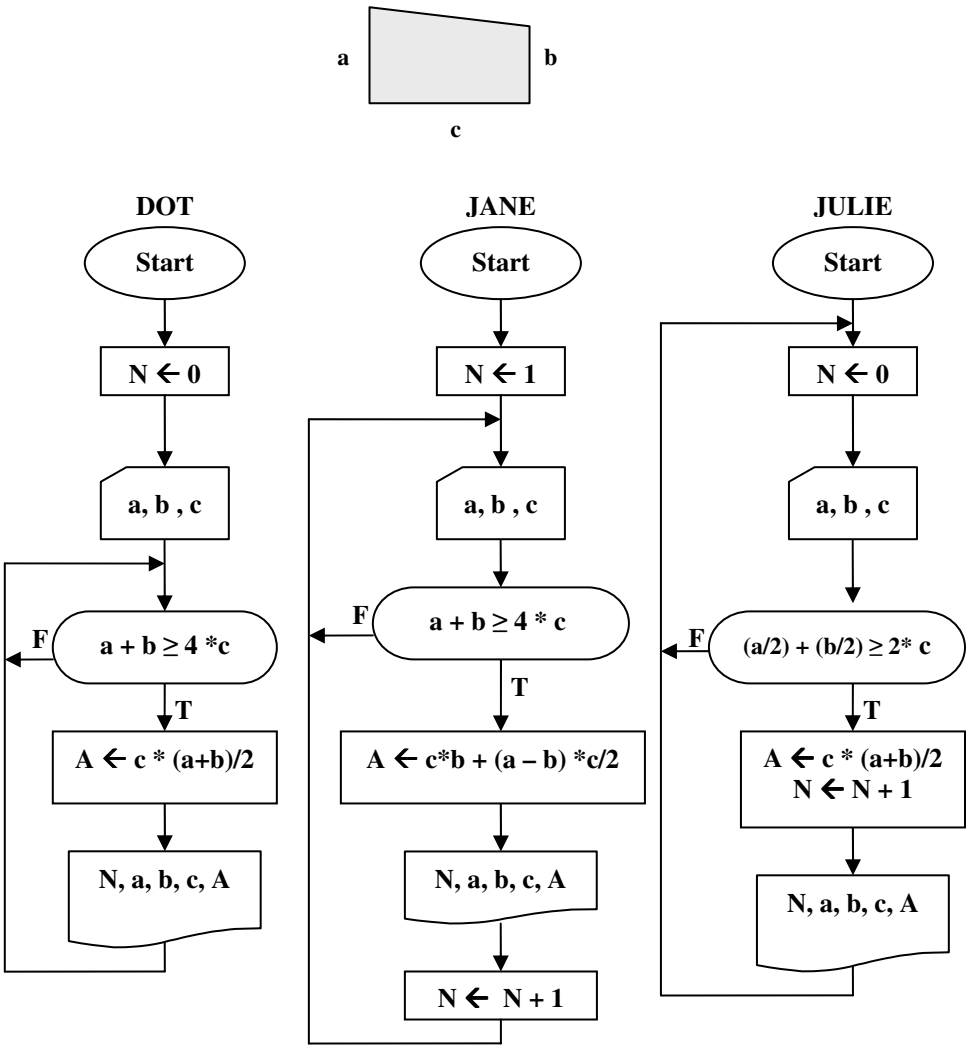
پ- آخرین مقدار که برای A3 داده شده است، به داخل آورده نمی‌شود.

ت - در این الگوریتم متغیر n به‌عنوان شمارنده‌ای برای کنترل حلقه مورد استفاده قرار می‌گیرد.



شکل ۲-۱۸ : کارنماهای تمرین ۹

۱۰.در شکل ۲-۱۹ پنج کارنما نشان داده شده که به‌عنوان حل مسئلهٔ زیر توسط دانشجویان ساخته شده‌اند. به‌عنوان ورودی، مقادیر قاعدهٔ c و اضلاع a و b از چند ذوزنقه داده شده است. (فرض کنید a > b) فهرستی تهیه کنید از سطرهایی که دارای شمارهٔ ردیف بوده و هر سطر متشکل از a و b و c و مساحت A ی آن دسته از ذوزنقه‌هایی باشد که ارتفاع متوسط آنها یعنی (a + b) / 2 حداقل دو برابر قاعدهٔ c است.

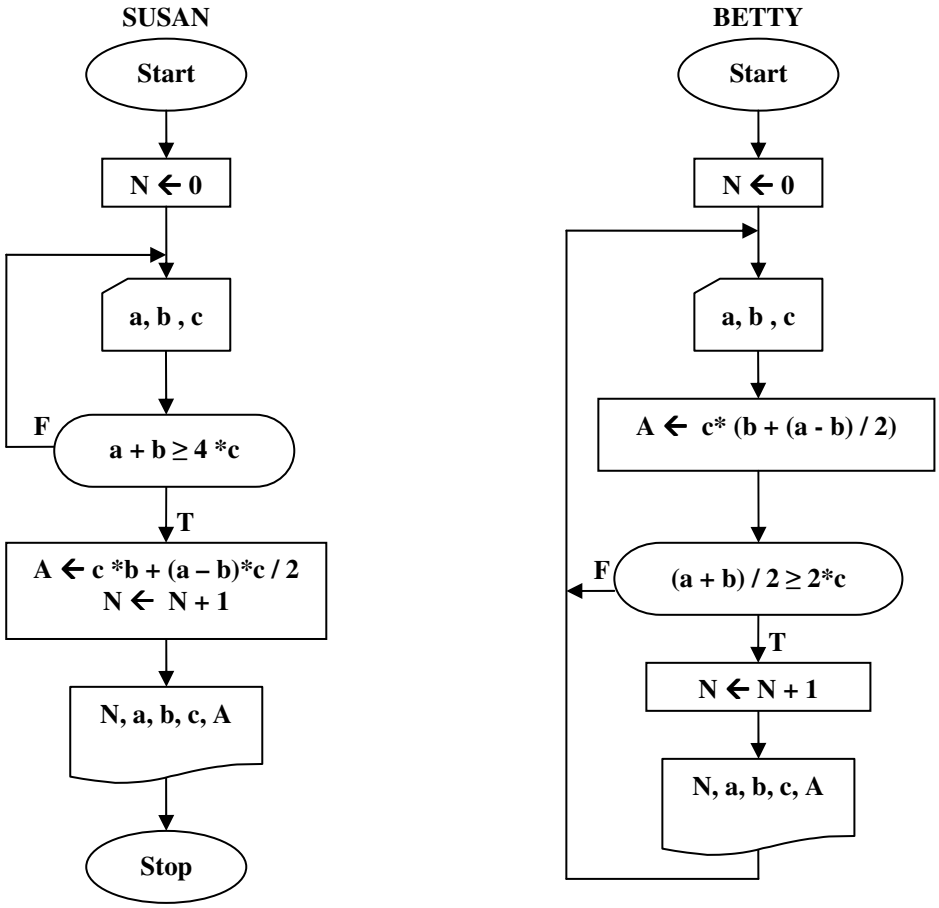


شکل ۲-۱۹ (الف)- کارنماهای تمرین ۱۰

وظیفهٔ شما به شرح زیر است:

الف- تعیین کنید راه حل کدام دانشجو صحیح و کدام غلط است.

ب- با استفاده از کاراترین خصوصیات هر یک از این ۵ راه‌حل یک کارنمای صحیحِ کارآمد بسازید.



شکل ۲-۱۹ (ب)- کارنماهای تمرین ۱۰

۱۱.کارنمایی رسم کنید که مجموع **n** عدد صحیح را محاسبه کند. برروی روش‌های ممکن برای کنترل تعداد دفعات تکرار حلقه در این مسئله بحث کنید.

۱۲.الگوریتمی بنویسید که مقسوم‌علیه‌های عدد مفروض **N** را چاپ کند.

راهنمایی: در حل این تمرین از تقسیم صحیح باید استفاده کنید. یک عدد بر دیگری بخش‌پذیر است، اگر باقی ماندهٔ تقسیم عدد بزرگتر بر عدد کوچکتر صفر باشد. به مثال ۲-۵ توجه کنید.

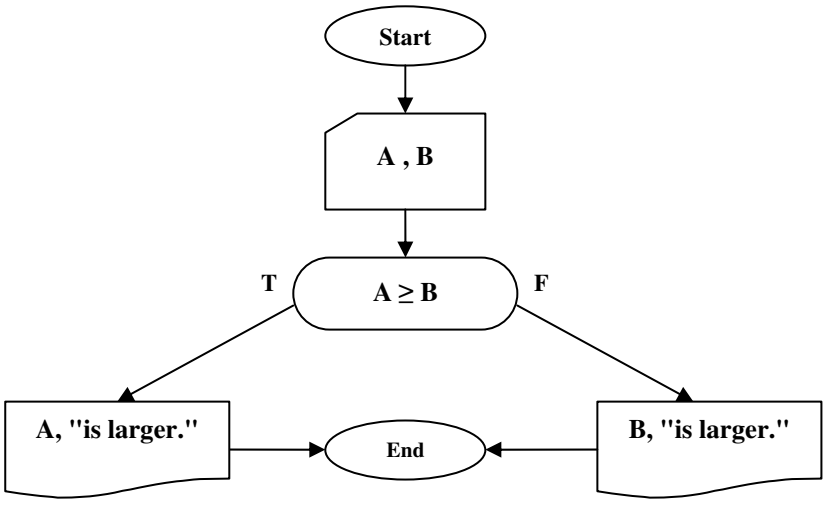
۲-۶: مقدمه‌ای بر الگوریتم‌های جستجو

ساختن یک الگوریتم و کارنمای مربوط به آن، اساساً از مراحل حل مسئله است. برای آنکه شما را در ساخت الگوریتم‌های کارآمد، کارآزموده‌تر سازیم باید با روش حل مسائل بهتر آشنا شوید. شما باید ببینید که برای حل یک مسئله چگونه یک نقطه شروع انتخاب می‌کنیم. شما باید برخی شروع‌های نادرست و غفلت‌ها و برخی الگوریتم‌های ناخوشایند را که در اولین کوشش‌هایمان به‌دست می‌آوریم را ببینید، تا بتوانید از این تجارب در حل مسائل پیچیده‌تر و مهمتر بهره بگیرید و آنها را به‌کار ببندید. از همه مهمتر باید بیاموزید که در ساختن الگوریتم، ابتدا می‌کوشیم تا یک نوع راه‌حل را برای مسئله به‌دست آوریم. سپس راه‌حل خود را با دیدی انتقادی مورد بررسی قرار می‌دهیم و سعی می‌کنیم آن را بهتر سازیم و در صورت لزوم به موارد کلی‌تر تعمیم دهیم. لذا در اکثر فصول کتاب سعی شده تا با مطرح ساختن برخی مسائل روشنگر شما را با شیوه‌های حل مسئله آشناتر سازیم. آشنایی با الگوریتم‌های جستجو اولین قدم در این راستا است.

الگوریتم پیدا کردن بزرگترین عدد از میان چند عدد

بیایید مسئلهٔ پیدا کردن بزرگترین عدد در میان چند عدد را مدنظر قرار دهیم. این مسئله گرچه به خودی خود ساده است، لیکن بخشی از تعداد زیادی الگوریتم‌های دیگر را تشکیل می‌دهد. در واقع، ساده‌ترین الگوریتم از یک نوع کامل الگوریتم‌ها است که به نام *" الگوریتم‌های جستجو"* معروف هستند.

برای حل این مسئله ساده‌ترین حالت را، که پیدا کردن بزرگترین عدد بین دو عدد است، در نظر می‌گیریم و سپس الگوریتم خود را برای تعداد N عدد تعمیم می‌دهیم. برای این منظور در شکل ۲-۲۰ کارنمایی ترسیم شده است که ساده‌ترین حالت را نشان می‌دهد.



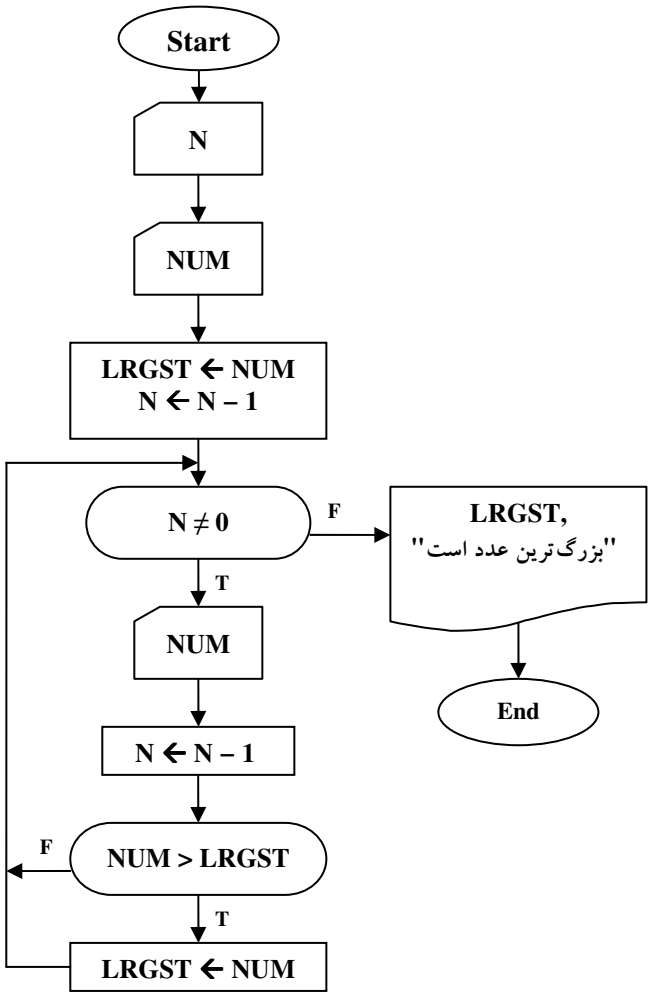
شکل ۲- ۲۰ : کارنمای یافتن بزرگترین عدد از بین دو عدد

حال بیایید به حالتی بپردازیم که در آن می‌خواهیم بزرگترین عدد را در میان N عدد پیدا کنیم. این مسئله یکی از مسائلی است که اگر بخواهیم با توجه به کارنمای ساده‌ترین حالت آن را حل کنیم دچار گمراهی خواهیم شد. فرض کنید بخواهید هر متغیری را که می‌خوانید به یک متعییر مانند A ، B ، C ، D ، E و ... نسبت دهید و سپس با توجه به روش ارائه شده در ساده‌ترین حالت کارنمایی بسازید که بزرگ‌ترین عدد را پیدا کند مسلماً به این نتیجه خواهید رسید که تعداد جعبه‌های موردنیاز کارنمای آن، بیشتر از تعداد مقادیر داده‌ها خواهد بود، و احتمالاً نتیجه خواهید گرفت که انجام این عملیات با دست راحتتر است. این مثال موردی است از اینکه در برخی مسائل عقل سلیم راهنمای بهتری برای حل مسئله است تا بررسی کارنمای ساده‌ترین حالت.

بیایید مسئله را با یک مسئلهٔ مشابه که در زندگی روزمره برای انسان پیش می‌آید شبیه سازی کنیم، تا شاید با این کار به روش مناسبی برای حل مسئله دست یابیم. در حقیقت داریم به عقل و تجربهٔ خود رجوع می‌کنیم. فرض کنید دسته‌ای کارت که برروی هر کدام عددی یادداشت شده است و تعداد آنها بیشتر از دو عدد است، را به شما بدهند و بخواهند بزرگترین کارت را از میان این کارت‌ها پیدا کنید چگونه این عمل را انجام خواهید داد؟ احتمالاً روش زیر را برای حل این مسئله به کار خواهید برد:

۱. ابتدا دو کارت برخواهید داشت، کارت بزرگتر را انتخاب می‌کنید و کارت کوچکتر را کنار خواهید گذاشت.
۲. سپس کارت دیگری برمی‌دارید و آن را با کارتی که از مرحلهٔ قبلی به‌دست آمده مقایسه می‌کنید، دوباره با انتخاب کارت بزرگتر، کارت کوچکتر را کنار خواهید گذاشت.
۳. مرحلهٔ دوم را آنقدر تکرار خواهید کرد تا دیگر هیچ کارتی باقی نماند.
۴. هر کارتی که در نهایت باقی‌مانده باشد، بزرگ‌ترین کارت خواهد بود.

در شکل ۲-۲۱ با اقتباس از همین روش کارنمایی برای یافتن بزرگترین عدد از بین N عدد ارائه شده است. با توجه به روش فوق باید به این نتیجه رسیده باشید، که اصلاً نیازی به نگهداری داده‌هایی‌که کوچکتر از بزرگترین دادهٔ موجود هستند، نیست. چنانکه در هر مرحله کارت کوچکتر را کنار می‌گذاشتیم. کارنمای ما تنها از جهت مرحلهٔ اول با روش فوق متفاوت است، چرا که ما اعداد را یکی‌یکی می‌خوانیم و در مرحلهٔ نخست دو عدد کارت را به‌طور همزمان برنمی‌داریم. به‌طوری که می‌بینید ابتدا با خواندن اولین عدد فرض می‌کنیم که این عدد، بزرگترین عدد موجود است، لذا آن عدد را در متغیر کمکی LRGST قرار می‌دهیم. سپس با خواندن عدد بعدی، چک می‌کنیم که اگر عدد خوانده شده از بزرگترین مقداری که تا اینجا به‌دست آورده‌ایم، بزرگتر است عدد جدید را در متغیر LRGST قرارمی‌دهیم در غیر این صورت عدد بعدی را می‌خوانیم. این روند تا جایی ادامه پیدا می‌کند که شمارندهٔ حلقهٔ تکرار به صفر نرسیده باشد. با توجه به استراتژی به‌کار رفته در حل این مسئله باید به نقش متغیرهای کمکی مثل LRGST پی برده باشید. در بسیاری از مسائل متغیرهای کمکی می‌توانند راه حل مسئله را بهینه سازند. متأسفانه هیچ معیار خاصی برای آنکه بگوییم در چه زمانی باید از متغیر کمکی استفاده کنید وجود ندارد، و چنانکه پیشتر گفته شد این مهارت رفته‌رفته با تجربه به‌دست می‌آید.

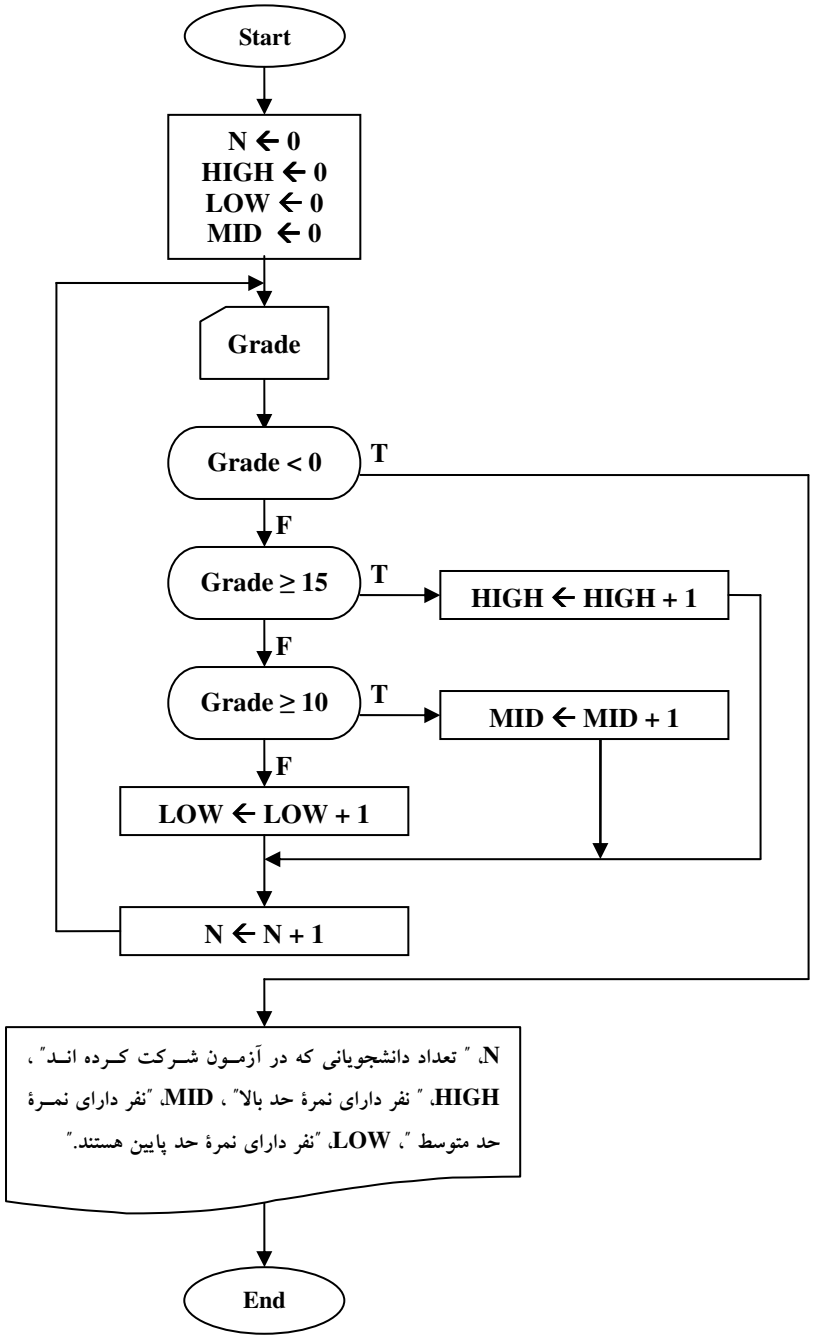


شکل ۲- ۲۱ : کارنماهای یافتن بزرگترین عدد از بین N عدد

الگوریتم شمارش نمرات

آخرین کارنمایی که در این مبحث ارائه می‌کنیم کارنمایی است که تعداد داده‌های متفاوتی را که خوانده شده است، را می‌شمارد. بر این اساس فرض کنید که تعداد نامشخصی نمره بین صفر تا ۲۰ را می‌خوانیم. الگوریتم باید تعداد نمرات بین ۱۵ تا۲۰ را به‌عنوان نمرات دستهٔ **A**، تعداد نمرات بین ۱۰ تا ۱۵ را به‌عنوان نمرات دستهٔ **B** ، تعداد نمرات پایین‌تر از ۱۰ را به‌عنوان نمرات دستهٔ **C** و تعداد کل نمرات خوانده شده را لیست کند. متغیرهای **HIGH** ، **MID** ، **LOW** متغیرهای شمارنده هستند که به ترتیب تعداد نمرات دسته‌های **A** و **B** و **C** را در خود ذخیره می‌کنند. برای هر مقدار ورودی متغیرِ **Grade** ، به یکی از سه شمارندهٔ فوق برحسب شرایط یک واحد اضافه می‌شود. همچنین متغیر دیگری مثل **N** ، تعداد نمرات خوانده شده را نگه می‌دارد. در اینجا نیز برای متوقف شدن حلقهٔ تکرار آخرین نمره باید یک نمره

خارج از دامنهٔ تعریف شدهٔ نمرات باشد. چرا که از روش نگهبان حلقهٔ تکرار استفاده شده است. مثلاً آخرین ورودی می‌تواند یک نمرهٔ منفی باشد.

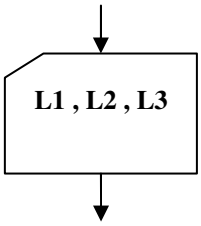


شکل ۲- ۲۲ : کارنمای شمارش حدود نمرات

۲-۷: تمرین

۱. کارنمای مقررات پستی خود، متعلق به مسئلهٔ ۳، از سری تمرینات ۲-۵ را طوری تغییر دهید که بتواند در موردی که ابعاد جعبه بدون ترتیب معینی وارد می‌شود (یعنی بزرگترین بُعد لزوماً اولی نباشد) کار کند.

راهنمایی: در هر دو مسئله، می‌توانید چنین فرض کنید که کارنما با قدم ورودی به شکل زیر شروع می‌شود به‌طوری که L1 و L2 و L3 طول سه یال جعبه را نشان می‌دهند. در مسئلهٔ اصلی از قبل معلوم بود که L1 باید بزرگترین باشد به‌طوری که اندازهٔ دور جعبه برابر 2*(L1+L2) می‌شد. در مسئلهٔ حاضر اندازهٔ دور جعبه برابر است با 2*(L1+L2+L3-X) که در آن X بزرگترین مقدار از بین L1 و L2 و L3 است.



۲. کارنمایی رسم کنید که مقادیر b و c را به‌عنوان ضرایب معادلهٔ bX + c = 0 دریافت کند و سپس در خصوص ریشه‌های این معادله برای حالت‌های زیر پیام خواسته شده را چاپ کند.

الف- اگر b برابر صفر و c مخالف صفر باشد، پیام "معادلهٔ bX + c = 0 ریشه ندارد." را چاپ کند.

ب- اگر b و c هر دو برابر صفر باشند، پیام "هر عدد حقیقی در bX + c = 0 صدق می‌کند." را چاپ کند.

ج- اگر b مخالف صفر باشد، ریشهٔ معادلهٔ bX + c = 0 را حساب کند و عبارت " ریشهٔ معادلهٔ bX + c = 0 برابر است با " و به دنبالش مقدار ریشه را چاپ کند.

۳. کارنمای الگوریتمی را رسم کنید که قد گروهی از مردم را دریافت کند و نزدیکترین قد به ۱۸۵ سانتی‌متر را چاپ کنید. فرض کنید که تعداد داده‌ها را ندارید اما آخرین دادهٔ ورودی صفر باشد.

۲–۸ : تقدم عملگرها در عبارات محاسباتی(جبری)

قابل پردازش کردن عبارت‌های محاسباتی(جبری) برای کامپیوتر

چنانکه در بخش‌های قبلی دیدید، محاسبات اساسی در کامپیوتر مشتمل بر محاسبهٔ مقدار عبارت‌های محاسباتی و رشته‌ای است که در جعبه‌های انتساب، جعبه‌های تصمیم و جعبه‌های خروجی یافت می‌شود. به‌عنوان مثالی برای عبارات محاسباتی، عبارت جبری زیر که مربوط به معادلات درجهٔ دوم می‌باشد را در نظر بگیرید:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

این نماد جبری ممکن است در زبان غیررسمی کارنما مورد قبول باشد ولی در هیچ زبان برنامه‌نویسی نمی‌توان چنین عبارتی را نوشت، در اینگونه زبان‌ها، عبارت‌ها باید به شکلی نوشته شوند که توسط ماشین قابل خواندن باشند. برای اینکه عبارت محاسباتی فوق را برای ماشین قابل خواندن کنیم، چندین تغییر به شرح زیر باید در عبارت مذکور اعمال کنیم:

۱. قبلاً مشاهده کردید که با توجه به قواعد مربوط به نامگذاری متغیرها، استفاده از کنار همنویسی متغیرها برای نشان دادن عمل ضرب را غیرممکن می‌کند. چرا که برای کامپیوتر **ac** یک متغیر جدید محسوب می‌شود و مفهوم **a*c** را ندارد. بنابراین برای نشان‌دادن صورتِ‌صحیح‌عبارت **4ac** باید بنویسیم **4*a*c** .

۲. برای آنکه عبارت محاسباتی توسط کامپیوتر قابل خواندن باشد، باید به‌صورت یک رشتهٔ خطی از نمادها نوشته شود. نمادهایی که از حالت خطی خارج شوند مثل توان در b^2 قابل قبول نیستند. بنابراین در زبان کارنما و یا برخی زبان‌های برنامه‌نویسی مثل **QBasic** عملگرهایی مانند ^ برای نشان دادن عمل به توان رساندن مورد استفاده قرار می‌گیرد. و عباراتی مثل b^2 را به‌صورت **b^2** نشان می‌دهند.

۳. استفاده از نمادهای خاص برای نشان دادن برخی توابع ریاضی عمل خواندن توسط ماشین را پیچیده می‌کند، مثل علامت جذر در عبارت فوق. بنابراین می‌توان توابع ریاضی را با حروفی که معرف آن توابع باشند نشان داد. چنان که بهتر است به جای علامت جذر از حروف **SQRT** استفاده کرد. و در نتیجه عبارت ریاضی:

$$\sqrt{b^2 - 4ac}$$

به‌صورت :

SQRT (b^2 – 4*a*c)

نشان داده می‌شود. به همین نحو، به‌جای علامت قدرمطلق در عبارت جبری مثل |R| در اکثر زبان‌های برنامه نویسی نوشته می‌شود **ABS(R)**. اما کامپیوتر از کجا بفهمد که **ABS** و **SQRT** نام یک متغیر نیستند بلکه نشان-دهندهٔ توابع‌اند!؟ جواب این سؤال نیز بسیار روشن است. از آنجایی که علامت پرانتز باز جزء کاراکترهای مجاز جهت نامگذاری متغیرها نیست، می‌فهمد که این حروف نام تابع هستند. از طرفی ورودی‌های توابع را بین دو پرانتز باز و بسته قرار می‌دهیم تا کامپیوتر بداند برروی چه مقدار یا مقادیری باید عمل کند. مثلاً به‌جای عبارت جبری پر کاربرد **SinX** که در ریاضیات به همین صورت نوشته می‌شود، در الگوریتم‌های خود بنویسید **SIN(X)**. البته برخی

از توابع ممکن است به بیش از یک ورودی نیاز داشته باشند که در این صورت ورودی‌ها را با علامت ویرگول از یکدیگر جدا می‌کنیم. مثلاً تابع MIN(a,b) می‌تواند تابعی باشد که مـقدار کوچکتر را در بین دو ورودی a و b به دست می‌دهد. و یا تابعی مـثل POW(a, b) می‌تواند تابعی باشد که a را به توان b می‌رساند، و در نتیجه می‌توان عبارتی مثل b^2 را به‌صورت POW(b, 2) نیز نشان داد. البته باید توجه داشته باشید که در چنین توابعی ترتیب آمدن ورودی‌ها حائز اهمیت است. چرا که مقداری که توسط تابع POW(b,2) بهدست می‌آید با مقداری که توسط تابع POW(2, b) به‌دست می‌آید به کلی متفاوت است. اما در مورد تابعی مثل MIN این چنین نیست و ترتیب آمدن ورودی‌ها اهمیت ندارد.

۴. همچنین در عبارت بالا تقسیم بر 2a را با نمادی که از حالت خطی خارج شده است نشان دادیم. هنگامی که قرار باشد ماشین این عبارت را بخواند، بایستی از نماد کسری صرف‌نظر کنیم و برای نشان دادن تقسیم از عملگری خطی مثل (/) استفاده کنیم.

هنگامی‌که تمامی این تغییرات را اعمال کنیم، عبارت مربوط به ریشه‌های معادلهٔ درجهٔ دوم به‌صورت زیر در می‌آید:

```
(–b + SQRT(b^2 – 4*a*c)) / (2*a)
```

۵. توجه داشته باشید هنگامی‌که خط کسری را به علامت / تبدیل کردیم، مجبور شدیم دو جفت پرانتز اضافی نیز به کار ببریم، یک جفت برای دربرگرفتن صورت و جفت دیگر برای دربرگرفتن مخرج. اگر هر یک از این پرانتزها حذف شوند، عبارت به‌طور صحیح محاسبه نخواهد شد. مثلاً اگر پرانتزهای اطراف مخرج را حذف کنیم، آن وقت عبارت حاصله به معنای زیر خواهد بود:

$$\frac{-b + \sqrt{b^2 - 4ac}}{2 * a}$$

کار در کلاس ۲-۴ :

مطابق مثال فوق عبارت جبری حاصله از هر یک از موارد زیر را تعیین کنید؟

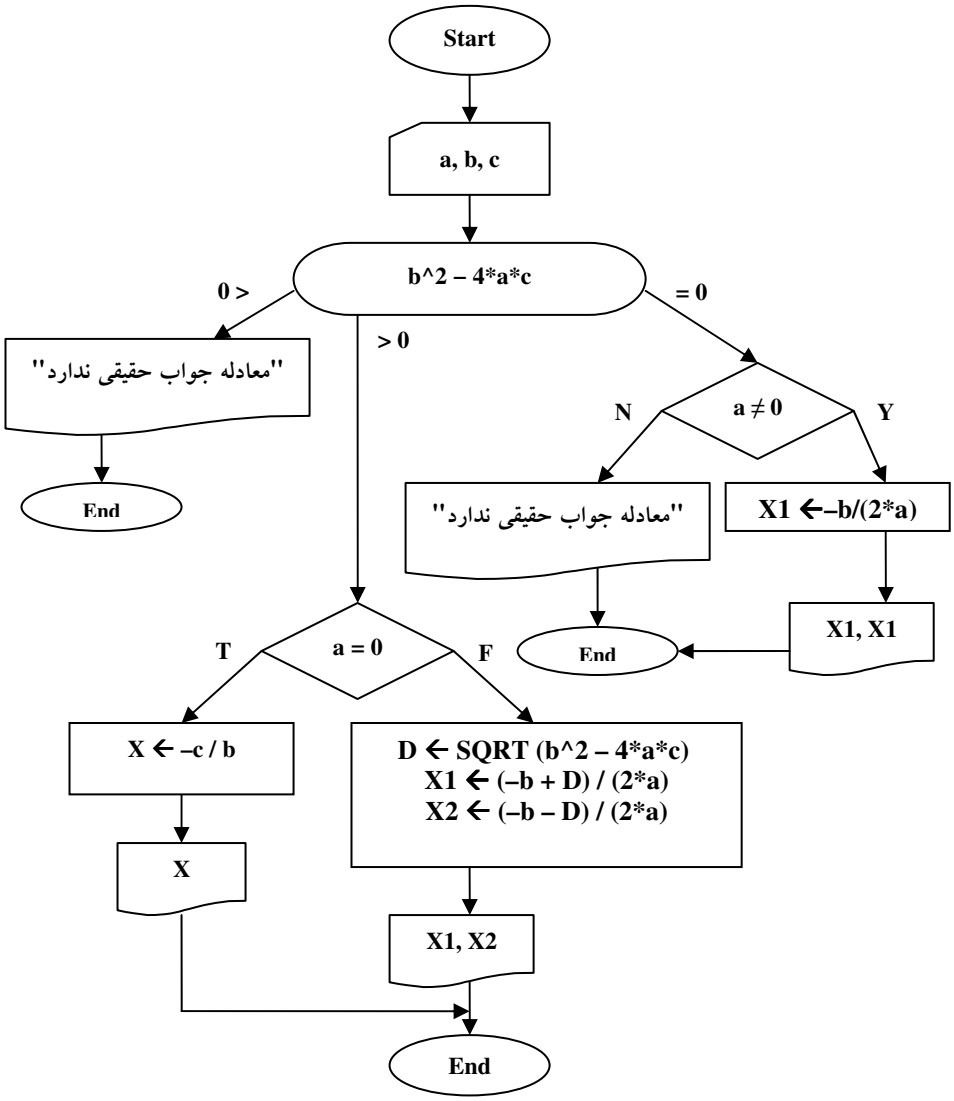
راهنمایی: باید به تقدم عملگرها توجه خاص داشته باشید.

۱. `–b + SQRT(b^2 – 4*a*c) / (2*a)`

۲. `–b + SQRT(b^2 – 4*a*c) / 2*a`

۶. نصیحت مفیدی که می‌توانیم در مورد قرار دادن یا قرار ندادن پرانتز بکنیم این است که: « اگر شک دارید پرانتز لازم است یا نه، بنا را بر لزوم آن بگذارید». لزوم و یا عدم لزوم پرانتز را می‌توان با مراجعه به قواعد تقدّم برای انجام محاسبات در غیاب پرانتز، تعیین نمود که در مبحث بعدی به تفصیل در این خصوص صحبت خواهیم کرد.

مثال ۲-۶ : کارنمایی که با دریافت ضرایب معادلهٔ درجه دوم $aX^2 + bX + c=0$ ریشه‌های این معادله را چاپ می‌کند. به کاربرد جعبه تصمیم چندگانه و طریقهٔ نوشتن عبارات جبری توجه کنید.



شکل ۲-۲۳ : کارنمای مثال ۲-۶

تقدم عملگرهای حسابی

قواعدی وجود دارد که تقدم عملگرهای حسابی را در عبارات محاسباتی در غیاب پرانتز مشخص می‌کند. این قواعد که به قواعد تقدّم عملگرهای حسابی موسوم هستند در مواردی که نظیر آنها در قراردادهای معمولی ریاضی وجود داشته باشد با آنها مطابقت دارند. مثلاً در محاسبهٔ عبارت محاسباتی

A + B * C

به ازای مقادیر **A=5 , B=3 , C=2** ، از آنجایی که ضرب بر جمع تقدم دارد ابتدا **B** را در **C** ضرب کرده و سپس **A** را با حاصل ضرب به‌دست آمده جمع می‌کنیم. بنابراین داریم:

A + B * C = A + (3 * 2) = A + 6 = 5 + 6 = 11

اگر بخواهیم که ابتدا **A** با **B** جمع شود، باید از پرانتز استفاده کنیم به این ترتیب عبارت **(A + B) * C** به‌صورت زیر محاسبه می‌شود.

(A + B) * C = (5 + 3) * C = 8 * C = 8 * 2 = 16

فهرست کامل سطوح تقدم عملگرهای حسابی در جدول ۲-۱ نشان داده شده است.

نشان عملگر	نام عملگر	سطح تقدم
^	به توان رساندن	بالاترین تقدم
*	ضرب	بعدی
/	تقسیم	
–	منفی کردن (یک تایی)	
+	جمع	پایین‌ترین تقدم
–	تفریق (دوتایی)	

جدول ۲- ۱ : جدول سطوح تقدّم عملگرهای حسابی

تذکر: همانگونه که در جدول فوق مشاهده می‌کنید، علامت منها در دو مکان به‌کار رفته است. در حقیقت منها دارای سه کاربرد مختلف به شرح زیر است: گاهی اوقات به‌عنوان جزئی از یک عدد منفی به کار برده می‌شود مثل عدد ۵- ، برخی از اوقات نیز به‌عنوان یک عملگر یک‌تایی جهت منفی کردن یک متغیر به کار برده می‌شود مثل **X–** و در نهایت می‌تواند به‌عنوان یک عملگر دوتایی تفریق به‌کار برده شود مثل **X–5** . اما ماشین از کجا می‌تواند تفاوت این سه عملگر را بفهمد؟ در واقع نقش منها را می‌توان با وضعیت قرار گرفتنش در عبارت تعیین کرد. بدین صورت که:

« منها همواره دوتایی است مگر وقتی که در شروع عبارت و یا بلافاصله پس از پرانتز باز بیاید. اگر منها دوتایی نبود و پس از آن یک رقم آمده باشد قسمتی از یک عدد و در غیر این صورت عملگر یک‌تایی منفی‌کردن است.» در عبارت زیر هرسهٔ این حالت‌ها نشان داده شده است. آیا می‌توانید نقش هریک از علامت‌های منها را در عبارت زیر نام ببرید؟

X – (– (–5))

ارزیابی عبارت‌های محاسباتی بدون پرانتز

در این قسمت می‌خواهیم با ارائهٔ مثالی ببینیم به واسطهٔ جدول ۲-۱ چگونه می‌توان عبارت‌های محاسباتی را مورد ارزیابی قرار داد.

کامپیوتر برای ارزیابی یک عبارت محاسباتی ابتدا عبارت را برای یافتن عملگری با بالاترین سطح تقدّم مورد پویش(پردازش) قرار می‌دهد. ولی این پویش چگونه صورت می‌گیرد؟ طبق قرار داد هر عبارت ریاضی از سمت چپ پویش می‌شود و مثلاً در عبارتی به‌صورت **A + B + C** با آنکه، دو عملگر جمع به‌کار رفته در آن، دارای تقدم یکسانی هستند، ولی عملگر سمت چپ دارای تقدم بالاتری است. چرا که در هنگام پویشِ عبارت، عملگر سمت چپ زودتر خوانده می‌شود. به‌عبارت دیگر دو عملگر با سطح تقدم یکسان دارای تقدم مکانی نسبت به هم هستند. اما با آنکه عبارت جبری **A + B * C** از سمت چپ پویش می‌شود و عملگر جمع زودتر خوانده می‌شود این عملگر ضرب است که مطابق جدول ۲-۱ دارای اولویت بالاتری است و زودتر عمل می‌کند. اصولاً پرانتزها قادرند اولویت انجام محاسبات را تغییر دهند. بدین صورت که گفته می شود داخلی ترین پرانتز دارای اولویت بالاتری است. بنابراین در عبارتی مثل **A * (B + (C − D))** ابتدا **(C − D)** که داخلی‌ترین پرانتز است مورد پردازش قرار می‌گیرد سپس مقدار **B** با حاصل تفریق، جمع شده و در نهایت حاصلِ **(B + (C − D))** با **A** در هم ضرب می‌شوند. اما در عباراتی که فاقد پرانتز هستند عملیات پویش را به‌صورت زیر باید انجام دهید:

۱. اول عبارت را برای پیدا کردن عملگرهایی با بالاترین سطح تقدم از چپ به راست بپویید.
۲. وقتی عملگری از این سطح پیدا کردید، عمل قید شده را انجام دهید و پس از جایگزین کردن حاصل به‌دست آمده، عمل پویش را از همان محلی که رها شده بود برای عملگرهای همان سطح تقدم ادامه دهید.
۳. وقتی که تمام عبارت محاسباتی را برای عملگرهایی با بالاترین سطح تقدم مورد پویش قرار دادید، پویش را دوباره از چپ به راست برای عملگرهای سطح تقدم بعدی تکرار کنید و به همین ترتیب ادامه دهید تا جواب نهایی عبارت به‌دست آید.

کار در کلاس ۲-۵ :

مطابق آنچه که در بالا گفته شد مقدار عبارت **A*L–G*O^R / I+T / H*M–** *را با توجه به جدول زیر محاسبه کنید. سپس عبارت فوق را به گونه ای پرانتز گذاری کنید که چگونگی مراحل محاسبه را نشان دهد.*

راهنمایی : (اولین عملیاتی که صورت می گیرد باید در داخلی ترین بافت پرانتز قرار گیرد.)

A	L	G	O	R	I	T	H	M
5	6	8	2	3	4	6	2	7

در جدول ۲-۲ مراحل پردازش عبارت محاسباتی مربوط به کار در کلاس ۲-۵ به‌صورت قدم به قدم نشان داده شده است. علامت مثلث شکل کوچک (▲) برای نشان دادن عملگری که باید در مرحلهٔ بعدی مورد محاسبه قرارگیرد، استفاده شده است. همان‌طور که می‌بینید مطابق سطوح تقدم آمده در جدول ۲-۱ ابتدا عملگرهای سطح اول، سپس عملگرهای سطح دوم و سپس عملگرهای سطح سوم مورد پردازش قرارگرفته‌اند.

عبارت حاصل بعد از عمل	عمل مورد پردازش	قدم	عملیات مربوط به سطحِ
–A*L–G*O^R / I+T / H*M ▲	عبارت اولیه	۰	بالاترین سطح تقدم
–A*L–G*8 / I+T / H*M ▲	O^R	۱	
–5*L–G*8 / I+T / H*M ▲	–A	۲	سطح تقدم میانی
–30 – G*8 / I+T / H*M ▲	– 5*L	۳	
–30 – 64 / I+T / H*M ▲	G*8	۴	
–30 – 16 +T / H*M ▲	64 / I	۵	
–30 – 16 + 3*M ▲	T / H	۶	
–30 – 16 + 21 ▲	3*M	۷	
–46 + 21 ▲	– 30 – 16	۸	پایین‌ترین سطح تقدم
–25	– 46 + 21	۹	

جدول ۲ – ۲ : نمایش قدم به قدم محاسبهٔ عبارت جبری مربوط به‌کار در کلاس ۲ – ۵

توجه کنید که در قدم ۲، A– به ۵– تبدیل شده و با وجودی که علامت منهای واقع در A– یکتایی است، علامت منهای واقع در ۵– قسمتی از عدد است و دیگر در محاسبات شرکت نمی‌کند. زیرا منهای مربوط به اعداد منفی عملگر نیست بلکه جزئی از خود عدد است.

تذکّر مهم: اگر قواعد مندرج در جدول ۲-۱ همراه با پویش از چپ به راست رعایت شوند فقط یک مورد وجود دارد که، ترتیب محاسبهٔ عبارت‌های محاسباتی، با قراردادهای معمولی ریاضی مطابقت ندارد. قواعدی که در این جدول ذکر شده‌اند عبارت A^B^C را به‌صورت (A^B)^C محاسبه می‌کند درحالی که در ریاضیات، قرار داد است که عبارت A^B^C به‌صورت A^(B^C) محاسبه شود. اگر این تفاوت را در نظر داشته باشید، همیشه می‌توانید مقصود خود را با استفاده از پرانتز اعمال کنید. علت این که با این مشکل برخورد کردیم از آن جهت است که تمامی عملگرها مثل جمع، تفریق، ضرب و تقسیم در عملگرِ سمتِ چپ، دارای اولویت بالاتری از نظر مکانی هستند ولی دو عملگر توان و منهای یکانی در عملگرِ سمتِ راست، دارای اولویت بیشتری هستند.

قواعد ارزیابی عبارت‌های محاسباتی دارای پرانتز

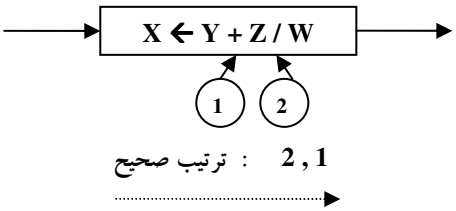
تمام روش‌های محاسبهٔ عبارت‌ها، مستلزم داشتن درجات تقدم برای عملگرها نیستند. مثلاً می‌توانستیم صرفاً چنین توافق کنیم که عبارت‌ها را از چپ به راست (یا برعکس) بپوییم و به هر عملگری که رسیدیم نتیجه را درجا تعیین کنیم. مثلاً در زبان قدیمی **APL** پویش را از راست به چپ و بدون رعایت درجات تقّدم انجام می‌دهد. مسلماً چنین روشی را راحت‌تر می‌توان در عمل پیاده‌سازی کرد. از طرف دیگر، این روش باعث می‌شود که ترتیب اجرای عملیات غالباً با قراردادهای معمولی ریاضی متفاوت باشد. مثلاً در **APL** عبارت **A*B+C** به‌صورت **A*(B+C)** محاسبه می‌شود. در برنامه‌های کامپیوتری مثل مفسرها و مترجم‌ها با اعمال تغییراتی برروی یک عبارت محاسباتی، آن عبارت را به فرمی تبدیل می‌کنند که بدون پویش‌های مکرر، با یکبار پویش، عبارت موردنظر را مورد محاسبه قرار دهند، ضمن آنکه درجات تقدم هم رعایت می‌شود.

با توجه به آنچه که پیشتر در خصوص پردازش عبارات دارای پرانتز گفتیم می‌توانیم روشی را برای محاسبهٔ عبارت‌های دارای پرانتز به‌صورت زیر بیان کنیم. با ذکر این نکته که عبارت موجود بین یک جفت پرانتز باز و بسته را *زیر عبارت* می‌گویند.

۱. عبارت را به‌منظور یافت اولین پرانتز بسته از سمت چپ بپوی.
۲. زیر عبارتی را که به پرانتز بستهٔ فوق ختم می‌شود را طبق قواعد محاسبهٔ عبارات بدون پرانتز (مطابق جدول ۲- ۱) محاسبه کن.
۳. اگر قبل از این زیر عبارت نام یک تابع آمده است مقدار تابع ذکر شده را برای حاصل این عبارت حساب کن.
۴. در صورتی که به یک مقدار نهایی نرسیده‌ای به قدم یک بازگرد.

۲-۹: تمرین

۱. در هر یک از قسمت‌های (آ) تا (ت) این مسئله، جعبهٔ انتسابی از یک کارنما به شما داده شده است. وظیفهٔ شما این است، که با استفاده از قواعد تقدّم (که در جدول ۲-۱ آمده است) ترتیب انجام عملیات حسابی را در هنگام اجرای جعبه تعیین کنید. برای سهولت در مراجعه، مطابق مثال زیر، شماره‌ای برای هر عملگر در زیر آن قرار دهید، سپس به ترتیب اجرای هر عملگر شمارهٔ عملگرها را از چپ به راست بنویسید.



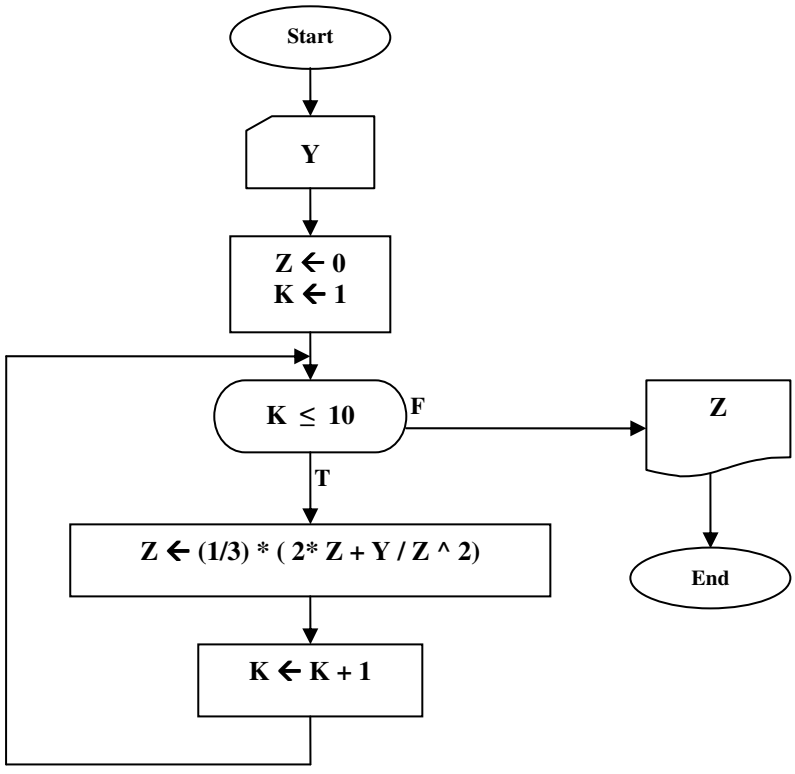
(الف) `I ← L ^ O / V / E – B / E * T * T – Y`

(ب) `I ← W + H ^ E / A * T * P – E ^ S ^ K`

(پ) `I ←S ^ E * Y ^ N / A * E ^ S * S – B`

(ت) `I ←B *C /D ^ C / E ^ C – F *G ^ H`

۲. کارنمای زیر از تمام قواعد کارنمایی ما پیروی می‌کند. ولی اگر به یک برنامهٔ کامپیوتری تبدیل و اجرا شود باعث می‌شود که کامپیوتر پیغام خطا دهد. توضیح دهید اشکال کار در کجاست.



شکل ۲-۲۴ : کارنمای تمرین ۲

۲–۱۰ : چند تابع ریاضی مهم

تابع جزء صحیح (براکت ، کف)

با تابع براکت یا جزء صحیح یک عدد در دروس ریاضی دوره دبیرستان آشنا شده‌اید. تابع براکت یکی از مهمترین توابع در حل بسیاری از مسائل الگوریتم‌نویسی است. چنانکه کاربردهای متعددی از قبیل انجام تقسیم صحیح، یافتن باقی‌مانده تقسیم دو عدد بر یک دیگر و گردکردن اعداد دارد. در این کتاب از سه حرف FLR[1] به‌عنوان نمادی برای تابع جزء صحیح استفاده شده است. تنها به جهت یادآوری مقدار جزء صحیح چند عدد را به‌دست می‌آوریم:

1) FLR (35.94) = 35 **2) FLR (π) = 3** **3) FLR (7⅔) = 7**
4) FLR (– 0.6) = –1 **5) FLR (8) = 8** **6) FLR (–2) = –2**

حال به بیان مثال‌هایی از کاربردهای تابع براکت می‌پردازیم.

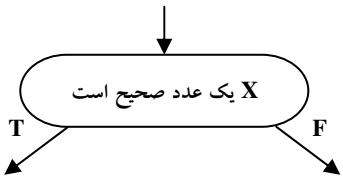
۱. انجام تقسیم صحیح

منظور از تقسیم صحیح آن است که با تقسیم مقسوم بر مقسوم‌علیه تنها، خارج قسمت تقسیم به‌دست آید و از باقی‌ماندهٔ آن صرف‌نظر شود. به‌عنوان مثال حاصل تقسیم صحیح ۱۴ بر ۵ برابر ۲ است که به‌صورت زیر نشان داده می‌شود:

FLR (14 / 5) = 2

۲. تشخیص عدد صحیح

غالباً در رسم الگوریتم‌ها آزمون‌هایی از قبیل



انجام می‌دهیم. برای تشخیص صحیح بودن یک عدد می‌توان از عبارت شرطی زیر در جعبه‌های تصمیم استفاده کرد:

X = FLR (X)

در صورتی که متغیر X شامل عددی صحیح باشد مقدار آن با مقدار جزء صحیح خود برابر است.

۳. گرد کردن اعداد

در دورهٔ دبیرستان با گـردکردن آشنا شده‌اید. برای گردکردن اعداد اعشاری که قسمت اعشاری آنها کمتر از **0.5** بود تنها قسمت صحیح عدد را در نظر می‌گرفتیم. و اعدادی را که قسمت اعشاری آنها بیشتر از **0.5** بود را به عدد صحیح بعدی گرد می‌کردیم. به‌عبارت دیگر می‌توان از فرمول زیر برای گردکردن عدد تا رقم یکان استفاده نمود. در این فرمول ROUND نام تابع گردکننده است.

ROUND (X) = FLR (X + 0. 5)

---

۱. مخففی برای کلمهٔ Floor به معنای کف. در کتب ریاضی از علامت [ ] برای نشان دادن تابع جزء صحیح استفاده می‌کنند.

همانگونه که از ظاهر فرمول برمی‌آید روش فوق تنها قادر به گردکردن عدد تا رقم یکان است. اما در بسیاری از موارد پیش می‌آید که نیازمند گردکردن تا یک رقم اعشار، دو رقم اعشار، سه رقم اعشار و... باشیم. در اینگونه موارد ابتدا تعداد رقم‌های اعشاری موردنظر را با ضرب کردن عدد در توان‌های ۱۰ به قسمت صحیح عدد منتقل می‌کنیم. سپس با استفاده از فرمول فوق حاصل‌ضرب را گرد کرده، و در نهایت با تقسیم عدد گرد شده بر همان توان ده عدد را به‌صورت اعشاری تبدیل می‌کنیم. به‌عنوان مثال برای گرد کردن عدد اعشاری **12.7896354** تا سه رقم اعشار به‌صورت زیر عمل می‌کنیم.

```
ROUND (12.7896354) = FLR (12.7896354 * 10 ^ 3) / 10 ^ 3
                   = FLR (12789.6354) / 1000
                   = 12789 / 1000
                   = 12.789
```

یک تذکر : در کامپیوترهایی با سازمان کلمه‌ای (بایتی)، تمام محاسبات را به وسیلهٔ روشی مشابه روش فوق گرد می‌کنند تا نتایج به‌دست آمده در یک کلمهٔ کامپیوتر بگنجد. ما در زبان کارنمای خود، معمولاً اثر گردکردن را به حساب نمی‌آوریم و چنین وانمود می‌کنیم که کلمه‌های کامپیوتر آنچنان ظرفیتی دارند که تمام عملیات و محاسبات به‌طور دقیق انجام می‌شود. ولی قبل از ترجمهٔ کارنما به زبان‌های برنامه‌سازی غالباً لازم است که جعبه‌های تصمیم نظیرِ

```
B^2 – 4*A*C = 0
```

را با شکل‌های تقریبی مثل شکل زیر جایگزین کنیم.

```
|B^2 – 4*A*C| < 0.0001
```

شما چه نتیجه‌ای از بحث فوق می‌گیرید؟

کار در کلاس ۲-۶:

کارنمایی رسم کنید که زمان دویدن دونده ای را بر حسب ثانیه دریافت کند و سپس معین کند آن دونده چند ساعت و چند دقیقه و چند ثانیه دویده است؟ سپس الگوریتم خود را برای **S=8534** *آزمایش کنید.*

تابع باقی ماندهٔ تقسیم

یکی از تکنیک‌های بسیار مهمی که ما را در حل‌کردن برخی از الگوریتم‌ها کمک می‌کند یافتن باقی‌ماندهٔ تقسیم بر یک عدد است. پیشتر در مثال ۲-۵ دیدید که با به‌دست آوردن باقی‌ماندهٔ تقسیم عدد X بر دو توانستیم زوج یا فرد بودن X را احراز کنیم. چنانکه در این مثال دیدید برای به‌دست آوردن باقی ماندهٔ تقسیم، از تابع براکت [ویژگی تقسیم صحیح آن] استفاده کردیم. به‌طور کلی برای به‌دست آوردن باقی‌ماندهٔ تقسیم X بر Y از روش زیر استفاده می‌کنیم:

ابتدا خارج قسمت تقسیم این دو عدد را به‌دست می‌آوریم: **Q = FLR (X / Y)**

پس باقی ماندهٔ تقسیم را به‌صورت زیر به‌دست می‌آوریم: **R = X – Q*Y**

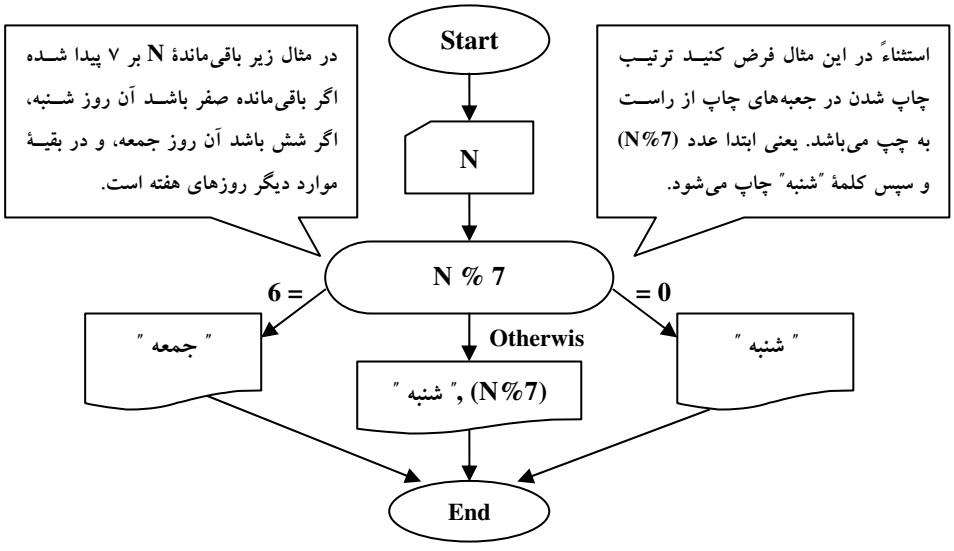
چنانکه می‌بینید به‌دست آوردن باقی‌ماندهٔ تقسیم یکی از کاربردهای تقسیم صحیح است. اما به جهت اهمیت و کاربرد زیادی که یافتن باقی‌ماندهٔ تقسیم در حل برخی مسائل دارد، در اکثر زبان‌های برنامه‌نویسی تابعی با نام **MOD** جهت یافتن باقی‌ماندهٔ تقسیم X بر Y تعریف شده است. این تابع دارای دو ورودی به‌صورت زیر است.

MOD (X , Y)

اما در زبان **C++** چنانکه در فصل آتی خواهید دید نماد % را به‌عنوان یک عملگر برای یافتن باقی‌ماندهٔ تقسیم تعریف شده است. از آنجایی که در این کتاب زبان **C++** نیز تدریس شده، ما نیز به جای تابع **MOD** از عملگر % برای نشان دادن عمل باقی‌ماندهٔ تقسیم استفاده کرده‌ایم. از نظر اولویت نیز این عملگر در ردیف ضرب و تقسیم قرار می‌گیرد. حال باقی‌ماندهٔ تقسیم چند عدد بر یکدیگر را به‌دست می‌آوریم:

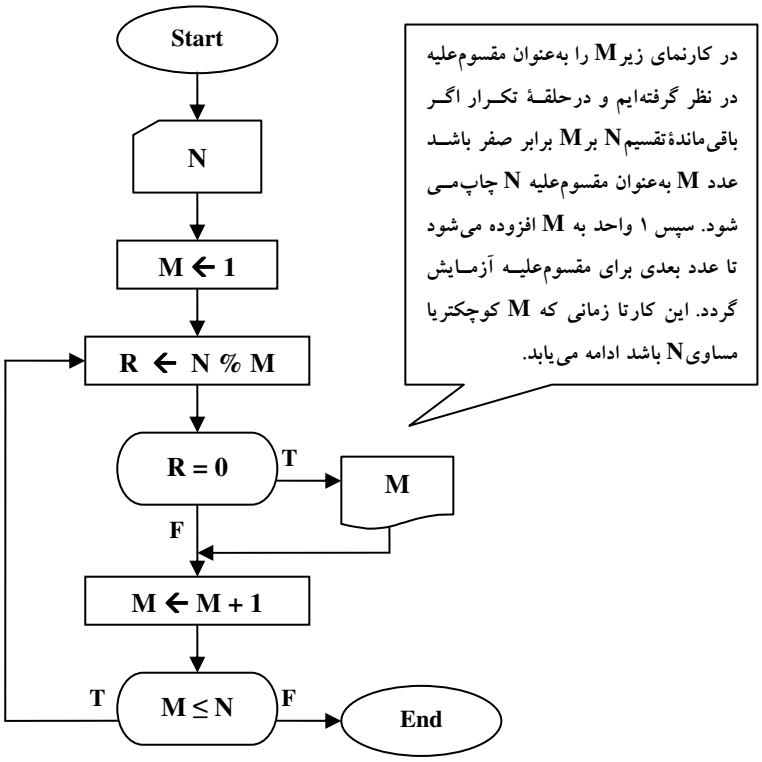
5 % 2 = 1 **8 % 2 = 0** **9 % 3 = 0** **11 % 5 = 1** **2 % 4 = 2**

مثال ۲-۷ : روز اول سال، یکشنبه است. کارنمایی رسم کنید که معین کند روز N اُم سال چه روزی از هفته است. (مثلاً، روز چهاردهم شنبه است).



شکل ۲-۲۵ : کارنمای مثال ۲-۷

مثال ۲-۸: کارنمایی رسم کنید که با دریافت عدد طبیعی N کلیهٔ مقسوم‌علیه‌های آن را چاپ کند؟



شکل ۲-۲۶ : کارنمای مثال ۲-۸

کار در کلاس ۲-۷ :

کارنمایی رسم کنید که با دریافت یک عدد بگوید آن عدد تام است یا نه؟

راهنمایی : عددی تام است که مجموع مقسوم علیه های کوچک تر از خودش برابر خود عدد باشد.

۲-۱۱: تمرین

۱. آیا می‌توانید کارنمای مثال ۲-۸ را به‌گونه‌ای بهینه‌سازی کنید که تعداد تکرار بیرونی ترین حلقهٔ تکرار به نصف کاهش پیدا کند؟

۲. قسمت‌های (۱) تا (٦) این مسئله مربوط به کارنمایی است که در شکل ۲-۲۷ آمده است.

۱)درست یا نادرست: به‌ازای هربار که جعبهٔ ۱ اجرا می‌شود، جعبهٔ ٦ چهار بار اجرا می‌شود.

۲)اگر ٦ جفت مقدار (هر جفت شامل یک مقدار برای N و یک مقدار برای L) به‌عنوان ورودی داده شود، تعداد مقادیر چاپ شده توسط برنامه:

(۱) بستگی به مقدار واقعی داده‌ها دارد.

(۲) دقیقاً ۱۸ تاست.

(۳) حداقل ٢٤ تاست.

(٤) هیچ مقداری چاپ نخواهد شد.

(٥) دقیقاً ۳ تاست.

۳) فرض کنید که ورودی‌های این برنامه به ترتیب از چپ به راست به‌صورت زیر باشد:

٣٢٤ و ۷۹۲- و ۳۲۳ و۷۹۱

کدام دسته از مقادیر زیر توسط برنامه چاپ خواهد شد:

(۱) ۱۹۷ و ۷٦۰ و٤۳۷ و۱۱٤

(۲) ٦۹٦ و ۹۰٥ و ۱۱٤

(۳) ۹۸۸ و ٥٥۱ و ٤۳۷ و ۱۱۱٤

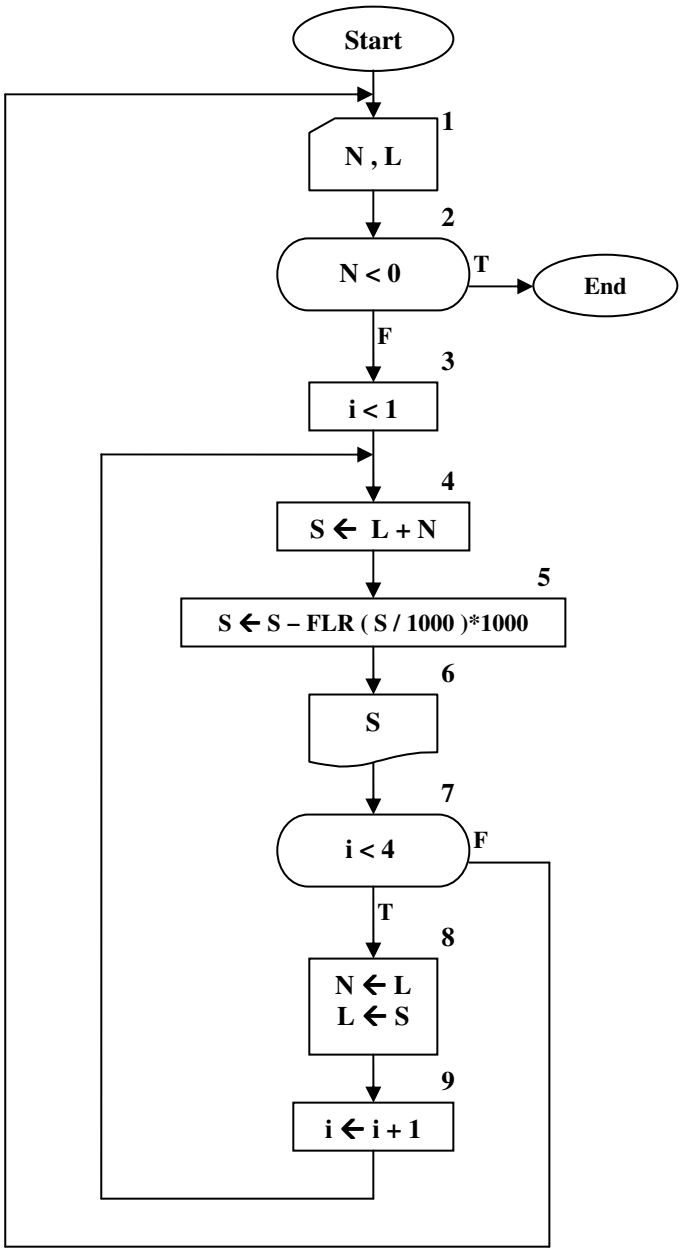
(٤) ۹۸۸ و ٥٥۱ و ٤۳۷ و ۱۱٤

(٥) ۹۸۸ و ٦۰۱ و ٦۹٦ و ۹۰٥ و ۱۱۱٤

٤) درست یا نادرست: حلقهٔ داخلی کارنما، یعنی جعبه‌های ٤ تا ۹ ، حلقه‌ای است که توسط شمارنده کنترل می شود.

٥) درست یا نادرست: اگر فرض کنیم مقادیر منفی N غیر مجاز هستند، حلقهٔ خارجی کارنما، یعنی جعبه‌های ۱ تا ۷ ، حلقه‌ای است که به وسیلهٔ نگهبان کنترل می‌شود.

٦) چگونه می توان جعبه‌های ٤ و ٥ کارنما را به جعبه‌ای با یک انتساب واحد تبدیل کرد بدون اینکه اثر مهمی برروی برنامه و یا نتیجهٔ آن بگذارد؟



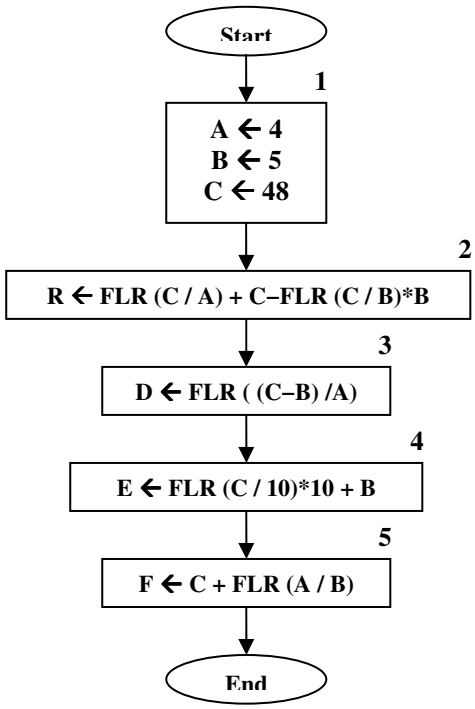
شکل ۲-۲۷ : کارنمای تمرین ۲

۳. کارنمایی رسم کنید که یک مجموعهٔ صدتایی از مقادیر داده را خوانده و تمام اعداد زوج به استثنای مضارب ۱۰ را در خروجی بنویسد. مطمئن شوید که کارنمای شما دقیقاً ۱۰۰ مقدار را می‌خواند.

٤. برای هریک از پنج دستور انتساب زیر مقادیر متغیرهای مربوط به قبل از اجرای دستور العمل انتساب داده شده است. وظیفهٔ شما تعیین مقداری است که در هر دستور به **T** نسبت داده می‌شود.

(1) $T \leftarrow I \times J + K$, I=11, J=3, K=75
(2) $T \leftarrow FLR\ (\ (I - J\ )\ /\ T\ )$, I=52 , J=7.9 , T=2.1
(3) $T \leftarrow FLR\ (\ I\ /\ J\ +\ K\ )$, I=50 , J=82 , K=−1
(4) $T \leftarrow N - FLR\ (\ N\ /\ D\ )\ *\ D$, N=10, D=2
(5) $T \leftarrow N\ /\ 3\ -\ FLR\ (\ (\ N\ /\ 3\ )\ /\ D\ )\ *\ D$, N=14 , D=7

۵. کارنمای زیر را پیگیری و جمله‌های زیر را کامل کنید:



شکل ۲-۲۸ : کارنمای تمرین ۵

الف- پس از اجرای جعبهٔ ۲ ، مقدار **R** برابر است با ...

ب- پس از اجرای جعبهٔ ۳، مقدار **D** برابر است با ...

پ- پس از اجرای جعبهٔ٤ ، مقدار **E** برابر است با ...

ت- وقتی به **End** برسیم، مقدار **F** برابر است با...

۶. کارنمایی رسم کنید که تعدادی عدد صحیح را، یکی یکی به داخل آورده و به‌ازای هر مقدار، یک عدد دو رقمی چاپ می‌کند که تشکیل شده است از دو رقم سمت راست (یعنی رقم‌های یکان و دهگان) مقدار ورودی، تعدادی عدد صحیح مثبت به‌عنوان داده در اختیار داریم که یکی یکی باید خوانده شوند و آخری‌ن ورودی ی‌ک عدد منفی است.

۷. آیا می‌توانید در مورد نحوهٔ تشخیص قابل قسمت بودن یک عدد بر عدد دیگر توضیح دهید، کارنمایی رسم کنید که مقسوم‌علیه‌های هر عدد ورودی **n** را چاپ کند.

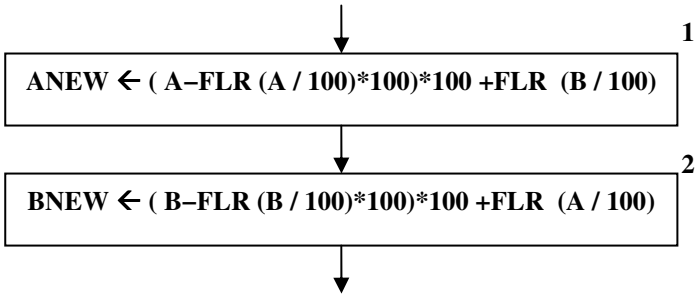
۸. الگوریتمی رسم کنید که مضارب کوچکتر از ۱۰۰۰۰ عدد ورودی **n** را محاسبه و چاپ کند.

۹. الگوریتم تمرینات قبل را به‌گونه‌ای گسترش دهید که بزرگترین مقسوم‌علیه مشترک دو عدد را یافته و چاپ نماید.

۱۰.الگوریتم تمرین قبل را به‌گونه‌ای گسترش دهید که کوچکترین مضرب مشترک دو عدد دریافتی را نیز بیابد.

راهنمایی : هدف از این تمرین آن است که توجه شما را به این نکته جلب کنیم که همیشه استفاده از روش‌های الگوریتمی و تکرار مناسب‌ترین راه حل نیست، به‌طوری که با توجه به قواعد ریاضی می‌توانید ک.م.م را با توجه به ب.م.م محاسبه کنید.

۱۱.**A** و **B** اعداد صحیح چهار رقمی هستند. کارنمای ناتمام شکل ۲-۲۹ داده شده است.



شکل ۲-۲۹ : کارنمای تمرین ۱۱

الف- اگرمقدار **A** برابر ۱۴۶۸ و مقدار **B** برابر ۲۳۵۷ باشد، مقدار **ANEW** پس از اجرای جعبهٔ ۱ چیست؟

ب- به ازای همین مقادیر **A** و **B** ، مقدار **BNEW** پس از اجرای جعبهٔ ۲ چیست؟

۱۲.مقدار**ROUND(X)** را وقتی که **X** مضرب فردی از ۰/۵ باشد تعیین کنید.

۱۳.عدد **N** مفروض است. کارنمایی رسم کنید که مجموع ارقام عدد **N** را به‌دست آورد.

۱۴.کارنمایی رسم کنید که مقادیر را دوتا دوتا خوانده و عدد جدیدی متشکل از دو رقم سمت چپ عدد اول و دو رقم سمت راست عدد دوم ساخته و چاپ کند. این پردازش برای جفت‌های بعدی اعداد ورودی نیز به همین نحو تکرار می‌شود. تعدادی عدد صحیح مثبت چهار رقمی به‌عنوان داده در اختیار داریم و آخرین ورودی یک عدد منفی است.

۱۵.هزینهٔ ارسال نامهٔ هوایی به ازای هر ۳۰ گرم ۱۰۰ ریال است و به‌منظور محاسبهٔ هزینهٔ پست، وزن‌نامه‌ها را با تقریب اضافی برحسب تعداد ۳۰ گرم‌های کامل تعیین می‌کنند. قدم‌های کارنمایی را رسم کنید که نشان دهندهٔ تعیین هزینهٔ ارسال یک برنامه هوایی به‌صورت تابعی از متغیر (حقیقی) وزن **WT** باشد. از یکی از توابع **FLR** یا **ROUND** استفاده کنید.

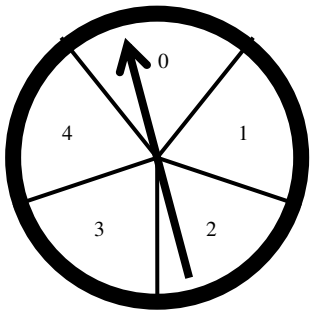
۱۶.روز اول سال، چهارشنبه است. الگوریتمی بنویسید که معین کند روز **N**- اُم سال چه روزی از هفته است. (مثلاً، روز چهاردهم سه شنبه و روز صد و چهل و سوم جمعه است).

۱۷.فرض کنید در **N**- اُمین روز سال هستیم، الگوریتمی بنویسید که تاریخ روز را با دریافت **N** معین کند؟ (مثلاً، اگر در روز شصت و چهارم سال باشیم، تاریخ دوم خرداد ماه است ۳/۲، هدف تعیین شمارهٔ روز و شمارهٔ ماه مربوطه است.)

۱۸.فرض کنید در روز **D**- اُم از ماه شمارهٔ **M**- اُم هستیم، الگوریتمی بنویسید که معین کند در روز چندم سال هستیم؟ (مثلاً، اگر **D** = ۲۱ و **M** = ۸ برای **N** مقدار ۲۳۷ به‌دست می‌آید.)

۱۹.چرخ بازی‌ای که در شکل ۲-۳۰ نشان داده شده به پنج قطاع مساوی تقسیم شده است و این قطاع‌ها در جهت حرکت عقربه‌های ساعت به ترتیب از صفر به بالا شماره‌گذاری شده‌اند. عقربه‌ای وجود دارد که حول محوری که در وسط چرخ سوار شده است می‌چرخد.

فرض کنید **S** شمارهٔ قطاعی است که عقربهٔ در حال سکون روی آن قرار گرفته است. حال با دست، ضربه‌ای در جهت حرکت عقربه‌های ساعت به عقربه می‌زنیم. عقربه از روی **M** قطاع عبور می‌کند و در داخل یک قطاع متوقف می‌شود (یعنی روی خط نمی‌ایستد).



شکل ۲-۳۰

الف- فرمولی بنویسید که با استفاده از یکی از توابع گرد کردن، شمارهٔ قطاع جدید **NEWS** را برحسب شمارهٔ قطاع اصلی در حال سکون **S** و تعداد قطاع‌هایی که عقربه از روی آنها عبور کرده است یعنی **M** ، به‌دست دهد.

ب- فرمول قسمت (آ) این مسئله را برای حالتی که چرخ به **K** قطاع تقسیم و از صفر به بالا شماره‌گذاری شده باشد تعمیم دهید.

ج- قدم‌های کارنمای مورد نیاز برای رسیدن به شمارهٔ قطاع **NEWS** را، که طرز محاسبهٔ آن در قسمت (آ) ذکر شد به نحوی ترسیم کنید که هم برای چرخش در جهت عقربه‌های ساعت و هم برخلاف آن قابل استفاده باشد.

۲۰.اگر قطاع‌های چرخ مسئله قبل را به جای ۰ ، ۱ ، ۲ ، ۳ و ۴ به‌صورت ۱ ، ۲ ، ۳ ، ۴ و۵ شماره‌گذاری کنیم، چه تغییراتی لازم است در حل مسئله داده شود.

۲-۱۲: عبارات رابطه ای مرکب

عملگرهای منطقی

پیشتر با عبارت‌های رابطه‌ای ساده در بخش ۲-۲ آشنا شدید. حال تصمیم داریم با عملگرهای منطقی به عبارات رابطه‌ای مرکب و پیچیده‌تری دست پیدا کنیم، به‌طوری که می‌توان چندین عبارت شرطی ساده را با عملگرهای منطقی (و ، **and**) و (یا ، **or**) ترکیب کرد و عبارات شرطی مرکب را ساخت. در حالت کلی اگر **p** و **q** دو عبارت شرطی با ارزش درستی **T** یا نادرستی **F** باشند، نتیجهٔ ترکیب آنها تحت شرایط مختلف با دو عملگر منطقی **and** و **or** در جداول زیر نشان داده شده است. **r** را به‌عنوان گزارهٔ شرطی معادل ترکیب **p** و **q** در نظر بگیرید:

جدول درستی عملگر *or*

p	*q*	*r*
T	T	T
T	F	T
F	T	T
F	F	F

جدول ۲-۳

جدول درستی عملگر *and*

p	*q*	*r*
T	T	T
T	F	F
F	T	F
F	F	F

جدول ۲-۴

به‌عنوان مثال عبارت زیر مطابق جدول ۲-۴ زمانی درست است که هر دو عبارت شرطی سادهٔ آن درست باشند، در غیر این صورت دارای ارزش نادرست است:

$$(B^2 - 4*A*C > 0) \text{ and } (A \neq 0)$$

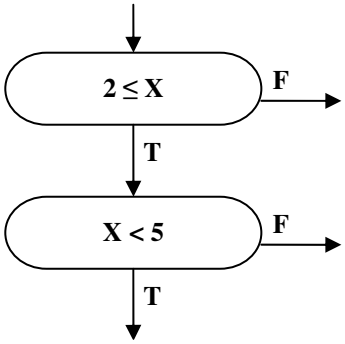
بهتر است در اطراف عبارات شرطی(رابطه‌ای) ساده پرانتز قرار دهید تا از خوانایی عبارت کاسته نشود. اما اگر پرانتز هم نگذارید اشکالی در محاسبات رخ نمی‌دهد. اگر بخواهید جدول ۲-۱ که مربوط به تقدم عملگرها بود را کامل کنید به جدول ۲-۵ خواهید رسید، و چنانکه می بینید در این جدول تقدم عملگرهای رابطه‌ای پس از عملگرهای محاسباتی می‌باشد و عملگرهای منطقی در انتهای جدول قرار دارند.

نشان عملگر	نام عملگر	سطح تقدم
^	به توان رساندن	اول
* , /	ضرب و تقسیم	دوم
٪	باقی مانده تقسیم	
–	منفی کردن (یک تایی)	
+	جمع	سوم
–	تفریق (دوتایی)	
$=, \neq, \geq, >, <, \leq$	عملگرهای رابطه‌ای	چهارم
And , Or	عملگرهای منطقی	پنجم

جدول ۲-۵ : تقدم تمامی عملگرها نسبت به هم در عبارات رابطه ای مرکب

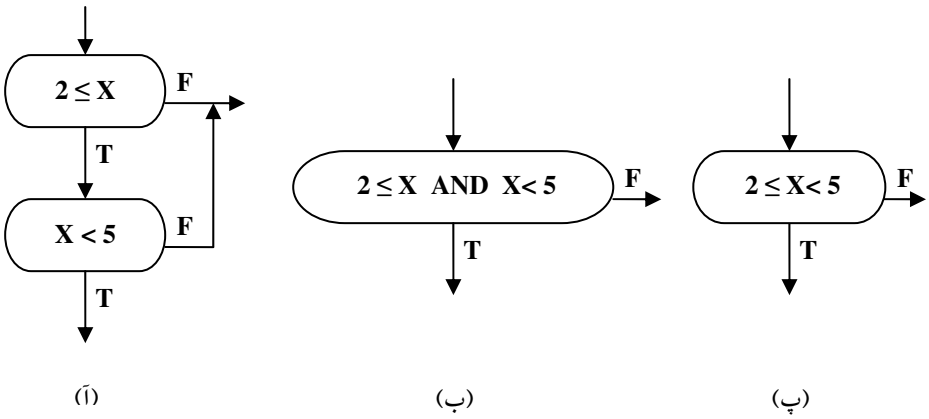
نماد خلاصه شده برای چند تصمیم پشت سرهم

یکبار دیگر به کارنمای شکل ۲-۲۲ مراجعه کنید و دقایقی آن را مدنظر قرار دهید. چنانکه در کارنمای شکل ۲-۲۲ می‌بینید در کارنماها گاهی با یک سری جعبهٔ تصمیم به همراه عبارات رابطه‌ای ساده به شکل زیر مواجه می‌شویم.



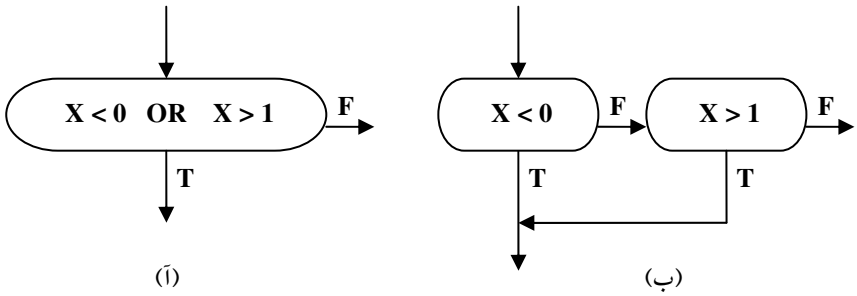
شکل ۲-۳۱ : مجموعه جعبه‌های تصمیم ساده

غالباً به‌منظور خلاصه کردن کارنما و برای مشاهدهٔ تصویر کلی آن، مناسب است که یک جعبهٔ تصمیم با عبارت رابطه‌ای مرکب معادل این سری از جعبه‌ها را جایگزین آنها سازیم. از این رو به‌جای شکل ۲-۳۵ (آ) می‌توان شکل ۲-۳۵ (ب) یا ۲-۳۵ (پ) را قرار داد.



شکل ۲-۳۲ : مجموعه جعبه‌های تصمیم ساده و دو جعبهٔ مرکب معادل آن

از طرف دیگر مواردی هم هست که می‌خواهیم در جهت عکس عمل کنیم و جعبه‌ای را که شامل یک شرط پیچیده‌تر یا مرکب است برداریم و به جای آن مجموعه‌ای از شرایط ساده را قرار دهیم. در این صورت مجموعه شرایط سادهٔ شکل ۲-۳۶ (ب) می‌تواند جانشین شرط مرکب شکل ۲-۳۶ (آ) شود.



شکل ۲-۳۳ : شرط مرکب و مجموعه شرایط سادهٔ معادل آن.

نکاتی در خصوص جعبه‌های تصمیم چندگانه (چند راهه)

پیشتر در بخش ۲-۲ و در شکل ۲-۸ با جعبه‌های تصمیم چندگانه آشنا شدید. همچنین در کارنمای شکل ۲-۲۳ از این نوع جعبهٔ تصمیم استفاده کردیم. در واقع جعبه‌های تصمیم چندگانه راهی دیگر برای ساده‌سازی کارنماها هستند. اما به جهت رابطهٔ این جعبه‌ها با خلاصه‌سازی کارنماها و به جهت اینکه در بخش ۲-۲ سخنی در خصوص ویژگی‌های این جعبه‌ها به میان نیاوردیم در این قسمت به بیان ویژگی‌های این جعبه‌ها خواهیم پرداخت.

۱. در جعبه‌های تصمیم چندگانه هر مسیر خروجی باید به وضوح برچسب‌گذاری شود به نحوی که نشان دهد تحت چه شرط یا شرایطی آن مسیر انتخاب می‌شود.

۲. شرایطی که برروی مسیرهای خروجی ذکر می‌شوند نباید همپوشانی بکنند. به‌عنوان مثال اگر برچسب یک خروجی $x > ۵$ و برچسب دیگری $x < ۸$ باشد و مقدار x برابر ۶ به‌دست آید از کدام یک از این مسیرها باید برویم!؟

۳. همهٔ حالات ممکن نیز باید در نظر گرفته‌شود، چنانکه اگر حالتی پیش آید که با هیچ یک از برچسب‌های خروجی سازگاری نداشته باشد، راهی برای خارج شدن از جعبه نخواهیم داشت.

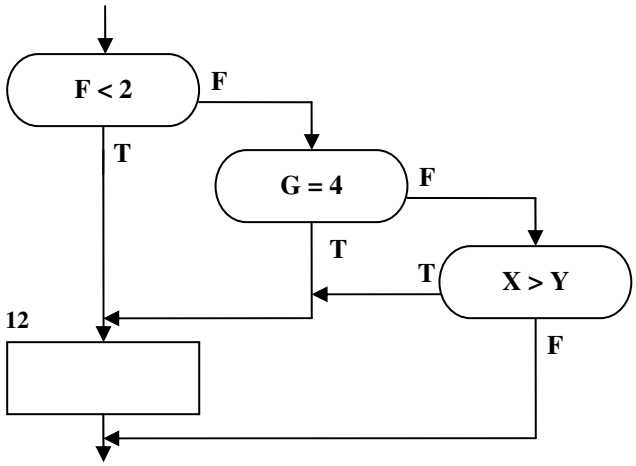
کار در کلاس ۲-۸:

کارنمای شکل ۲-۲۲ که در خصوص شمارش حدود نمرات امتحانی است را با جعبه ی تصمیم چندگانه بازنویسی کنید.

۲-۱۳: تمرین

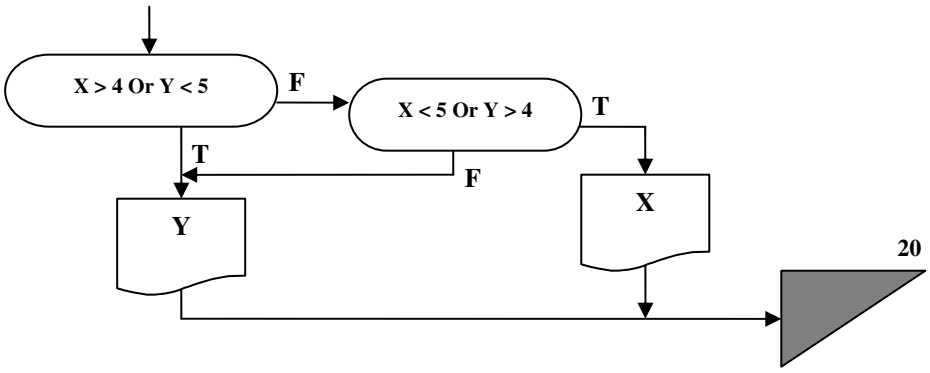
در مسائل ۱ تا ۳ یک قاعده تصمیم‌گیری همراه با کارنمای پیشنهادی معادل آن به شما داده شده است. متغیرهایی که در کارنما آمده‌اند خود بیانگر معانی مربوطه‌اند. تکلیف شما این است که تصمیم بگیرید که آیا هر کارنما معادل منطقی قاعدهٔ تصمیم‌گیری مربوطه هست یا نه، و در صورت منفی بودن جواب، کارنما را چنان اصلاح کنید که منطقاً معادل آن باشد.

۱. اگر **X** بیشتر از **Y** است و یا **F** کمتر از ۲ یا **G** مساوی ۴ است (یا هر دو)، آنگاه جعبهٔ ۱۲ را اجرا کن در غیر این صورت جعبهٔ ۱۲ را اجرا نکن.



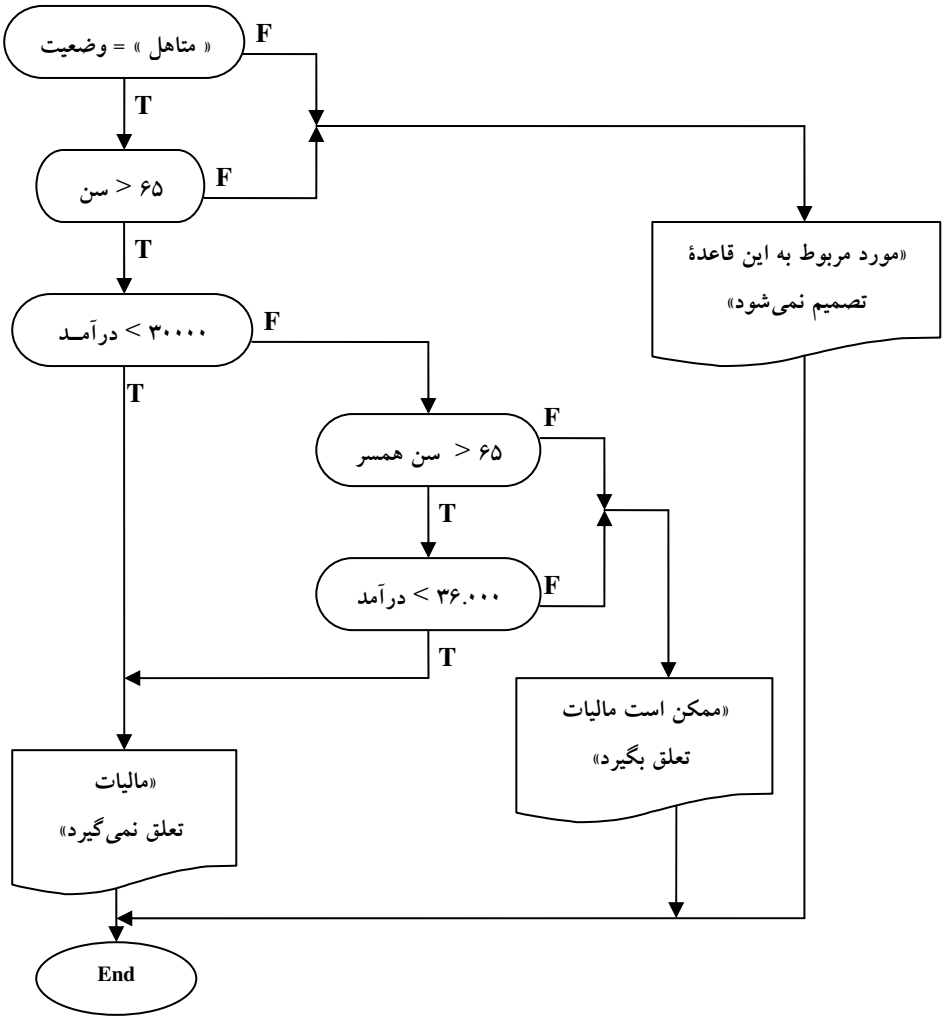
شکل ۲-۳۴ : شکل تمرین ۱

۲. اگر **X** بیشتر از ۴ است در حالی که **Y** کمتر از ۵ است یا **X** کمتر از ۵ است، در حالی که **Y** بزرگتر از ۴ است، آنگاه مقدار **Y** را چاپ کن؛ در غیر این صورت مقدار **X** را چاپ کن و در هر دو صورت، پس از آن به جعبهٔ ۲۰ برو.



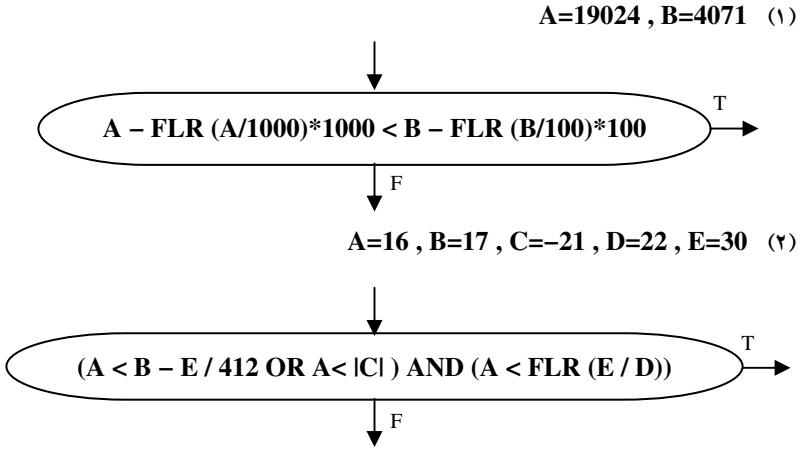
شکل ۲-۳۵ : شکل تمرین ۲

۳. این قاعده مربوط به شخصی است که بیش از ۶۵ سال سن داشته، متاهل باشد و با همسر خود مشترکاً یک اظهارنامه برای مالیات برای درآمد پرمی‌کند. این شخص در صورتی که درآمد سالیانه‌اش کمتر از ۳۰.۰۰۰ تومان باشد یا در صورتی که درآمدش کمتر از ۳۶.۰۰۰ تومان و سن همسرش بیش از ۶۵ سال باشد، مالیاتی نمی‌پردازد .



شکل ۲-۳۹ : کارنمای تمرین ۳

٤. در قسمت‌های (۱) و (۲) زیر، عبارت‌های رابطه‌ای در کارنما داده شده‌اند. وظیفهٔ شما این است که با استفاده از مقادیر داده شده، درستی یا نادرستی این عبارت‌ها را تعیین کنید.



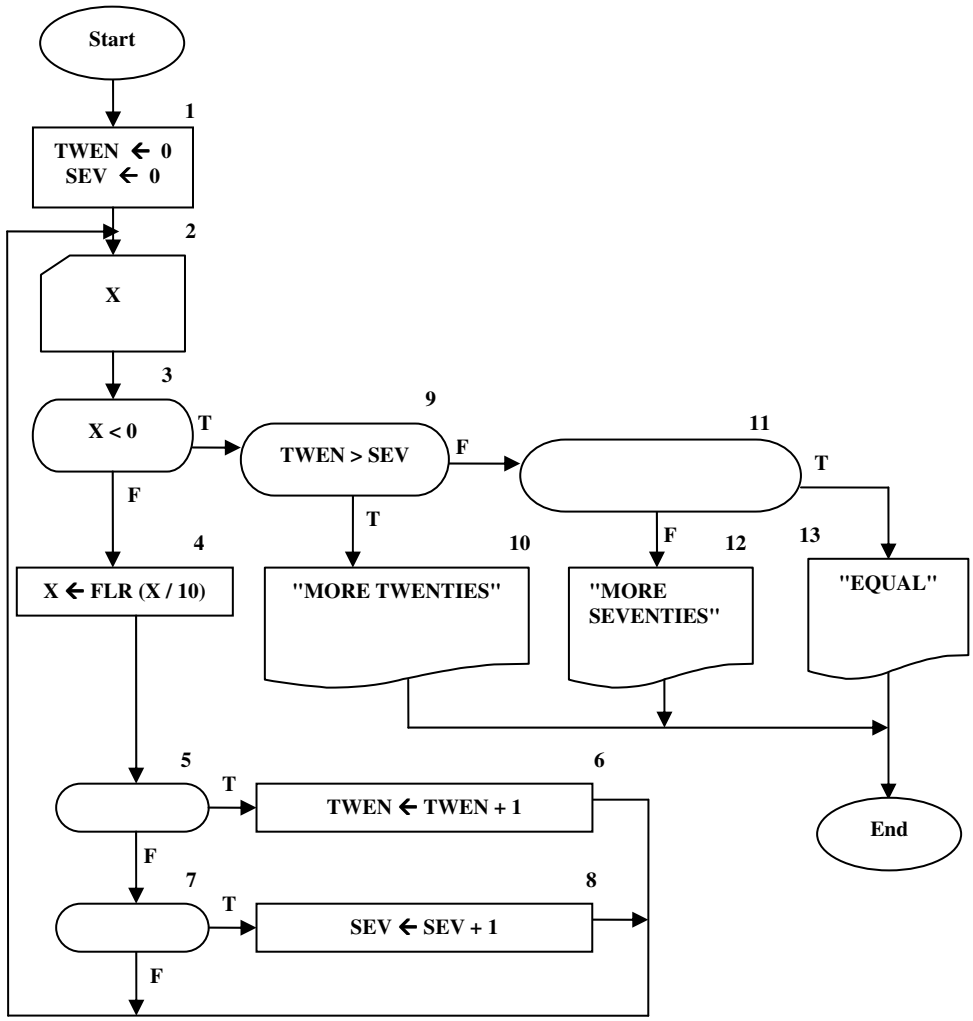
۵. در شکل ۲-۴۷، کارنمای ناتمامی برای یک الگوریتم نشان داده شده است. این الگوریتم به‌منظور خواندن مجموعه‌هایی از اعداد مثبت از ورودی و تعیین اینکه آیا تعداد اعداد واقع در محدودهٔ ۲۰ الی ۲۹ بیشتر از تعداد واقع در محدودهٔ ۷۰ الی ۷۹ هست یا نه، طرح شده است. تعداد اعدادی که باید خوانده شود نامعلوم است ولی تمام شدن داده‌ها به‌وسیلهٔ عددی که شامل یک عدد منفی است، علامت داده می‌شود. در قسمت‌های (آ) و (ب) در زیر، شما باید تعیین کنید که چگونه باید کارنما را به‌طور صحیح تکمیل کرد. به اثر تابع **FLR** در جعبهٔ ۴ توجه داشته باشید.

(آ)

1.	X = 0	5	X > 0	7
2.	X = 20	5	X = 70	7
3.	X = 2	5	X = 7	7
4.	X < 30	5	X < 80	7
5.	هیچ کدام			

(ب)

1.	TWEN < SEV	11
2.	TWEN = SEV	11
3.	70 < 20	11
4.	TWEN = 0	11
5.	هیچ کدام	



شکل ۲- ۴۰ : کارنمای تمرین ۵

۶. کارنمایی رسم کنید که محاسبات لازم برای تعیین ربع صفحه‌ای از محورهای مختصات که نقطهٔ مفروض **P** در آن واقع شده است را انجام دهد و اعداد ۱ ، ۲ ، ۳ یا ۴ را به‌صورت پیامی برای نشان دادن ربع صفحهٔ مربوطه چاپ کند. مختصات **P** ، (**X** و **Y**) مفروض‌اند. تصمیم در مورد نحوهٔ عمل در مواردی که **P** روی یکی از محورها یا هر دوی آنها قرارگرفته باشد با شماست.

۷. کارنمایی برای ترسیم دایره ای به معادلهٔ $(x-a)^2 + (y-b)^2 = r^2$ در صفحهٔ مختصات، رسم کنید. مسلماً ثوابت **a** و **b** و **r** باید از کاربر دریافت شود.

۸. کارنمایی ترسیم کنید که بتواند تشخیص دهد نقطه مفروض **P** نسبت به دایرهٔ فوق چه وضعیتی دارد.

۹. یک استاد ریاضی طالب الگوریتمی است که هرگاه دو عدد مثبت **A** و**B** که ($A < B$)، به آن داده شود، یا یک عدد صحیح **C** طوری پیدا کند که $A \le C^2$ و $C^3 \le B$، یا عدم وجود چنین عددی را گزارش کند. اگر بیش از یک **C** با خواص فوق وجود داشته باشد اهمیتی ندارد که کدام یک انتخاب می‌شود. دو خط مشی ممکن برای این کار در زیر عرضه می‌شود.

الف- مقادیر ... و ۳ و ۲ و ۱ = **C** را امتحان کن تا اینکه یک مقدار قابل قبول پیدا شود یا اطمینان حاصل شود که چنین عددی نمی‌توان یافت.

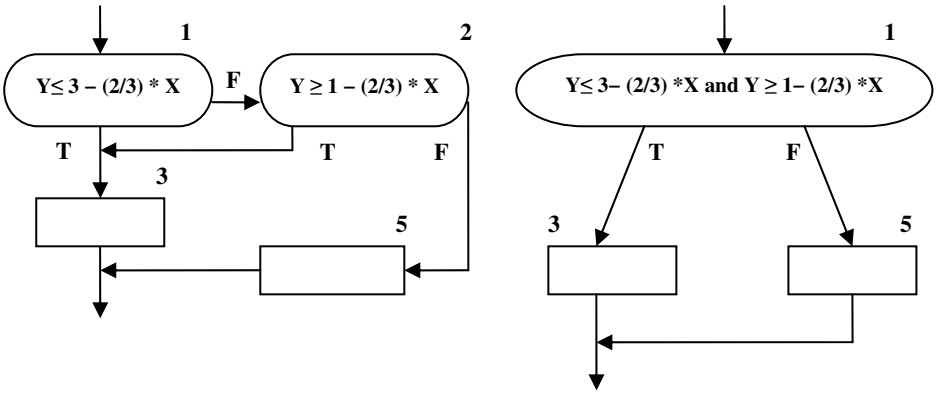
ب- بزرگترین مقدار **C** را که به ازای آن $C^3 \le B$، پیدا کن و ببین آیا این مقدار **C** به اندازهٔ کافی بزرگ هست که $A \le C^2$ یا نه، و بدین‌ترتیب یک مقدار قابل قبول (در صورت وجود) برای **C** محاسبه کن.

وظیفهٔ شما به شرح زیر است:

(۱) برای هریک از دو خط مشی (الف) و (ب) کارنمایی رسم کنید. کارنماهای شما باید با چاپ پیام و مقدار، نتیجه را نشان دهند.

(۲) شما پیشنهاد می‌کنید که استاد فوق از کدام کارنما استفاده کند؟چرا؟

۱۰.آیا دو ترکیب زیر معادل‌اند؟

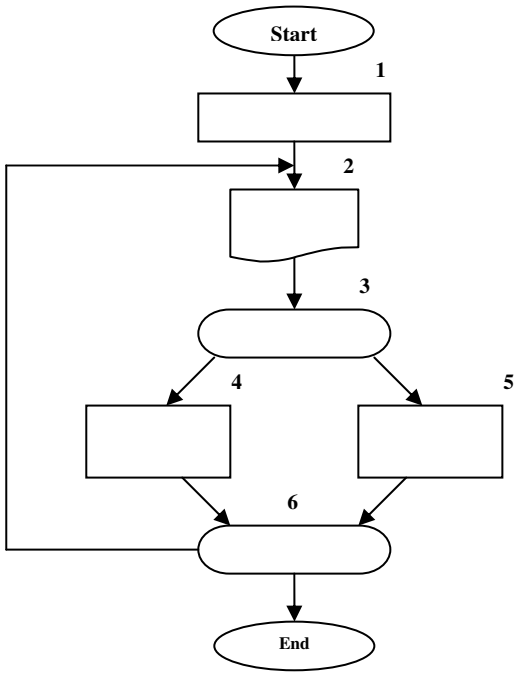


شکل ۲- ۴۱ : کارنماهای تمرین ۱۰

۲-۱۴ : تمرینات تکمیلی

۱. فرض کنید در کنار رودخانه‌ای هستید و دو سطل در اختیار دارید، یکی پنج لیتری و دیگری نه لیتری (سطل‌ها مدرج نیستند). اگر بخواهید چهار لیتر آب داشته باشید، به‌وسیلهٔ سطل پنج لیتری، پنج لیتر از آب سطل نه لیتری بر می‌دارید و در رودخانه خالی می‌کنید.

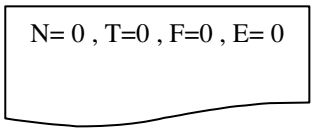
الف- وظیفهٔ شما این است که کارنمایی بسازید که تمام تعداد لیترهایی را که با این قبیل پر و خالی کردن‌ها میتوان جدا کرد، در خروجی بنویسد. فقط به یک متغیر، T، نیاز دارید که حساب مجموع مقدار آب دو سطل را نگه دارد. مقدار T وقتی تغییر می‌کند که آب از رودخانه برداشته شده و یا به آن ریخته شود ولی اگر آب از سطلی به سطل دیگر ریخته شود مقدار T تغییر نمی‌کند. اگر T کوچک‌تر از ۵ باشد سطل نه لیتری را پرمی‌کنید و بدین ترتیب ۹ واحد به T می‌افزایید، در حالی که اگر T بزرگتر یا مساوی ۵ باشد، سطل پنج لیتری را خالی کرده و در نتیجه پنج واحد از T کم می‌کنید. تمام مقادیر T را که محاسبه می‌شود در خروجی بنویسید. فراموش نکنید که مقدار اولیهٔ صفر را به T نسبت دهید. برنامه را وقتی متوقف کنید که مقدار T دوباره صفر شود. ساخت کارنما در شکل ۲-۴۲ نشان داده شده است. تنها کاری که لازم است انجام دهید، پر کردن جعبه‌ها است.



شکل ۲-۴۲ : کارنمای تمرین سطل آب

۲. کارنمای مسئلهٔ ۱ را طوری تغییر دهید که با هر مقدار **T** که چاپ می‌شود، تعداد دفعات پر شدن سطل نه لیتری و همچنین تعداد دفعات خالی شدن سطل پنج لیتری را که برای آن مقدار **T** لازم است، نیز چاپ کند. لازم است از دو متغیر جدید مثل **F** و **E** استفاده کنید. فراموش نکنید که به آنها مقدار اولیه نسبت دهید.

۳. کارنمای مسئله قبل را طوری تغییر دهید که با هر ظرفیتی (**A** و**B**) برای دو سطل کار کند (**A** و**B** اعداد صحیح و برحسب لیتر هستند). سطل **A** لیتری را پر کنید و سطل **B** لیتری را خالی کنید. پیش‌بینی لازم را برای خواندن **A** و **B** به عمل آورید. همچنین از متغیر **N** استفاده کنید که سطرهای خروجی را با شروع از صفر شماره‌گذاری کند به‌طوری که اولین سطر خروجی به‌صورتی که در شکل ۲-۳۳ نشان داده شده است بشود.



شکل ۲-۴۳ : نمونه‌ی خروجی تمرین سه

۴. کارنمای مسئله ۳ را به ازای مقادیر ورودی زیر برای **A** و **B** پیگیری کنید (فقط سطرهای خروجی را بنویسید).

الف- **B=۱۰** و **A= ۷**

ب- **B= ۷** و**A= ۱۰**

ج- **B=۳۶** و**A= ۵۴**

۵. در رابطه با مسئلهٔ ۴، توضیح دهید که چرا روابط زیر همواره صادق است.

الف- $N = F + E$

ب- $T = F \times A - E \times B$

۶. کارنمایی رسم کنید که با دریافت عدد **N** عبارت زیر را محاسبه کند.

$$1^2 + 2^2 + 3^2 + \ldots + N^2$$

آیا می توانید یک فرمول برای عبارت فوق پیدا کنید؟ در این مثال کدام یک از دو روش مکاشفه‌ای و الگوریتمی بهتر هستند؟

۷. کارنمایی رسم کنید که یک عدد در مبنای ۲ شامل ارقام صفر و یک را دریافت کند. سپس این عدد را به مبنای ۱۰ برده و چاپ کند.

۸. کارنمایی رسم کنید که محیط و مساحت یک مثلث را با داشتن مختصات رئوس آن چاپ کند. رئوس مثلث برابر **A** و **B** و **C** هستند و اضلاع آن برابر **a** و **b** و **c** هستند.

طول یک ضلع مثلث $L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

در هر مثلث نصف مجموع اضلاع مثلث را با **p** نشان می‌دهند که برابر نصف محیط مثلث نیز می‌باشد. با توجه به **p** ، مساحت و ارتفاع مثلث برابر است با:

$$p = \frac{a+b+c}{2}$$

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

$$h = \frac{2}{a}\sqrt{p(p-a)(p-b)(p-c)}$$

۹. کارنمایی رسم کنید که شمارهٔ پرسنلی، ساعات کار و دستمزد ساعتی کارکنان یک شرکت را خوانده و حقوق آنها را محاسبه کند. اگر کارمندی بیش از ۴۰ ساعت کار کرده باشد، اضافه کار به او تعلق می‌گیرد. به ازای هر ساعت اضافه‌کاری، یک و نیم برابر دستمزد ساعتی، به‌عنوان اضافه‌کاری پرداخت می‌شود. این الگوریتم را برای حالات زیر حل کنید.

الف- تعداد کارگران مشخص باشد.

ب - تعداد کارگران نامشخص باشد، اما اگر شمارهٔ پرسنلی نامعتبر وارد شد الگوریتم خاتمه یابد.

۱۰.کارنمایی رسم کنید با خواندن عدد **N** ، مجموع **N** جملهٔ اول سری زیر را محاسبه و چاپ کند.

$$S = 1 - 2 + 3 - 4 + \ldots + (-1)^{N+1} N$$

۱۱.کارنمایی رسم کنید که دو عدد **L** و **H** را از ورودی خوانده و کلیهٔ اعداد زوج مضرب ۷ بین این دو عدد را چاپ کند.

۱۲.کارنمایی رسم کنید که دو عدد صحیح را از ورودی خوانده و حاصل‌ضرب آنها را با استفاده از عمل جمع محاسبه می‌کند. (لزوماً این اعداد مثبت نیستند!)

۱۳.مسئلهٔ قبل را برای محاسبهٔ خارج‌قسمت و باقی‌مانده تقسیم به واسطهٔ عمل تفریق و جمع تکرار کنید. این مسئله را یکبار با فرض عدد طبیعی بودن ورودی‌ها یعنی مثبت بودن ورودی‌ها و دیگر بار با فرض دامنه اعداد صحیح یعنی هم مثبت و هم منفی برای ورودی‌ها حل کنید.

۱۴.کارنمایی رسم‌کنید که چه معادلهٔ درجه اول و چه معادلهٔ درجه دوم را به آن بدهند قادر به محاسبهٔ ریشه‌های آن باشد.

راهنمایی: دو راه برای تفکیک معادلات دارید: ۱) صفر بودن ضریب **a** ، ۲)پرسیدن نوع معادله از کاربر.

تصویر بیارنه استراوسروپ مبتکر زبان C++

فصل سوم

مقدّمات زبان C++

اهداف فصل و چکیده مطالب :

۱. آشنایی با تاریخچهٔ زبان C++
۲. چرا زبان C++ را برای یادگیری انتخاب کنیم؟
۳. آشنایی با انواع داده در C++
۴. آشنایی با انوع عملگر در C++
۵. آشنایی با کامپایلر Microsoft Visual C++ 6. 0
۶. بحثی کوتاه در مورد ساختار کلی برنامه در C++
۷. ورودی و خروجی در C++
۸. نوشتن نخستین برنامه‌ها در C++

۳-۱: ویژگی های زبان C++

مقدمه

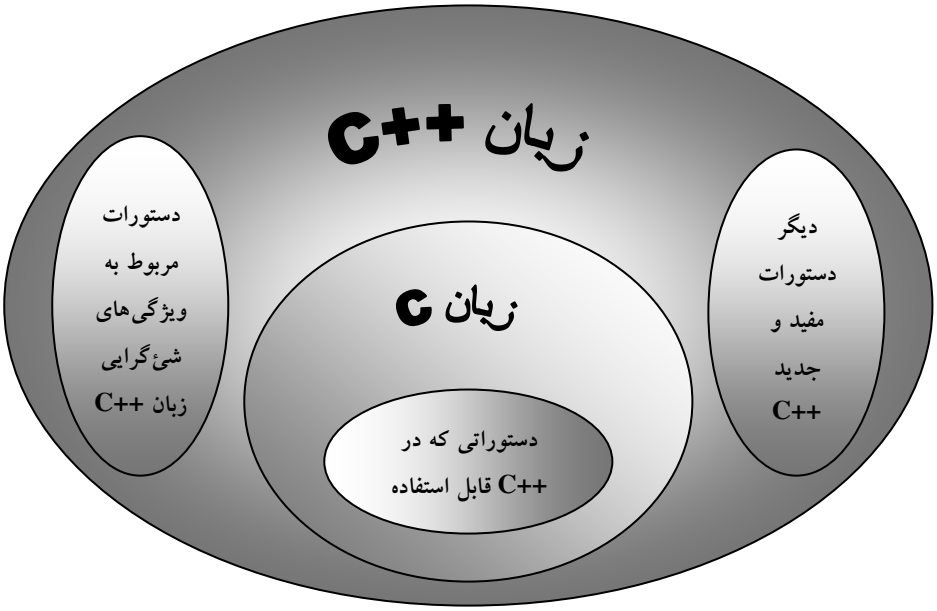
زبان C++ از زبان C مشتق شده است. گفته می‌شود که زبان C++ یک ابر مجموعهٔ[1] زبان C می‌باشد. بدین‌معنا که در زبان C++ می‌توان اکثر دستورات زبان C را به‌کار برد، اما عکس این مطلب صادق نیست. لذا با توجه به اینکه ریشهٔ زبان C++ در C قرار دارد، ابتدا به بررسی تاریخچهٔ زبان C می‌پردازیم.

زبان C در سال ۱۹۷۲ توسط دنیس ریچی (Dennis Ritchie) در آزمایشگاه بل (Bell) طراحی شد. این زبان در ابتدا برروی یک کامپیوتر DEC PDP-11 پیاده‌سازی شد، و تکامل یافتهٔزبان B می‌باشد. علّت نامگذاری C نیز این بوده است که بعد از B طراحی شد. کن تامپسون (Ken Thompson) برخی از ویژگی‌های زبان BCPL را در زبان ابداعی خود یعنی B مدل‌سازی کرد و در سال ۱۹۷۰ از زبان B برای ایجاد نسخه‌های اولیهٔ سیستم‌عامل یونیکس در آزمایشگاه بل بر روی یک کامپیوتر DEC PDP-7 استفاده نمود. اما زبان B نیز خود تکامل یافتهٔزبان دیگری به نام BCPL می‌باشد که طراح آن مارتین ریچاردز (Martin Richards) در سال ۱۹۶۷ این زبان را برای نوشتن نرم افزارهای سیستم عامل و کامپایلر نوشته بود.

زبان‌هایی چون B وBCPL از دسته زبان‌های "بدون نوع" (Type less) هستند. به‌عبارت دیگر هر داده‌ای تنها یک "کلمه" (Word) از حافظه را اشغال می‌کرد و کار با آن داده مثلاً به‌عنوان یک عدد صحیح یا یک عدد اعشاری به عهدهٔ برنامه نویس است. اما از مهمترین ویژگی‌هایی که به زبان C افزوده شد"نظارت بر انواع داده" بود که از دلایل محبوبیت و گسترش آن نیز به‌شمار می‌آید. تا اواخر دههٔ ۱۹۷۰ میلادی زبان C تکامل یافت. زبان C را در ابتدا عمدتاً به‌عنوان زبان توسعه و پیاده‌سازی سیستم‌عامل UNIX می‌شناختند. استفادهٔ گسترده از زبان C توسط انواع گوناگون کامپیوترها، موجب به‌وجود آمدن گونه‌های مختلفی از آن شد. اینگونه‌ها غالباً با یکدیگر ناسازگار بودند و این پدیده برای برنامه‌نویسانی که نیاز بود برنامه‌هایی بنویسند که برروی انواع گوناگون کامپیوترها قابل اجرا باشند، مشکلی جدی بود. از این رو ضرورت وجود یک نسخهٔ استاندارد C احساس می‌شد. در سال ۱۹۸۳ کمیته فنی X3J11 تحت نظر "کمیته استانداردهای ملی امریکا (ANSI)" در خصوص کامپیوترها و پردازش اطلاعات (X3) تشکیل شد تا یک تعریف واضح (نامبهم) و مستقل از سیستم کامپیوتری را برای این زبان ارائه نماید. بالاخره در سال ۱۹۸۹ این استاندارد مورد تصویب قرار گرفت. کمیته ANSI با همکاری سازمان استانداردهای بین‌المللی (ISO) زبان C را در سراسر جهان استاندارد کرد. مستندات استاندارد مشترک در سال ۱۹۹۰ تحت عنوان ANSI/ISO 9899:1990 انتشار یافت. چنانکه پیشتر گفته شد زبان C++ گسترش یافتهٔ زبان C است، که در اوایل دههٔ ۱۹۸۰ توسط بیارنه استراوسروپ (Bjarne Stroustrup) در آزمایشگاه بل ایجاد گردید. امّا مهمترین عناصری که به زبان C اضافه شدند تا زبان C++ ایجاد شود، کلاس‌ها، اشیاء و دریک‌کلام *برنامه‌نویسی‌شئ‌گرا*[2] بود. در نتیجه نام اولیّهٔ زبان C++، "C کلاس‌دار" بود. با وجود این، زبان C++ ویژگی‌های بسیار زیاد دیگری دارد، که از جملهٔ آن می‌توان به روش بهبودیافته‌ای برای ورودی و خروجی (I/O) اشاره کرد. در شکل۳-۱ رابطهٔ بین زبان‌های C و C++ نشان داده شده است. در نوامبر ۱۹۹۷ استاندارد

1. Super set
2. Object Oriented Programming (OOP)

C++ ANSI/ISO عرضه شد. C++ استاندارد، ویژگی‌های جـدیدی به این زبان اضافه کرده است که از میان آنها می‌توان کتابخانهٔ قالب استاندارد (STL) را نام برد.



شکل ۳-۱ : رابطهٔ زبان‌های C و C++

زبان C++ یک زبان سطح میانی است. اصولاً زبان‌های برنامه‌نویسی را به سه دستهٔ زبان‌های سـطح بالا، زبان‌های سطح میانی و زبان‌های سطح پایین تقسیم می‌کنند. علت سطح میانی بودن زبان C++ آن است که هم مانند زبان‌های سطح پایینی مثل اسمبلی دسترسی‌مستقیم به حافظه دارد و می‌توان با مفاهیمی چون بیت، بایت وآدرس‌های حافظه کارکرد، و هم اینکه دستورات این زبان همچون دستورات زبان‌های سطح بالایی مثل پاسکال به زبان محاوره‌ای انسان نزدیک است و خوانایی بالایی دارد.

زبان‌های برنامه‌نویسی از نگاه دیگری بر دو دستهٔ ساخت‌یافته و غیرساخت‌یافته تقسیم می‌شوند. در جدول۳-۱ نام تعدادی زبان ساخت‌یافته و غیرساخت‌یافته را مشاهده می‌کنید.

زبان‌های ساخت‌یافته	زبان‌های غیرساخت‌یافته
پاسکال	فورترن
اِدا	بیسیک
C , C++	کوبول

جدول ۳-۱ : زبان‌های برنامه‌نویسی از نظر ساخت یافتگی

با وجود آنکه C++ به خاطر ویژگیِ شئ‌گرای خود محبوبیت یافته، ولی با توجه به نزدیکی آن به زبان C از برنامه‌نویسی ساخت‌یافته نیز به‌طور کامل پشـتیبانی می‌کند. در بسیاری از کتب آموزش برنامه‌نویسی C++ بدون توجه به سطح معلومات مخاطبین مبتدی از ابتدای کار شروع به گفتن مفاهیمِ اساسی برنامه‌نویسی شئ‌گرا می‌کنند؛ اما چگونه می‌توان به خواننده‌ای که هنوز تفاوت برنامه‌نویسی ساخت‌یافته و غیرساخت‌یافته نیز را نمی‌داند مفاهیم شئ‌گرا را تدریس کرد؟

برنامه‌نویسان در زبان‌های برنامه‌نویسی ابتدایی، که غیرساخت‌یافته به شمار می‌آمدند با مشکلات متعددی در زمینهٔ برنامه‌نویسی برخورد کردند. در دههٔ ۱۹۶۰ میلادی، تولید بسیاری از نرم‌افزارها با مشکل مواجه شد. زمان‌بندی تولید نرم‌افزار به تأخیر می‌افتاد، هزینه‌ها بالا بود و در نتیجه بودجهٔ تولید نرم‌افزار افزایش می‌یافت و نرم‌افزارهای تولیدی نیز از قابلیت اعتماد بالایی برخوردار نبودند. تحقیقاتی که برای برطرف‌کردن این مشکلات به عمل‌آمد، منجر به برنامه‌نویسی ساخت‌یافته شد، و در نتیجه زبان‌های ساخت‌یافته‌ای چون پاسکال، C و فورترن ساخته‌شدند. در این زبان‌ها ویژگی‌هایی افزوده شد مثل پیمانه‌ای بودن (ماجولار یا رویه‌ای بودن) تا برنامه‌نویسی ساخت‌یافته در این زبان‌ها امکانپذیر شد. در برنامه‌نویسی ساخت‌یافته از ایدهٔ *"تفرقه بینداز و حکومت کن"* استفاده شده است. به‌طوری‌که یک برنامه به‌صورت مجموعه‌ای از فعالیت‌ها تصور می‌شود که باید برروی داده‌ها انجام گیرد. بنابراین در این نوع برنامه‌نویسی برای هر فعالیت یک تابع (رویه، ماجول) نوشته می‌شود، و سپس برای اجرای کامل برنامه باید به ترتیب توابع موردنظر را برروی داده‌ها اجرا کرد. به عبارت دیگر برای حل یک مسئله آن را به مسائل کوچکتری تقسیم می‌کنیم و این روند را آنقدر ادامه می‌دهیم تا زیر مسئله‌ها قابل حل گردند. اما زبان‌های ساخت یافته نیز خالی از اشکال نبودند و مشکلات خاص خود را داشتند که به مرور به آنها اشاره خواهیم‌کرد تا به روند تکاملی زبان‌های برنامه‌نویسی واقف گردیم. برای برطرف‌کردن مشکلات مربوط به زبان‌های ساخت‌یافته نیز تحقیقات وسیعی صورت‌گرفت تا آنکه زبان‌های برنامه‌نویسی شئ‌گرا ابداع شدند. برخی از زبان‌هایی که در این زمینه ساخته شدند مثل زبان اسمالتاک (Smalltalk) یک زبان برنامه‌نویسی کاملاً شئ‌گرا (شیءگرای محض) بودند. به عبارت دیگر در اینگونه زبان‌ها جزء به روش شئ‌گرا به روش دیگری نمی‌توان برنامه‌نویسی کرد. آموزش اینگونه زبان‌ها برای افرادی پیشنهاد می‌شود که از قبل با یک زبان ساخت‌یافته آشنایی داشته باشند. اما برای آموزشِ اصولیِ افرادِ مبتدی راهی جزء این نیست که ابتدا با برنامه‌نویسی ساخت‌یافته آشنا شوند و سپس برنامه‌نویسی شئ‌گرا به آنها آموزش داده شود.

از طرفی به علت اینکه اشیاء و کلاس‌هایی که در برنامه‌های شئ‌گرا ساخته می‌شوند، خود به‌عنوان بخشی از برنامه‌نویسی ساخت‌یافته به شمار می‌روند در جلد نخست این مجموعه تنها به برنامه‌نویسی ساخت‌یافته پرداخته شده است و در جلد دوم به سراغ سیستم‌های شئ‌گرا خواهیم رفت. لذا در این فصل بیش از این در خصوص ویژگی‌های شئ‌گراییِ زبان C++ صحبت نخواهیم کرد. در فـصل‌های آتی نیز تنها در خصوص ویژگی‌های برنامه‌نویسی ساخت‌یافته و تفاوت‌های آن با برنامه‌نویسی غیرساخت‌یافته به تفصیل بحث خواهیم کرد. لذا در حال حاضر نگران چگونگی ایجاد برنامهٔ ساخت‌یافته و عدم آشنایی با ویژگی‌های آن نباشید. شایان ذکر است که مطالب فصول برنامه‌نویسی با این فرض تنظیم شده است که خواننده هیچگونه آشنایی قبلی با هیچ یک از زبان‌های برنامه‌نویسی ندارند، لذا سعی بر آن داریم که مثال‌های برنامه‌نویسی از هیچگونه پیچیدگیِ خاصی برخوردار نباشد. و این بدان علت است که از کیفیت آموزش کاسته

نشود. امّا برای حفظ ارزش علمی کتاب برخی مثال ها و تمرینات پیچیده و ارزشمند را برای تکمیل بحث در انتهای فصول برنامه‌نویسی تحت عنوان "مورد مطالعاتی" آورده‌ایم. امّا یک سؤال مهم همواره برای افرادی که به دنبال یادگیری یک زبان شئ‌گرا هستند وجود دارد و آن اینکه:

از بین دو زبان شئ‌گرای مطرح یعنی جاوا و C++ کدام یک ارزش بیشتری برای یادگیری دارد؟ و یا در اولویت قرار دارد؟

در جواب این دوستان باید گفت: از میان زبان‌های برنامه‌نویسیِ شئ‌گرا، زبان C++ پر استفاده‌ترین زبان شئ‌گرا محسوب می‌شوند. چرا که زبانی مثل جاوا فاقد بعضی از ویژگی‌های سطح پایین C++ مثل اشاره‌گرها است که از قدرت آن می‌کاهد و انعطاف‌پذیری آن نسبت به C++ کمتر است. امّا با توجه به آنکه زبان‌هایی چون جاوا، C# ، J# و امثال آن روزبه‌روز بر کاربردشان افزوده می‌شود در جلد دوم این مجموعه صفحاتی را به زبان C# اختصاص داده‌ایم، امّا در این کتاب تنها به زبان‌های C++ وC خواهیم پرداخت. خوشبختانه برای یادگیری زبان C++ نیازی نیست که ابتدا زبان C را یاد بگیرید و برخی دستورات C تنها به این جهت مطرح شده است که ارزش دستورات آسان و کاربردی زبان C++ روشن گردد. اما دانستن زبان C++ در فراگیری هر چه بهتر زبان‌های شیءگرای جاوا و C# بسیار مؤثر است.

به هرحال آنچه که در این کتاب در خصوص زبان C++ مطرح شده مطابق با زبان C++ استاندارد است. گرچه برخی شرکت‌های تولیدکنندهٔ کامپایلر C++ چون مایکروسافت و بورلند ویژگی‌هایی را به دلخواه خود به این زبان اضافه کرده‌اند امّا هدف ما بیشتر آشنایی شما با قوانین استاندارد C++ است تا بتوانید در کامپایلرهای این زبان در سیستم‌عامل لینوکس نیز به راحتی و تغییراتی جزئی برنامه‌نویسی کنید. برای این منظور یک ضمیمه در خصوص نحوهٔ برنامه‌نویسی به زبان C++ در سیستم عامل لینوکس در انتهای کتاب آورده شده است.

برای اجرای کدهای این کتاب نیازمند یک کامپایلر قابل اعتماد و دارای ویژگی‌های روز دنیا مثل امکانات دیداری(ویژوالی) هستید. چرا که در جلد سوم تصمیم داریم برنامه‌نویسی رویدادگرا را نیز آموزش دهیم لذا بهتر است با کامپایلری که دارای این ویژگی‌ها باشد کار را شروع کنید. برای این منظور در این کتاب تمامی برنامه‌ها با توجه به ویژگی‌های کامپایلر Microsoft Visual Studio 6.0 نوشته شده است. لذا ابتدا با مراجعه به ضمیمهٔ شمارهٔ یک مطابق دستورالعمل ارائه شده این کامپایلر را به واسطهٔ لوح‌های فشردهٔ ۲ و۳ و ۴ از مجموعه لوح‌های فشرده همراه کتاب[۱] نصب کنید. شاید بسیاری بر این اعتقاد باشند که بهتر بود کامپایلر جدیدتری مثل کامپایلر ۲۰۰۵ شرکت مایکروسافت را برای آموزش انتخاب می‌کردیم. اما به دو دلیل عمده از این کار خودداری نمودیم. اول آن که شرکت مایکروسافت ویژگی‌هایی به کامپایلرهای .NET، خود افزوده که از زبان C++ استاندارد کمی فاصله گرفته است. دوم آن که برنامه هایی که در این کامپایلرها نوشته می‌شود برای اجرای، نیازمند وجود پلت‌فرم .NET، بر روی کامپیوتر مقصد هستند که بدون شک بر روی ویندوزهای قدیمی مثل ویندوز ۹۸ این پلت فرم قابل نصب نیست. لذا از آن جا که امکان دارد شما روزی مجبور به کار کردن با کامپیوترهای قدیمی ارزان قیمت و به طبع با سیستم عامل های قدیمی شوید، مثل زمانی که بخواهید با یک ۸۰۸۶ یک خط تولید ساده را کنترل کنید، لزوم فراگیری کار با VS98 به خوبی روشن می‌شود.

---

۱. لوح فشردهٔ شمارهٔ ۱ همراه کتاب است. اما لوح‌های فشردهٔ ۲ تا ۴ به‌صورت اختیاری می‌باشد که شما می‌توانید جهت تهیهٔ آنها به وب سایت کتاب به آدرس www.programmingbook.4t.com مراجعه فرمایید.

۳-۲: کار با انواع داده ها در C++

انواع داده در C++

چنانکه در فصل گذشته دیدید در الگوریتم‌ها هرجا لازم بود متغیری تعریف می‌کردیم و مقادیر صحیح، اعشاری، رشته‌ای و... را می‌توانستیم به آن نسبت دهیم. امّا برای کامپیوترهای واقعی چنین امری ممکن نیست. یعنی باید نوع متغیر از نظر اینکه چگونه اطلاعاتی را می‌تواند در خود ذخیره‌کند مشخص باشد. لذا هنگّامی که می‌خواهیم متغیری را تعریف کنیم باید قبل از نام آن متغیر، نوع متغیر را مشخص سازیم. در C++ انواع متغیرهایی که در جدول ۳-۲ آمده امکانپذیر است. به توضیحاتی که در خصوص جدول آمده خوب توجه کنید.

نوع داده	اسامی دیگر (انواع مشابه)	اندازه به بیت	بازهٔ قابل قبول
int	signed , signed int	۳۲ یا ۱۶	بستگی به نوع سیستم عامل دارد
unsigned int	unsigned	۳۲ یا ۱۶	بستگی به نوع سیستم عامل دارد
__int8	char , unsigned char	۸	از ۱۲۸- تا ۱۲۷
__int16	Short , short int , signed short int	۱۶	از ۳۲.۷۶۸- تا ۳۲.۷۶۷
__int32	signed , signed int	۳۲	از ۲.۱۴۷.۴۸۳.۶۴۸- تا ۲.۱۴۷.۴۸۳.۶۴۷
__int64	ندارد	۶۴	از ۹.۲۲۳.۳۷۲.۰۳۶.۸۵۴.۷۷۵.۸۰۸- تا ۹.۲۲۳.۳۷۲.۰۳۶.۸۵۴.۷۷۵.۸۰۷
char	signed char	۸	از ۱۲۸- تا ۱۲۷
unsigned char	ندارد	۸	از ۰ تا ۲۵۵
short	short int , signed short int	۱۶	از ۳۲.۷۶۸- تا ۳۲.۷۶۷
unsigned short	unsigned short int	۱۶	از ۰ تا ۶۵.۵۳۵
long	Long int , signed long int	۳۲	از ۲.۱۴۷.۴۸۳.۶۴۸- تا ۲.۱۴۷.۴۸۳.۶۴۷
unsigned long	unsigned long int	۳۲	از ۰ تا ۴.۲۹۴.۹۶۷.۲۹۵
float	(دقت تا۷رقم اعشار) ندارد	۳۲	از $۳/۴\times۱۰^{-۳۸}$ تا $۳/۴\times۱۰^{۳۸}$
double	ندارد(دقت‌تا۱۵رقم اعشار)	۶۴	از $۱/۷\times۱۰^{-۳۰۸}$ تا $۱/۷\times۱۰^{۳۰۸}$
long double	ندارد(دقت‌تا۱۹رقم اعشار)	۸۰	از $۱/۲\times۱۰^{-۴۹۳۲}$ تا $۱/۲\times۱۰^{۴۹۳۲}$
bool	ندارد	۸	صفر یا یک

جدول ۳-۲ انواع داده‌ها و مقادیر قابل قبول آنها

در C++ استاندارد، چهار نوع دادهٔ اصلی وجود دارد که عبارتند از: **int** ، **char** ، **float** ، **double**. در ادامه به بررسی انواع داده‌های اصلی و معادل‌های آنها مطابق با جدول فوق می‌پردازیم.

انواع داده‌ی اصلی در C++ استاندارد

۱. نوع دادهٔ صحیح: از آنجایی‌که می‌توان رقم صفر را به‌صورت عدم وجود جریان برق در یک خانهٔ حافظه و رقم یک را به‌صورت وجود جریان برق در یک خانه از حافظه نشان داد، در کامپیوتر جهت ذخیرهٔ عدد آن را به مبنای دو می‌برند تا به یک رشته شامل صفر و یک تبدیل شود. در این صورت می‌توان نمایش دودویی عدد را که به‌صورت رشته‌ای از صفرها و یک‌ها است به راحتی ذخیره ساخت. چنانکه دیدید در الگوریتم‌نویسی در بیشتر موارد با اعداد صحیح سروکار داریم. به‌عنوان مثال شمارنده‌های حلقه‌های تکرار و یا متغیرهایی که معرف تعداد یک چیز هستند مثل تعداد دانش‌آموزان یک کلاس و هزاران نمونهٔ دیگر از کاربردهای اعداد صحیح در الگوریتم‌نویسی به شمار می‌آیند. برای کارکردن با اعداد صحیح در C++ باید متغیر مورد نظر را از نوع **int** تعریف کنید.

چنانکه در جدول ۳-۲ نیز بیان شده طول نوع **int** بستگی به سیستم‌عامل دارد. در سیستم‌های ۳۲ بیتی مثل ویندوز ۹۸ و بعد از آن طول این نوع برابر ۳۲ بیت و دارای ظرفیتی از ۲.۱۴۷.۴۸۳.۶۴۸- تا ۲.۱۴۷.۴۸۳.۶۴۷ می‌باشد. اما در سیستم‌های ۱۶ بیتی مثل MS-DOS طول این نوع ۱۶ بیت بوده و محدوده‌ای از ۳۲.۷۶۸- تا ۳۲.۷۶۷ را شامل می‌شود. همان‌طور که می‌بینید اعداد صحیح در کامپیوتر مانند آنچه که در ریاضیات فرض‌می‌کردیم [که شامل هیچ حد و مرزی نبوده و از $-\infty$ تا $+\infty$ را شامل می‌شود.) نیستند. چرا که خانه‌های حافظهٔ کامپیوتر محدود بوده و با تخصیص ۴ بایت برای نوع **int** این بازه از اعداد را می‌توان در این نوع ذخیره کرد.

۲. نوع دادهٔ کاراکتری : مسلماً همیشه با ذخیره کردن اعداد روبرو نیستیم و گاهی اوقات پیش می‌آید که بخواهیم مثلاً حروف الفبای لاتین مثل حروف **a , b , c** و... یا علائمی مثل **$, [,) , ? ,** و... را در کامپیوتر ذخیره کنیم. امّا با توجه به آنکه در خانه‌های حافظهٔ کامپیوتر فقط می‌توان صفر و یک ذخیره کرد چه راه‌حلی برای ذخیرهٔ این علائم و حروف به نظر شما می‌رسد؟

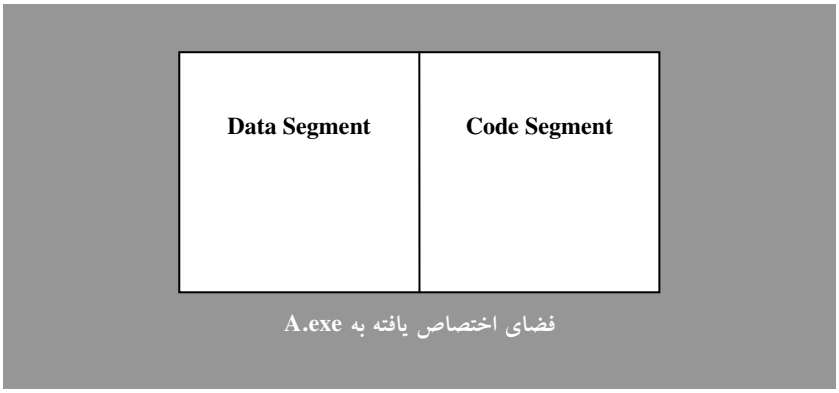
در ابتدا مشاهده شد که در یک بایت می‌توان از صفر تا ۲۵۵ را در حالت بدون علامت، و یا از ۱۲۸- تا ۱۲۷ را در حالت با علامت، ذخیره کرد. یعنی ۲۵۶ عدد مختلف، که می‌توانست به‌عنوان نمایندهٔ ۲۵۶ نماد مختلف در نظر گرفته شود. لذا در جدولی با نام جدول **ASCIICode** برای هر یک از حروف و علائمی که روی صفحه کلید موجود بودند و همچنین برخی علائمی که روی صفحه کلید موجود نیستند ولی برای اموری مثل کادربندی و امثال آن مورداستفاده قرار می‌گرفت، یک عدد بین ۱۲۸- تا ۱۲۷ را در نظر گرفتند، مثلاً برای حرف **A** ، عدد ۶۵ را در نظر گرفتند. حال اگر در یک بایت عدد ۶۵ را ذخیره می‌کردند، هنگام نمایش آن بایت به جای عدد ۶۵ حرف **A** نمایش داده می‌شد. امّا در محاسبات و ذخیره‌سازی عدد ۶۵ به جای حرف **A** مورد استفاده قرار می‌گرفت. برای ذخیرهٔ کاراکترها باید متغیری را از نوع **char** تعریف کنید.

کار در کلاس ۳-۱ :

*امّا یک مشکل هنوز حل نشده باقی ماند! کامپیوتر از کجا بفهمد که عدد ۶۵ ذخیره شده در یک بایت مربوط به کاراکتر **'A'** است و یا خود عدد ۶۵ به‌صورت **int** است؟ در کلاس خود در این باره بحث کنید.*

در جواب سؤال مطرح شده در کار در کلاس ۳-۱ می‌توان گفت:

در زمانی که می‌خواهیم یک فایل اجرایی (exe) را برروی کامپیوتر اجرا کنیم، سیستم عامل ابتدا مقداری از فضای آزاد حافظه (RAM) را به میزان مورد نیاز برای اجرای برنامه به آن اختصاص می‌دهد، مثلاً فرض کنید از کل فضای RAM که در شکل ۳-۲ نشان داده شده فضای مربع سفید رنگ به برنامه A.exe اختصاص داده شده است.



شکل ۳-۲ : تقسیم بندی فضای حافظه

حال از فضای اختصاص یافته به برنامه یک سگمنت جهت داده‌ها و یک سگمنت جهت کد برنامه اختصاص داده می‌شود. مقصود از سگمنت داده‌ها فضایی از RAM است که متغیرها و داده‌های مربوط به برنامه‌یA.exe را درخود ذخیره می‌سازد و مقصود از سگمنت کد آن قسمتی از RAM است که دستورات اجرایی برنامه در آن قرار می‌گیرد مثل دستورات انتساب، دستورات شرطی و... . البته سگمنت‌های دیگری هم می‌تواند وجود داشته باشد که از بحث ما خارج است. سگمنت کد فعلاً مدنظر ما قرار ندارد، امّا سگمنت داده‌ها به نوبهٔ خود به قسمت‌های مختلفی تقسیم‌می‌شود؛ مثلاً یک بخش آن جهت داده‌های نوع‌صحیح (int) و بخش دیگری جهت داده‌های نوع کاراکتر(char) و ... تقسیم می‌شود. هنگامی که از کامپیوتر می‌خواهیم به متغیری از نوع char مراجعه کند [که در آن عدد ۶۵ را به‌عنوان نماد حرف A از قبل ذخیره کرده‌ایم] و محتویات آن را نمایش دهد، اساساً محل ذخیرهٔ کاراکترها با انواع دیگری مثل نوع int به کلی مجزا است، بنابراین کامپیوتر به راحتی تشخیص می‌دهد که این متغیر از نوع کاراکتر است نه نوع صحیح و عدد ۶۵ ذخیره شده در آن معرف کاراکتر A می‌باشد و نه خود عدد ۶۵ . لذا در هنگام نمایشِ این محل از حافظه، حرف A نمایش داده می‌شود و نه عدد ۶۵ . از طرفی عدد ۶۵ که در متغیری از نوع صحیح ذخیره می‌گردد در ۴ بایت ذخیره می‌شود ولی همین عدد وقتی در متغیری از نوع کاراکتر ذخیره شود تنها در ۱ بایت ذخیره می‌گردد. این نوع تقسیم‌بندی سگمنت داده‌ها در مباحثی مثل آرایه‌ها بسیار حائز اهمیت می‌باشد که در فصل آرایه‌ها و رشته‌ها به آن اشاره خواهیم کرد.

اما این تقسیم‌بندی ذاتاً توسط سخت افزار صورت نمی‌گیرد، بلکه این کامپایلر زبان C++ است که حافظه‌ی تحت اختیار برنامه را این چنین تقسیم بندی می‌کند. چنان که اگر شما خود کدهای اسمبلی برنامه‌ای را بنویسید می‌توانید سگمنت داده را بدین صورت تقسیم بندی نکنید و تمامی داده‌ها را بدون نظم در حافظه ذخیره کنید در آن حالت این برنامه نویس است که باید نوع داده و محتویات آن را تفسیر کند.

اما در برخی سیستم‌های کامپیوتری به جای آنکه در نوع **char** از ۱۲۸- تا ۱۲۷ را ذخیره کنند از صفر تا ۲۵۵ را ذخیره می‌سازند که در عمل تفاوت چندانی با حالت قبل ندارد. متأسفانه کاراکترهای بین ۱۲۸ تا ۲۵۵ استاندارد نبوده و مشکلات زمانی رخ می‌دهد که برنامه‌ها بخواهند بین سیستم‌های مختلف کامپیوتری جابه‌جا شوند. جدول کدهای اسکی استاندارد در پیوست ۲ آمده است. با مراجعه به این جدول، معادل اسکی کد هر یک از حروف لاتین، علائم و کلیدهای ترکیبی روی صفحه کلید و برخی کاراکترهای گرافیکی را مشاهده کنید. امّا حتماً با مشاهدهٔ این جدول می‌پرسید حروف فارسی و دیگر زبان‌های غیرانگلیسی چگونه در کامپیوتر ذخیره می‌شوند؟ پس از گسترش کامپیوتر در سراسر جهان و احساس نیاز به ذخیره حروف دیگر زبان‌های طبیعی، به این نتیجه رسیدند که به جای استاندارد **ASCI I Code** از استاندارد **UNI Code** استفاده کنند که در آن از دو بایت برای ذخیره سازی حروف زبان‌های مختلف و علائم گوناگون از جمله علائم ریاضی و... استفاده شد. لذا در **C++** استاندارد، نوع وسیع تری نسبت به نوع **char** به نام **wchar_t** در نظر گرفته شده که برای نوشتن برنامه‌هایی با توزیع جهانی در نظر گرفته شده است. امّا از آنجایی که فعلاً جدول اسکی کد نیاز ما را برای برنامه‌نویسی‌های مقدماتی برطرف می‌سازد و همچنین در حالت کنسول که ما در آن برنامه‌نویسی می‌کنیم از **UNI Code** پشتیبانی نمی‌شود امکان استفاده از این قابلیت **wchar_t** فعلاً برای ما وجود ندارد، و لذا مطالعهٔ بیشتر در خصوص جدول **UNI Code** و نوع **wchar_t** را به خوانندگان عزیز محول می‌سازیم، البته در برنامه‌های کاربردی ویندوز امکان استفاده از **wchar_t** هست.

۳. انواع دادهٔ اعشاری : نوع دادهٔ **float** جهت ذخیره‌سازی اعداد اعشاری ممیز شناور تا هفت رقم اعشار دقت مورد استفاده قرار می‌گیرد. اما مفهوم هفت رقم اعشار دقت چیست!؟ آیا به معنای هفت رقم پس از ممیز اعشار است !؟ مسلماً خیر. چراکه اگر چنین تعبیری صحیح بود هیچگاه نمی‌توانستیم عددی مثل $۳/۴ \times ۱۰^{-۳۸}$ را در متغیری از این نوع ذخیره کنیم، زیرا تعداد ارقام اعشار این عدد ۳۹ رقم است. قبل از این که به رفع این ابهام بپردازیم باید دو مقولهٔ نمایش اعداد به‌صورت نماد علمی و ممیز شناور را بررسی کنیم.

هنگامی که از سیستم ممیز شناور برای نمایش اعداد استفاده می‌کنیم بدین مفهوم است که تعداد ارقام معنی‌دار عدد ثابت است. به‌عنوان مثال اعداد زیر هر کدام دارای چهار رقم معنی دار می‌باشند:

$$۸۴/۵۶ \times ۱۰^{۲} \quad ، \quad ۰/۰۰۰۸۵۴۰ \quad ، \quad ۵/۶۲۳ \times ۱۰^{-۳} \quad ، \quad ۰/۲۰۰۰ \quad ، \quad -۰/۰۳$$

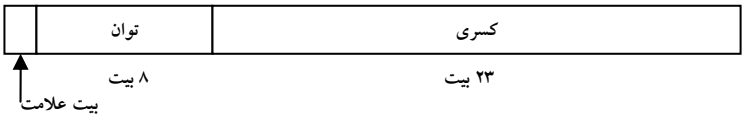
اما مفهوم هفت رقم معنی‌دار چیست؟ به تعداد هفت رقم، به غیر از صفرهای موجود قبل از اولین رقم ناصفر، هفت رقم معنی‌دار گفته می‌شود. به‌عنوان مثال عددی مثل ۰/۵۶۲۳۷۸۹ دارای هفت رقم با معنی است. همچنین عدد ۰/۰۰۰۵۲ نیز دارای هفت رقم معنی‌دار است، زیرا می‌توان این عدد را چنین نوشت $۰/۵۲۰۰۰۰۰ \times ۱۰^{-۳}$ ، و یا عدد ۸۵/۰۰۲۳ نیز چنین است. مسلماً متوجه شده‌اید که صفرهای سمت راست آخرین رقم ناصفر اعشاری نیز جزء رقم‌های معنی‌دار شمارش می‌شوند. اما عددی مثل ۲۳۱/۹۸۷۵۶۴ حداقل دارای ۹ رقم معنی می‌باشد. آیا می‌تواند۱۰ رقم معنی‌دار نیز داشته باشد؟

اما بهترین روش برای نمایش اعداد ممیز شناور استفاده از نماد علمی است. در روش نماد علمی عدد به دو بخش تقسیم می‌شود. یک قسمت کسری یا مانتیس(شامل ارقام با معنی) و یک قسمت نیز حاوی عدد صحیحی است که توانی از ۱۰ می‌باشد. قسمت کسری عددی است بین صفر و ده، و قسمت توان یک عدد صحیح است. با نمایش عدد به‌صورت

نماد علمی به‌راحتی می‌توان تعداد ارقام معنی دار عدد را شمرد. به‌عنوان مثال اعداد زیر به‌صورت نماد علمی نمایش داده شده‌اند:

$$-۲/۳۶۵۴۷۱ \times ۱۰^{-۲۴} \quad ، \quad ۹/۲۳۵۰۰۰ \times ۱۰^{۳۲}$$

حال که با سیستم نمـایش اعداد به‌صورت ممیز شناور در قالب نماد علمی آشنا شدید به راحتی می‌توانید بفهمید که اعداد ممیز شناور چگونه در ۴ بایت یک متغیر **float** ذخیره می شود، هر متغیر **float** دارای ساختاری مشابه ساختار زیر است:



*شکل ۳-۳ : شکل یک متغیر **float***

مسلماً با توجه به شکل ۳-۳ می‌توانید حدث بزنید، که یک عدد اعشاری ممیز شناور چگونه در یک متغیر از نوع **float** ذخیره می‌گردد. حال به نظر شما آیا عددی مثل ۰/۳۲۵۶۴۱۷۸ که ارقام اعشاری کمتری نسبت به حد پایین نوع **float** یعنی $۳/۴ \times ۱۰^{-۳۸}$ دارد را می‌توان در متغیری از نوع **float** ذخیره کرد.

۴. نوع دادهٔ double : جهت ذخیره‌سازی اعداد اعشاری بزرگتری که در نوع **float** جای نمی‌گیرند مورد استفاده قرار می‌گیرد. نوع دادهٔ **double** که به ممیز شناور دقت مضاعف نیز موسوم است، اعداد اعشاری تا ۱۵ رقم معنی‌دار را می‌تواند در خود ذخیره کند.

۵. نوع دادهٔ منطقی : بسیاری از اوقات نیازمند ذخیره‌سازی دو مقدار یک و صفر به ترتیب به‌عنوان **true** و **false** هستیم. برای این منظور نوع دادهٔ **bool** در نظرگرفته‌شده است. در عمل برای ذخیرهٔ نوع دادهٔ **bool** به یک بیت از حافظه نیاز است امّا در عمل، بسته به نوع کامپایلر، این نوع داده را به‌صورت اعداد صحیح در چهار بایت و یا در یک بایت ذخیره می‌کند. زیرا به اعداد صحیح به سرعت می‌توان دسترسی پیدا کرد، حال آن که یک بیت از یک عدد صحیح باید استخراج شود که نیازمند صرف زمان بیشتری است. بعداً خواهید دید که از نوع دادهٔ **bool** بیشتر برای ذخیره‌سازی نتایج عملیات منطقی(مقایسه‌ای یا شرطی) استفاده می‌شود و به عبارت دیگر می‌توان نتایج گزاره‌ها را در نوع دادهٔ منطقی ذخیره‌سازی کرد. توجه داشته باشید که این نوع جزء انواع استاندارد **C++** نیست و شرکت‌های تولیدکنندهٔ کامپایلر **C++** برحسب سلیقه و نیاز کاربران، این نوع را در کامپایلرهای خود ممکن است افزوده باشند ولی در هر صورت در کامپایلر مورد بحث این کتاب یعنی **VC6** این نوع به خوبی پشتیبانی می‌شود.

در ادامه به بررسی انواع داده فرعی در **C++** استاندارد و برخی از انواع که مختص کامپایلرهای شرکت مایکروسافت مثل **VC6** هستند می‌پردازیم.

انواع دادهٔ فرعی در C++ استاندارد

با استفاده از کلماتی مثل **signed** (با علامت)، **unsigned** (بدون علامت)، **long** و **short** می‌توان انواع جدیدی را ایجاد کرد. این کلمات را می‌توان با انواع داده‌های اصلی به کار برد. مثلاً نوع **unsigned int** با حذف بیت علامت از نوع دادهٔ **int** قادر است بازهٔ بزرگتری را در محدودهٔ اعداد مثبت یعنی بین صفر تا ۴.۲۹۴.۹۶۷.۲۹۵ را در چهار بایت ذخیره سازد. امّا از آنجایی که بسیاری از انواع داده مثل **int** از نوع علامت‌دار هستند به‌کاربردن کلمهٔ **signed** در جلوی این انواع داده‌ای بی‌معناست. در کامپیوترهایی که در نوع **char** اعداد صفر تا ۲۵۵ را ذخیره می‌سازند به‌کارگیری کلمهٔ **signed** در جلوی نوع **char** می‌تواند بازهٔ قابل قبول این نوع را به ۱۲۸- تا ۱۲۷ تغییر دهد.

چنانکه پیشتر گفته شد طول نوع دادهٔ **int** بسته به ۱۶ یا ۳۲ بیتی بودن سیستم، متغیر است. به‌کارگیری کلمهٔ **long** تضمین کنندهٔ ۳۲ بیتی بودن نوع صحیح و به‌کارگیری کلمهٔ **short** تضمین کنندهٔ ۱۶ بیتی بودن نوع صحیح و به نوعی صرفه‌جویی در حافظه است. همچنین نوع دادهٔ **long double** جهت ذخیره سازی اعداد اعشاری با دقتی تا ۱۹ رقم اعشار است. البته مطابق جدول ۳-۲ تفاوتی بین کلمهٔ **short** و نوع **short int** نیست و شما می‌توانید در به‌کارگیری انواع، از اسامی دیگر آنها به‌راحتی استفاده کنید. همچنین از حالت‌های مختلف نوع دادهٔ **char** می‌توان برای ذخیرهٔ اعدادی هم که در بازهٔ ۱۲۸- تا ۲۵۵ قرار دارند نیز استفاده کرد، البته نحوهٔ بکارگیری این نوع برای ذخیره‌سازی اعداد در فصول آتی بیان خواهد شد.

برای نوع صحیح در کامپایلرهای شرکت مایکروسافت انواعی تهیه شده است که اندازهٔ هر کدام از این انواع بر حسب بیت به دنبالهٔ آن نوع ذکر شده است. و هدف از تهیهٔ این انواع صرفه‌جویی در حافظه و همچنین پشتیبانی کردن از محدوده‌های بزرگتر نسبت به بازهٔ قابل قبول **long int** می‌باشد. اصولاً در دیگر کامپایلرها نیز چنین انواعی پیش‌بینی شده و به‌کارگیری آنها به شدّت توصیه می‌شود. این انواع که در ابتدای نام آنها از دو کاراکتر زیرواژه[1] (_) استفاده شده است عبارتند از: **__int 64 , __int32 , __int 16 , __int8** ، جهت اطلاعات بیشتر در خصوص بازهٔ قابل قبول این انواع به جدول ۳-۲ مراجعه کنید.

دامنهٔ اشغال شده توسط انواع داده‌ای مختلفِ C++، در سرفایل **limits.h** ارائه شده است. برای مشاهدهٔ محتویات این سرفایل می‌توانید به پیوست شمارهٔ ۳ مراجعه کنید. همچنین اگر بر فرض کامپایلر **VC6** خود را در درایو **C** خود نصب کرده باشید می‌توانید به آدرس زیر مراجعه‌کنید و سرفایل **limits.h** را به واسطهٔ ویرایشگر **VC6** یا هر ویرایشگر دیگری باز کنید و محتویات آن را مشاهده کنید.

C:\Program Files\Microsoft Visual Studio\VC98\Include

دیگر سرفایل‌های C++ نیز در همین آدرس قرار دارند.

---

1. underline

نامگذاری متغیرها

با قوانین نامگذاری متغیرها در فصل دوم آشنا شدید. در C++ نیز از همان قوانین برای نامگذاری متغیرها استفاده می‌شود، جز در دو مورد استثنائی که در زیر آمده است:

۱. نام متغیرها هر تعداد کاراکتری که می‌خواهید می‌تواند باشد امّا در VC6 تنها ۲۴۷ کاراکتر اول و یا در بورلند C++ ۲۵۰ کاراکتر اول نام متغیر مورد استفاده قرار می‌گیرد.

۲. در نامگذاری متغیرها نمی‌توان کلمات رزروی زبان C++ را به‌عنوان نام متغیر انتخاب کرد. لیست کلمات رزروی زبان C++ در جدول ۳-۳ آمده است. البته هیچ نیازی نیست که این جدول را حفظ کنید چرا که به محض تایپ شدن اکثر کلمات رزروی C++ در محیط ویرایشگر VC6، ویرایشگر این کلمات را شناخته و آنها را به رنگ آبی در می‌آورد که این خود نشانه خوبی برای تشخیص رزروی بودن یک کلمه است و خود مبین آن است که نباید این کلمه را به‌عنوان نام متغیر به‌کار برد.

align	enum	private	typedef	__fainaly
allocate	explicit	property	typeid	__foreceinline
auto	extern	protected	typename	__inline
bool	false	public	Union	__int16
break	float	register	unsigned	__int32
case	for	reinterpret_cast	using declaration	__int64
catch	friend	return	using directive	__int8
char	goto	selectany	uuid	__interface
class	if	short	Virtual	__leave
const	inline	signed	Void	__multiple_ inheritance
const_cast	int	sizeof	Volatile	__noop
continue	long	static	wchar_t	__pragma
default	mutable	static_cast	While	__ptr64
delete	naked	struct	__alignof	__sealed
deprecated	namespace	switch	__asm	__single_ inheritance
dllexport	new	template	__assume	__stdcall
dllimport	noinline	this	__based	__super
do	noreturn	thread	__cdecl	__try / __except / __finally
double	nothrow	throw	__declspec	__unaligned
dynamic_cast	novtable	true	__expect	__uuidof
else	operator	try	__fastcall	__virtual_ inheritance

جدول ۳-۳ : لیست کلمات رزروی Microsoft Visual C++ 6.0

کلماتی که در جدول فوق با کاراکتر زیرواژه آغاز شده‌اند مخصوص مایکروسافت می باشند.

تعریف متغیرها

برای اعلان یک متغیر تنها کافی است نوع داده و سپس نام متغیر را ذکر کنید.

مثال ۳-۱: در مثال زیر تعدادی متغیر تعریف شده است.

```
int      x;
float y,z,f;
unsigned long int  min_mark;
unsigned __int32  Sq;
```

تذکر ۱: در پایان همهٔ دستورات C++ باید یک علامت سمیکالن (;) به‌عنوان کاراکتر نشان‌دهندهٔ پایان خط قرار دهید.

تذکر۲: در تعریف متغیرها می‌توان با بکارگیری عملگر کاما (,) هم زمان بیش از یک متغیر را تعریف کرد. چنانکه در مثال فوق همزمان دو متغیر y و z را به‌عنوان متغیرهایی از نوع **float** تعریف کرده‌ایم.

مقدار دهی اولیه به متغیرها

هنگامی که متغیری را تعریف می‌کنید، فضایی از سگمنت داده‌ها، به آن نسبت داده می‌شود. اما از آنجایی که امکان دارد بیت‌های آن بخشی از حافظه که به این متغیر اختصاص داده شده، از قبل دارای هر آرایشی از صفر و یک باشد، در صورت استفاده از این متغیر در برنامه بدون مقداردهی اولیه، ممکن است با خطای زمان اجرا برخورد کنید[۱]. لذا بهتر است در هنگام تعریف یک متغیر به آن مقدار اولیه بدهید، تا از خطای مذکور جلوگیری کنید. برای مقداردهی به متغیرها سه روش وجود دارد:

۱. در زمان تعریف متغیر، که به آن مقداردهی اولیه **(Initialization)** گفته می‌شود.
۲. پس از تعریف با دستور انتصاب که به آن مقداردهی کردن **(Assignment)** گفته می‌شود.
۳. به وسیلهٔ دستورات ورودی

چنانکه ذکر رفت در هنگام تعریف یک متغیر می‌توان فوراً به آن مقدار اولیه داد. به‌عنوان مثال دستور زیر را در نظر بگیرید:

```
int    X,Z=0;
char   ch ='h';
```

در مثال فوق به متغیر **Z** مقدار اولیه‌ای برابر صفر داده شده و متغیر **X** بدون مقدار اولیه دهی رها شده است. توجه کنید که برای به‌دست‌آوردن کد اسکی حرف انگلیسی **h** اطراف آن را با علامت ' محصور کرده‌ایم.

همچنین می‌توان در هر کجای برنامه با دستور انتساب یک متغیر را به هر مقدار دلخواهی مقداردهی کرد. (که شامل مقداردهی اولیه نمی‌شود هرچند که آن متغیر پیشتر در هنگام تعریف مقدار دهی نشده باشد). دستور انتساب در الگوریتم‌ها به‌صورت ← به‌کار می‌رفت. امّا در C++ به وسیلهٔ علامت (=) عمل انتساب صورت می‌گیرد و چنانکه در بخش۳-۳ خواهید دید برای عملگر ریاضی تساوی از نماد (= =) استفاده می‌شود.

---

۱. در کاردرکلاس ۳-۳ با ذکر مثالی این مطلب را بیشتر بررسی خواهیم کرد.

مثال ۳-۲: در مثال زیر متغیرهایی را که در مثال ۳-۱ تعریف کردیم مقداردهی می‌کنیم.

```
Sq = 204;
min_mark = Sq;
y = z = 35.21459f;
f = 3.25e-23;
```

توضیح مثال: در مثال فوق می‌بینید که متغیر **Sq** را با مقدار ثابت ۲۰۴ مقداردهی کرده‌ایم. و سپس متغیر **min_mark** را به‌وسیلهٔ متغیر **Sq** مقداردهی کرده‌ایم یعنی هر دو متغیر مذکور دارای مقدار مشابه‌ای شده‌اند. همچنین در خط سوّم هم زمان دو متغیر **y** و **z** را مقداردهی کرده‌ایم، به‌طوری که ابتدا عدد اعشاری **35. 21459** در متغیر **z** قرار می‌گیرد و سپس متغیر **y** به واسطهٔ **z** مقداردهی می‌گردد. و اما در خط چهارم مقدار متغیر **f** با یک عدد علمی معادل 3.25×10^{-23} مقداردهی شده است. توجه کنید که کاراکتر **f** در انتهای خط سوم معرف نوع **float** است و هیچ ارتباطی با متغیر **f** تعریف شده در برنامه ندارد.

روش سومی که به‌وسیلهٔ آن می‌توان متغیرها را مقداردهی کرد، به‌کارگیری دستورات ورودی برای این منظور است. چنانکه در کارنما به واسطهٔ جعبه های ورودی متغیرها را مقداردهی می کردیم در برنامه نویسی هم می‌توانیم به وسیلهٔ دستورات ورودی متغیرها را مقداردهی کنیم. در ادامه مثالی از این نوع را خواهید دید.

اعلان ثوابت

ثوابت مقادیری هستند که در برنامه وجود دارند و قابل تغییر نیستند. برای اعلان ثوابت دو روش وجود دارد:

۱. استفاده از دستور **#define** که به‌صورت کلی زیر است:

```
#define      نام ثابت      مقدار ثابت
```

تذکرات مهم : نامگذاری ثوابت از قانون نامگذاری متغیرها تبعیت می کند. در هنگام تعریف ثابت با دستور **#define** مقداری که برای ثابت تعیین می‌شود، نوع ثابت را نیز مشخص می‌کند. دقت داشته باشید که در انتهای دستور **#define** علامت ; قرار نمی‌گیرد. علتش این است که این دستور، از دستورات پیش پردازنده است، نه دستور زبان **C++**. پیش پردازنده یک برنامهٔ سیستم است که قبل از ترجمهٔ برنامه توسط کامپایلر، تغییراتی در آن ایجاد می‌کند. پیش‌پردازنده مقدار ثابت را که در دستور **#define** آمده است، به جای نام ثابت در برنامه قرار می‌دهد و این دستور در زمان اجرا وجود ندارد. نام دیگر ثابتی که به این روش تعریف می‌شوند، ماکرو است. برای تفکیک ثوابت از متغیرهای برنامه، بهتر است تمامی نام آنها با حروف بزرگ انتخاب شود. البته به جهت عدم تعیین نوع به‌طور صریح، در دستور فوق امکان بروز خطا در برنامه با استفاده از این روش وجود دارد لذا در **C++** استفاده ار این روش توصیه نمی‌شود.

۲. استفاده از دستور **const** که به‌صورت کلی زیر است:

```
const      نوع دادهٔ ثابت      نام ثابت      =      مقدار ثابت      ;
```

اصولاً برنامه‌نویسان **C++** بیشتر از روش اخیر برای تعریف ثوابت استفاده می‌کنند، البته در این روش در صورت عدم ذکر نوع، نوع ثابت برابر **int** فرض می‌شود. مثال‌های این موارد را نیز در ادامه خواهید دید.

۳-۳ : انواع عملگر در C++

با عمـلگرها و نقش آنها در الگوریتم‌ها در فصل قبل آشنا شدید، لذا در اینجا تنها به اشاراتی کوتاه بسنده می‌کنیم و تنها به بیان ناگفته‌ها می‌پردازیم. مقصود از عملگر هر علامتی است مثل علامت جمع که قادر باشد عملیاتی را برروی متغیرها و یا مقادیر عددی انجام دهد. هر عملگر برروی یک یا دو متغیر یا مـقدار ثابت عمل می‌کند. متغیرها و یا مقادیری را که عملگرها بر روی آنها عمل می‌کنند، عملوند نامیده می‌شوند. برخی از عملگرها حتماً نیاز به دو عملوند دارند مثل عملگر جمع و برخی دیگر تنها نیاز به یک عملوند دارند مثل عملگر یکتایی منها. عملگرها در C++ بر چهار دستهٔ زیر تقسیم می‌شوند:

۱. عملگرهای حسابی
۲. عملگرهای رابطه‌ای
۳. عملگرهای منطقی
۴. عملگرهای بیتی

از بین چهار نوع عملگر فوق سه نوع عملگر اول را در همین فصل بررسی می‌کنیم. اما به دلیل این که عملگرهای بیتی به ویژگی‌های سطح پایین زبان C++ مربوط می‌شوند و فعلاً کاربرد زیادی برای ما ندارند در فصل جداگانه‌ای [هنگامی که ارتباط زبان C++ را با زبان اسمبلی عنوان می‌کنیم،] مورد بررسی قرار خواهیم داد.

بررسی عملگرهای حسابی

عملگرهای حسابی، عملگرهایی هستند که اعمال جبری را روی عملوندها انجام می‌دهند. با اکثر این عملگرها در فصل قبل آشنا شده‌اید و از آنها در عبارات محاسباتی(جبری) استفاده کردید. چنانکه در بخش ۲-۱ دیدید عملگرهای - ، + ، * ، / جهت چهار عمل اصلی ریاضی به‌کار می‌روند. عملگر % برای محاسبهٔ باقیماندهٔ تقسیم به‌کار می‌رود. این عملگر، عملوند اول را بر عملوند دوم تقسیم می‌کند (تقسیم صحیح) و باقیمانده را برمی‌گرداند. اما بیایید با دو عملگر جدید آشنا شویم. عملگر کاهش (--) که یک واحد از عملوندش کم می‌کند و نتیجه را در آن عملوند قرار می‌دهد و عملگر افزایش (++) که یک واحد به عملوندش اضافه کرده نتیجه را در آن عملوند قرار می‌دهد.

مثال ۳-۳ : دستورات زیر را در نظر بگیرید:

```
int x=10,m=15;
x++;
--m;
```

توضیح مثال : در مثال فوق ابتدا متغیرهای x و m با مقدار اولیه ۱۰ تعریف شده‌اند. دستور ;x++ یک واحد به x می‌افزاید و نتیجه را در x قرار می‌دهد و دستور ;m-- یک واحد از m می‌کاهد و نتیجه را در m قرار می‌دهد. در نتیجه در پایان در متغیر x مقدار ۱۱ و در متغیر m مقدار ۱۴ قرار دارد. بنابراین در این نوع دستورات، عملگر افزایش(یا کاهش) چه قبل از عملوند باشد چه بعد از آن، عملکرد آنها یکسان است. امَا عملکرد آنها در عبارات محاسباتی با هم متفاوت است. به مثال ۳-۴ توجه کنید.

مثال ۳-۴: در مثال زیر چگونگی تفاوت عملگرهای افزایش و کاهش در عبارات محاسباتی نشان داده شده است.

```
int x = 5 , y ;
y = 10;
int m = ++x + y--;
```

توضیح مثال: در مثال فوق ابتدا دو متغیر x و y از نوع صحیح تعریف شده اند، و متغیر x با مقدار ۵ مقداردهی اولیه شده است اما متغیر y بدون مقداردهی اولیه رها گشته. اما در خط دوم متغیر y به عدد ۱۰ مقداردهی شده است. و در نهایت در خط سوم با تعریف متغیر m از نوع صحیح آن را به‌واسطهٔ یک عبارت محاسباتی مقدار اولیه داده‌ایم. اما چگونگی محاسبهٔ حاصل این عبارت مهم است، چراکه بر حسب چگونگی ارزیابی عبارت محاسباتی می‌توان گفت چه مقداری در متغیر m قرار گرفته است. نکته قابل توجه در این خصوص آن است که، اگر عملگرهای ++ و -- در عبارات محاسباتی، قبل از عملوند قرار گیرند، ابتدا این عملگرها عمل کرده و نتیجهٔ عملیات آنها در محاسبهٔ عبارت شرکت می‌کند. ولی اگر بعد از عملوند ظاهر شوند، ابتدا عملوند با مقدار فعلی خود وارد محاسبات می شود، سپس عملگر برروی عملوند خود عمل می‌کند. بنابراین در مثال فوق ابتدا مقدار x به عدد ۶ ارتقاء می‌یابد و این عدد با مقدار فعلی y [یعنی ۱۰] جمع‌شده و حاصل آنها در متغیر m قرار می‌گیرد، سپس مقدار y یک واحد کاهش می‌یابد، که البته هیچ اثری در محاسبهٔ مقدار عبارت محاسباتی ما ندارد.

در جدول ۳-۴ عملگرهای محاسباتی C++ براساس درجهٔ تقدم لیست شده‌اند. چنانکه در بخش ۲-۸ نیز متذکر شدیم عملگرهایی که تقدم آنها یکسان است، نسبت به هم تقدم مکانی دارند. یعنی، هر کدام که زودتر ظاهر شود، همان عملگر زودتر عمل می کند. [در C++ عبارات از سمت چپ ارزیابی می‌شوند.]

ملاحظات	عملگرها	تقدم
در حالت پیشوندی دارای بیشترین تقدم هستند.	++ , --	بالاترین سطح
منظور منهای یکتایی است. (قرینه سازی).	-	
	* , / , %	
	+ , -	پایین‌ترین سطح

جدول ۳-٤ : عملگرهای حسابی C++ بر حسب تقدم (به ترتیب کاهش تقدم)

چنانکه در جدول ۳-۴ مشاهده می‌کنید در زبان C++ هیچ عملگری برای عمل به توان رساندن اعداد تعریف نشده است. پس مبادا به اشتباه عملگر ^ را برای توان به‌کار گیرید، این عملگر صرفاً در الگوریتم‌نویسی برای ما اعتبار دارد. البته از این نماد در بسیاری از زبان های برنامه نویسی به عنوان عملگر توان استفاده شده است، اما متأسفانه در زبان C++ هیچ عملگری برای انجام عمل توان نداریم، اما بعداً در بخش توابع ریاضی در فصل توابع، تابعی کتابخانه‌ای را جهت عمل به توان رساندن معرفی خواهیم کرد.

بررسی عملگرهای رابطه‌ای

با عملگرهای رابطه‌ای نیز در فصل قبل آشنا شدید. تنها تفاوتی که عملگرهای رابطه‌ای C++ با عملگرهای رابطه‌ای ریاضی مطرح شده در فصل قبل دارد شکل ظاهری آنها ست. در جدول ۳-۵ لیست عملگرهای رابطه‌ای در زبان C++ را مشاهده می‌کنید.

عملگر رابطه ای در C++	معادل ریاضی عملگر رابطه ای	مثال
>	>	x > y
>=	≥	x >= y
<	<	x < y
<=	≤	x <= y
==	=	x == y
!=	≠	x != y

جدول ۳-۵ : عملگرهای رابطه‌ای در C++

بررسی عملگرهای منطقی

با عملگرهای منطقی و جدول درستی دو عملگر **and** و **or** در فصل قبل آشنا شدید. چنانکه می‌دانید عملگرهای منطقی برروی گزاره‌ها عمل می‌کنند. در زبان C++، ارزش نادرستی یک عبارت منطقی با مقدار صفر و ارزش درستی یک عبارت منطقی با مقادیر غیر صفر نشان داده می‌شود. بنابراین هر مقدار غیر صفری در C++ برای ما ارزش درست دارد، اما اصولاً ارزش درستی را با مقدار یک نشان می‌دهند. دو ثابت **true** و **false** در زبان C++ برای نشان دادن مقادیر صفر و یک در عملیات منطقی به‌کار گرفته می‌شوند. چنانکه در بخش ۳-۲ در خصوص نوع دادهٔ منطقی مطرح شد، این نوع داده قادر است مقادیر صفر و یک حاصل از عملیات منطقی را در خود ثبت کند. در احکام انتساب، ثابت **true** به ۱ و ثابت **false** به صفر تبدیل می‌شود. به‌عنوان مثال با نسبت دادن یک متغیر از نوع **bool** که در آن مقدار **true** ذخیره شده است به یک متغیر از نوع صحیح، مقدار ثابت **true** به یک تبدیل می‌شود و در متغیر نوع **int** قرار می‌گیرد. عکس این مطلب نیز صحیح است. در جدول ۳-۶ عملگرهای منطقی به ترتیب کاهش تقدم، از بالا به پایین لیست شده‌اند.

عملگر منطقی در C++	معادل الگوریتمی عملگر	مثال
!	نقیض (not)	x!
&&	وَ (and)	x > y && m < n
||	یا (or)	x != y || m==n

جدول ۳-٦ : عملگرهای منطقی به ترتیب کاهش تقدم [از بالا به پایین]

پیشتر در جدول ۲-۵ عملگرهای رابطه‌ای و منطقی را از نظر تقدم در کنار یکدیگر ملاحظه کرده‌اید. سطوح تقدم مطرح شده در این جدول در خصوص زبان C++ نیز صادق است. اما در این جدول عملگر نقیض به‌طور استثناء پیش از عملگرهای رابطه‌ای قرار می‌گیرد.

عملگرهای ترکیبی

از ترکیب عملگرهای حسابی و عملگر انتساب (=)، مجموعهٔ دیگری از عملگرها ایجاد می‌شود که عملیات محاسباتی و انتساب را به‌طورهم‌زمان انجام می‌دهد. در جدول ۳-۷ لیست این عملگرها و کاربرد آنها را می‌توانید مشاهده کنید. تقدم این عملگرها از تمامی عملگرها پایین‌تر است.

عملگر ترکیبی در C++	مثال	معادل
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

جدول ۳-۷ : لیست عملگرهای ترکیبی در C++

عملگر ( )

چنانکه در فصل دوم دیدید پرانتزها عملگرهایی هستند که تقدم عملگرهای داخل خود را بالا می‌برند.

عملگر sizeof

این عملگر می‌تواند طول یک متغیر یا طول یک نوع داده را برحسب بایت در زمان کامپایل(ترجمه) برگرداند. شاید برای شما فعلاً این عملگر بدون کاربرد باشد، اما در فصول آتی مثال‌هایی از نحوهٔ به‌کارگیری این عملگر و اهمیت آن را خواهید دید. این عملگر برای تعیین طول یک متغیر و یا یک نوع داده به دو صورت مـتفاوت به‌کار می‌رود. شکل کلی کاربرد آن به‌صورت زیر است. البته توجه داشته باشید که در اطراف نام متغیر نیز می‌توان از علامت پرانتز بهره گرفت ولی به‌کارگیری آن اطراف نوع داده الزامی است.

```
sizeof      نام متغیر ;
sizeof      (نوع داده) ;
```

مثال ۳-۵: در مثال زیر چگونگی به‌کارگیری عملگر **sizeof** نشان داده شده است.

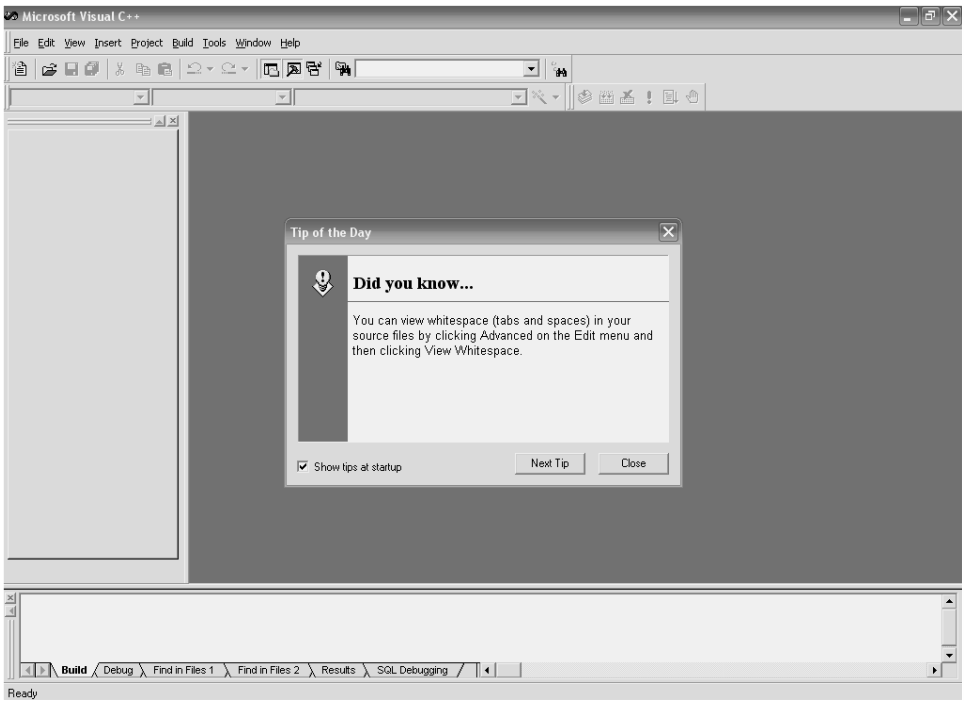
```
int   size_var,size_type,x;
x = 5;
size_var=sizeof (x);
size_type=sizeof(char);
```

توضیح مثال: در مثال فوق ابتدا سه متغیر از نوع صحیح تعریف شده اند. در خط سوم طول متغیر صحیح **x** در داخل متغیر **size_var** قرار داده شده است. نکتهٔ قابل توجه آنکه مقداری که در متغیر **x** ذخیره شده است هیچ تأثیری بر عملگر **sizeof** ندارد چنانکه عدد۴ را به‌عنوان طول متغیر **x** که از نوع **int** است برمی‌گرداند. در خط چهارم طول نوع داده‌ای **char** را در متغیر **size_type** ذخیره کردیم، که مسلماً عدد یک در این متغیر ذخیره شده است. به به‌کارگیری پرانتز در اطراف نوع داده‌ای و دلخواه بودن به‌کارگیری پرانتز در اطراف نام متغیر خوب توجه داشته باشید. اگر این قطعه کد را مطابق آن چه که در ادامه خواهید دید، در کامپایلر **VC6** اجرا کنید و پرانتز اطراف نوع داده‌ای **char** را قرار ندهید با خطا روبرو خواهید شد اما در خصوص نام متغیر شما می‌توانید از پرانتز به دلخواه خود استفاده کنید یا نکنید.

۳-۴ : نخستین تجربهٔ برنامه نویسی در محیط Visual C++ 6 .0

آماده سازی محیط ویرایشگر VC6

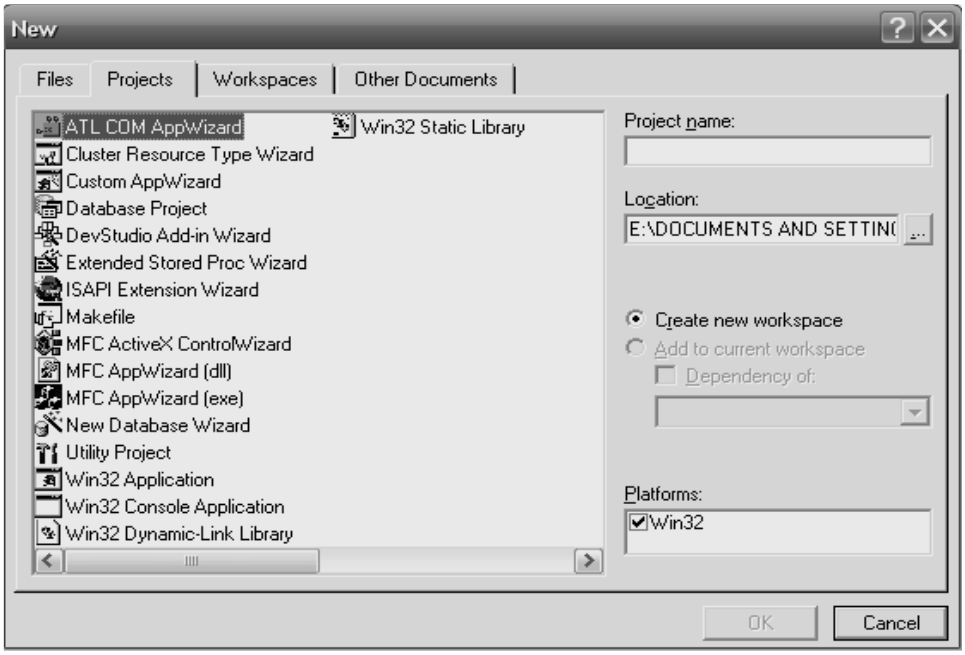
در صورتی که مراحل نصب محیط مجتمع **Microsoft Visual Studio 6.0** – مطابق پیوست ۱– را با موفقیت طی کرده باشید، دقیقاً یک آیکون با همین نام در منوی **All Programs** در منوی **Start** ویندوز شما ایجاد شده است. همچنین در صورتی که با استفاده از لوح های فشردهٔ ۳ و ۴ مجموعهٔ **MSDN** را هم نصب کرده باشید، آیکون دیگری نیز با نام **Microsoft Developer Network** در منوی **All Program** ایجاد شده است. از طریق **MSDN** می توانید به مستنداتی در خصوص کامپایلر **VC6** دست پیدا کنید و با جستجو در محیط **MSDN** مشکلات احتمالی خود را برطرف‌سازید. اما برای آغاز کار برنامه‌نویسی گزینهٔ **Microsoft Visual C++ 6.0** را از داخل منوی **Microsoft Visual Studio 6.0** انتخاب کنید تا صفحه‌ای مطابق شکل ۳-۴ پیش روی شما قرارگیرد.



شکل ۳-۴ : صفحهٔ آغازین محیط VC6

پس از باز شدن محیط **VC6** پنجره‌ای با عنوان **Tip of the day** باز می گردد که در آن توضیحاتی در خصوص این کامپایلر ارائه شده است. با دکمهٔ **Next Tip** می توانید توضیحات بعدی را مشاهده کنید و با دکمه **Close** می توانید پنجره را ببندید. در صورتی که تمایل دارید در دفعات بعدی که به **VC6** مراجعه می کنید این پنجره باز نشود علامت تیک جلوی پیغام **Show tip at startup** را قبل از بستن پنجره، بردارید.

با انتخاب گزینهٔ **New** از منوی **File** پنجره‌ای را مطابق شکل ۳-۵ مشاهده خواهید کرد.

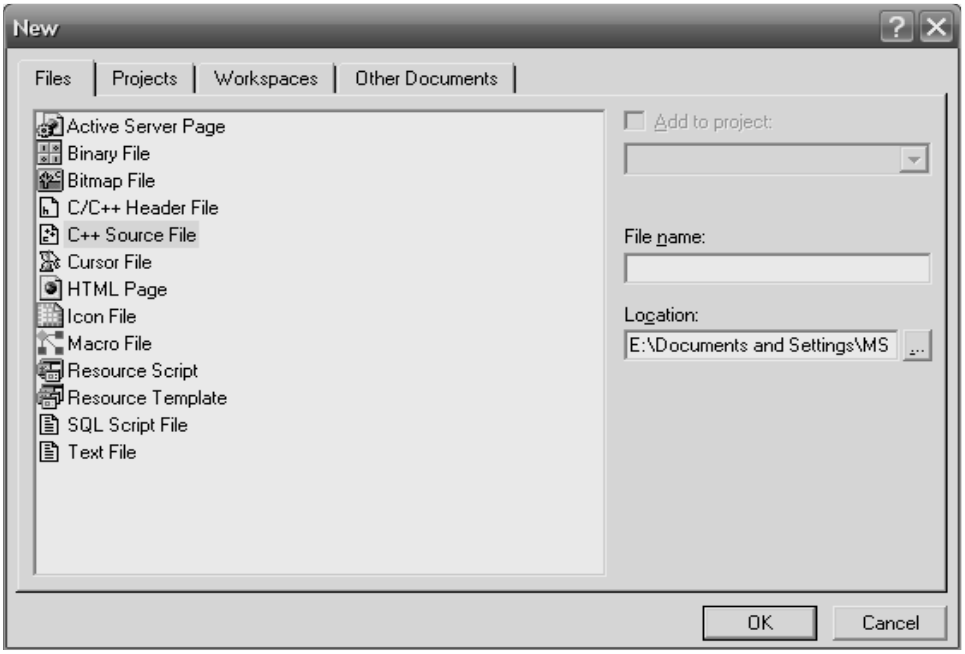


شکل ۳-۵ : پنجرهٔ New در حالت Projects

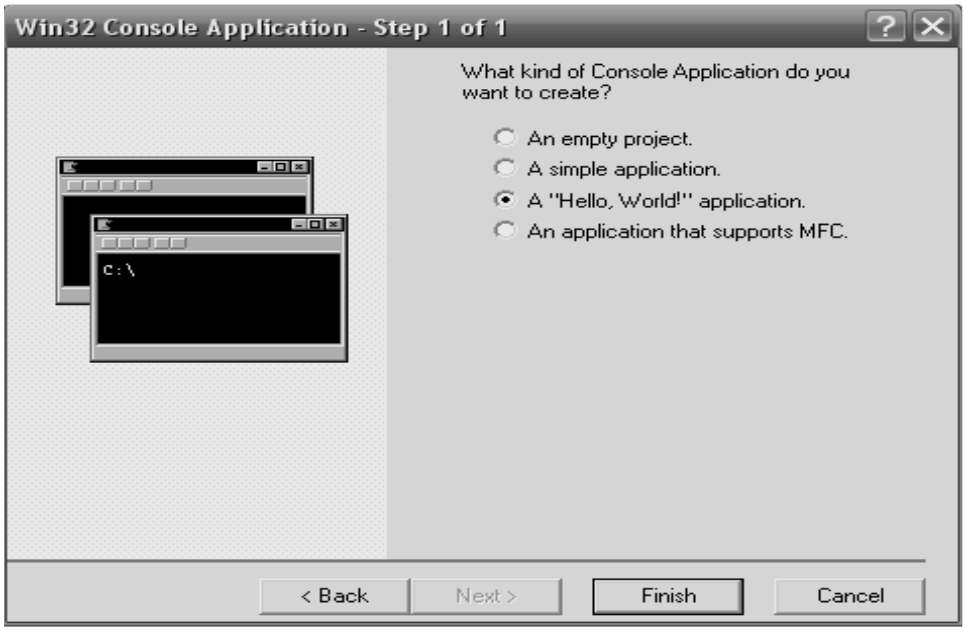
دو روش برای ایجاد برنامه‌های C++ در محیط VC6 وجود دارد. ابتدا روش ساده‌تر را بیان می‌کنیم، شما نیز برای ایجاد برنامه‌های C++ خود در جلد اول از این روش استفاده کنید.

۱. برای ایجاد برنامه C++ گزینهٔ **Files** را از پنجرهٔ **New** انتخاب کنید تا پنجره‌ای را مطابق شکل ۳-۶ مشاهده کنید. در این پنجره ضمن انتخاب گزینهٔ **C++ Source File** ، نام برنامهٔ خود را در قسمت **File Name** وارد کرده و سپس در قسمت **Location** محل ذخیرهٔ برنامهٔ خود را تعیین کنید. با فشردن دکمهٔ **ok** صفحه‌ای خالی جهت وارد کردن کد برنامه برای شما باز می‌شود. در این حالت یک فایل با پسوند **cpp.** و با نامی که شما انتخاب کرده‌اید ایجاد می‌شود. به خاطر داشته باشید پسوند فایل‌هایی که کد برنامه‌های خود را در آن قرار می‌دهیم **cpp.** باید باشد، تا توسط کامپایلر قابل ترجمه باشد.

۲. روش دیگر، ایجاد پروژه بر مبنای **Win 32 Console Application** است. در شکل ۳-۵ ضمن انتخاب گزینهٔ مذکور و وارد کردن نام پروژهٔ خود دکمهٔ **ok** را بزنید تا پنجره‌ای مطابق شکل ۳-۷ باز شود. در این پنجره گزینهٔ **A "Hello, World" application** را انتخاب کنید و دکمهٔ **Finish** را بفشارید تا قالب پروژهٔ موردنظر شما ایجاد گردد. پس از آنکه نوشتن برنامه در C++ استاندارد را فراگرفتید در خصوص نوشتن پروژه‌های **Win32** هم صحبت خواهیم کرد. لذا فعلاً از این روش برای ایجاد برنامه‌های C++ خود استفاده نکنید. برای نوشتن هر برنامه‌ای ابتدا باید مرحلهٔ ۱ را برای ایجاد پروژه بپیمایید.



شکل ۳-۶ : پنجرهٔ New در حالت Files



شکل ۳-۷ : پنجرهٔ Win 32 Console Application

اجرا کردن نخستین برنامه به زبان C++

مثال ۳-۶: برنامه‌ای که در آن پیام "Hello, World" برروی صفحه چاپ می‌شود.

۱. مطابق آنچه گفته شد یک فایل **cpp.** با نام **First** ایجاد کنید.

۲. کد زیر را به‌طور دقیق در آن تایپ کنید.

```
#include <stdio.h>
void main()
{
 printf("Hello,World.\n");//print the massage
}
```

۳. با انتخاب گزینهٔ **Compile** و یا زدن همزمان دکمه‌های **Ctrl** و **F7** برنامهٔ خود را ترجمه(کامپایل) کنید[۱]. در این مرحله باید پیغام **0 error(s), 0 warning(s)** را در پنجرهٔ **Build** ، در پایین صفحه دریافت کنید. اگر تعداد خطاها غیر از صفر باشد فایل اجرایی این کد قابل ساختن نخواهد بود، اما تعداد هشدارها فعلاً چندان مهم نیست. البته این بدان معنا نیست که از این پس به هشدارها توجه نکنید، چرا که برخی از اوقات توجه به هشدارها می‌تواند جلوی بسیاری از اشکالات در برنامه را بگیرد، لذا تنها در همین مثال به هشدارها بی‌توجه هستیم. اگر تعداد خطاها غیر از صفر است یک بار دیگر کد خود را کنترل کنید و با کتاب مطابقت دهید. مثلاً ممکن است فراموش کرده باشید علامت سمیکالن را در انتهای خط چهارم قرار دهید. از طرفی می‌توانید نوع خطا و محل آن را به واسطهٔ پیغام‌هایی که در جعبهٔ **Build** اعلام می‌شود مشاهده کنید. به‌عنوان مثال سمیکالن انتهای خط چهارم را بردارید و یک با دیگر کد خود را کامپایل کنید. در این صورت پیغام زیر را می‌توانید در این پنجره بیابید:

error C2143: syntax error : missing ';' before '}'

1 error(s), 0 warning(s)

با کلیک کردن برروی این پیغام یک پیکان در کنار صفحه شما را به محل خطا راهنمایی می‌کند. که می‌توانید به آنجا مراجعه و خطا را رفع کنید.

۴. با انتخاب گزینهٔ **Execute Program** از منوی **Build** و یا فشردن هم‌زمان کلیدهای **Ctrl** و **F5** برنامهٔ خود را اجرا کنید تا پیام **"Hello, World"** را بر روی صفحهٔ کنسول ببینید[۲]. با زدن یک کلید اجرای برنامه را خاتمه دهید.

---

۱. اگر برروی نوار ابزار جستجو کنید کلیدی را خواهید یافت که اگر چند لحظه اشاره‌گر را برروی آن نگه دارید کلمهٔ **Compile(ctrl+F7)** برروی آن نمایان می‌شود. به‌واسطهٔ این کلید نیز می‌توانید عمل کامپایل را انجام دهید.

۲. برای اجرا کردن برنامه نیز می‌توانستید دکمه‌ای را که دارای علامت ! است را از داخل نوار ابزار بفشارید.

۳-۵: ساختار کلی برنامه در C++

مثال ۳-۶ را به این جهت عنوان کردیم که در این قسمت راحت‌تر بتوانید مطالب مرتبط با ساختار برنامه در C++ را دنبال کنید. مباحثی که در این بخش با عنوان ساختار کلی برنامه مطرح می‌سازیم بسیار محدود است، به‌طوری که هر چه جلوتر برویم با قسمت‌ها و ساختارهای دیگری از برنامه‌های C++ آشنا خواهید شد.

تابع اصلی (تابع main)

اصلی‌ترین قسمت هر برنامه در C++ تابع اصلی آن است که با نام **main** شناسایی می‌شود. در حقیقت سیستم عامل برای اجرای برنامه به دنبال تابع اصلی آن می‌گردد، و پس از یافتن این تابع اجرای برنامه از اولین خط این تابع آغاز شده و تا آخرین خط آن ادامه می‌یابد.

هر تابع در برنامه‌های C++ دارای یک بدنه است. بدنهٔ هر تابع که شامل یک‌سری دستورات و فرمان‌هایی برای کامپیوتر است که در بین یک جفت آکولاد باز و بسته قرار می‌گیرد. فعلاً به پرانتز باز و بسته‌ای که پس از نام تابع اصلی آمده است کاری نداشته باشید، ولی در هنگام نوشتن کد حتماً آن را قرار دهید.

در C++ تمامی توابع باید دارای یکی از انواع موجود در C++ باشند. هر تابع بر اساس نوعی که دارد یک مقدار را به مجری خود برمی‌گرداند. مقصود از مجری یا فراخوانندهٔ هر تابع، کسی است که تابع را اجرا می‌کند. به‌عنوان مثال تابع اصلی توسط سیستم عامل فراخوانده می‌شود در نتیجه مجری تابع اصلی سیستم عامل است. همچنین ممکن است یک تابع فراخوانندهٔ تابع دیگری باشد که در این خصوص در فصل توابع بیشتر بحث خواهیم کرد. هر تابع باید مقداری را متناسب با نوعش به فراخوانندهٔ خود بازگرداند. مثلاً وقتی می‌گویند فلان تابع از نوع صحیح است منظور آن است که این تابع یک عدد صحیح را به فراخوانندهٔ خود بازمی‌گرداند. تابع **main** نیز از این قاعده مستثناء نیست، و باید مقداری را به سیستم عامل که فراخوانندهٔ آن است برگرداند. دلیل این امر آن است که سیستم عامل بفهمد تابع **main** با موفقیت به پایان رسیده است و تمامی دستورات آن با موفقیت اجرا گردیده. نوع تابع، قبل از نام تابع ذکر می‌گردد. به‌عنوان مثال اگر بخواهیم تابع اصلی از نوع صحیح باشد، این تابع دارای شکل کلی زیر خواهد بود.

```
int main()
{
    بدنهٔ تابع اصلی
   return 0;
}
```

چنانکه در قطعه کد فوق ملاحظه می‌کنید، هنگامی که تابع **main** از نوع **int** تعریف گردد باید توسط یک دستور **return** در آخرین خط خود یک مقدار صحیح برگرداند. همواره با دستور **return** مقداری متناسب با نوع تابع به مجری تابع برگردانده می‌شود. همچنین با رسیدن به این دستور، کنترل اجرای برنامه نیز به مجری تابع بازگردانده می‌شود، بنابراین اگر دستور یا دستوراتی پس از دستور **return** قرار گیرد به هیچ عنوان اجرا نخواهند شد. در خصوص تابع **main** مقدار بازگشتی هیچ اهمیتی ندارد که چه مقداری است، مثلاً در مثال فوق می‌توانیم به جای عدد صفر، عدد یک یا

هر عدد صحـیح دیگری را باز گردانیم اما به هیچ عــنوان نمی‌توانیم عددی اعشاری را به‌عنوان مقدار بازگشتی برای یک تابع **int** بازگردانیم. اصولاً برای توابع **main** ای که از نوع **int** تعریف می‌شوند مقدار صفر بازگردانده می‌شود.

چنانکه در مثال ۳-۶ نیز ملاحظه کردید اگر تابعی از نوع **void** تعریف شود نیازی به برگرداندن مقدار برای آن تابع نیست. امّا در **C++** استاندارد انجام این عمل برای تابع **main** خطا محسوب می‌شود. البته ممکن است برخی از کامپایلرها این خطا را نادیده بگیرند و یا آنکه به جای اعلام خطا یک هشدار صادر کنند ولی در هر صورت در مستندات مربوط به **C++** استاندارد وجود نوع صحیح برای تابع **main** الزامی شمرده شده است.

دستورات پیش‌پردازنده و سرفایل‌ها(هدرفایل)

به خط اول مثال ۳-۶ خوب توجه کنید. یکبار این خط را پاک کنید و دوباره برنامهٔ خود را کامپایل کنید. فکر می‌کنید با چه خطایی روبرو شده اید؟

اگر چنین عملی را انجام دهید با خطای عدم شناسایی دستور **printf** روبرو می‌شوید. در حقیقت دستورات پیش پردازنده که با علامت # شروع می‌شوند و به ; نیز ختم نمی‌شوند از اجزای اصلی برنامه محسوب نمی‌گردند، اما یک ناحیه از حافظه را اشغال کرده و وجود آنها هنگام اجرای برنامه ضروری است. دستورات پیش‌پردازنده جزء دستورات برنامه نیز محسوب نمی‌شوند چرا که دستورات برنامه به کامپیوتر فرمان انجام کاری، مثل جمع کردن دو متغیر را می دهند. اما کاربرد این دستورات در برنامه چیست؟

قسمتی از کامپایلر به نام پیش‌پردازنده قبل از آغاز فرآیند واقعی کامپایل این دستورات را مورد ارزیابی قرار می دهد. دستور پیش پردازندهٔ **#include** به کامپایلر می‌گوید، فایل دیگری را به فایل **cpp.** برنامهٔ ما اضافه کند. به این ترتیب به جای دستور **#include** محتوای فایل مشخص شده قرار می‌گیرد. اما این فایل‌هایی که باید اضافه شوند چه نقشی در برنامهٔ ما دارند؟

بسیاری از دستورات مفید و پرکاربرد در زبان **C++** در فایل‌هایی موسوم به هدرفایل(سرفایل) قرار دارند. به‌عنوان مثال اگر می‌خواستیم بدون استفاده از دستور **printf** پیغام خود را روی خروجی استاندارد (یعنی مانیتور) بنویسیم باید خطوط زیادی شامل کد اسمبلی می‌نوشتیم تا بتوانیم چنین پیامی را برروی صفحه نمایش چاپ کنیم. اما برای راحتی کار برنامه‌نویسان بسیاری از این کدها از قبل در فایل‌هایی نوشته شده و آمادهٔ استفاده می‌باشند. در واقع با دستورات پیش پردازنده ما می‌توانیم این فایل‌ها را به برنامهٔ خود اضافه کنیم، و از امکانات موجود در آنها استفاده نماییم. به‌عنوان مثال برای استفاده از دستور **printf** باید هدرفایل **stdio.h** را به برنامه اضافه کنیم. در مثال ۳-۶ دستور **#include <stdio.h>** به کامپایلر می‌گوید قبل از کامپایل برنامه، هدرفایل **stdio.h** را به برنامهٔ ما اضافه کند. سرفایل‌های جدید **C++** استاندارد، دارای پسوند فایل نیستند اما سرفایل‌های قدیمی که از **C++** مادر گرفته شده‌اند دارای پسوند **h.** هستند.

در هنگام استفاده از دستور **#include** نباید بین کلمهٔ **include** و # فاصله‌ای قرار داد. همچنین بین نام فایل و علامت های < و > نیز نباید فاصله قرار داد.

مستندسازی و توضیحات

معمولاً هر برنامه‌نویس پس از مدتی که به برنامه‌هایی که قبلاً نوشته مراجعه می‌کند با مشکلات متعددی در خصوص به یادآوری آنچه از قبل نوشته مواجه می‌شود. نمی‌داند فلان متغیر را برای چه منظوری در نظر گرفته بوده، فلان خط برنامه را برای چه نوشته و ... . در این بین مستندسازی کردن[1] در برنامه بسیار حائز اهمیت است. چرا که این مستندات هستند که می‌توانند ما را در فهمیدن نقاط کور برنامه کمک کنند. در C++ دو روش عمده برای مستند سازی و ارائهٔ توضیحات در برنامه وجود دارد.

روش اول که نمونهٔ آن را در مثال ۳-۶ دیدید، با قرار دادن // در انتهای هر خطی از برنامه قادر هستید، بعد از آن توضیحاتی را قرار دهید. به عبارت دیگر کامپایلر هرگاه به چنین علامتی برسد از آن نقطه تا پایان خط را نادیده می‌گیرد و به خط بعدی می‌رود. لذا علامت // باید پس از علامت ; قرار گیرد. به این نوع مستندسازی، توضیحات تک‌خطی گفته می‌شود.

اما روش دیگر مستند سازی، نوشتن توضیحات در چند خط، و یا به عبارت دیگر نوشتن بلوکی از توضیحات است. برای این منظور ابتدای توضیحات را با علامت */ آغاز کرده و انتهای آن را با علامت /* می‌بندیم. کامپایلر نیز محتویات کلیهٔ خطوطی که بین این دو علامت قرار می‌گیرد را به رنگ سبز در می‌آورد تا قابل تمایز باشد. به‌عنوان مثال توضیحات زیر را ببینید.

```
/*This Variable is for save the number of students.
   And initialize to zero.*/
```

مستند سازی در هنگام رفع عیوب برنامه، و همچنین به جهت خوانا کردن و قابل فهم کردن برنامه برای دیگر همکارانی که با شما برروی یک پروژه کار می‌کنند بسیار راه گشا می‌باشد.

نکات متفرقه در خصوص زبان C++

۱. در C++ کلیهٔ دستورات با حروف کوچک نوشته می‌شود. به‌عنوان مثال اگر تابع **main** را به‌صورت **Main** بنویسید برنامه با خطا روبرو خواهد شد.

۲. فضای خالی برای کامپایلر C++ بی‌اهمیت است. شما می‌توانید یک دستور C++ را در چندین خط بنویسید. تنها علامت ; است که انتهای خط و به عبارت بهتر انتهای دستور را برای کامپایلر مشخص می‌کند. با این حساب شما حتی می‌توانید دستورات برنامه را پشت سرهم در یک خط نیز بنویسید. اما از انجام چنین کاری به شدت خودداری کنید. اصولاً بهتر است هر خط را با کمی تو رفتگی نسبت به خط قبلی آغاز کنیم. در دو جا نمی‌توان از فضای خالی استفاده کرد. یکی در دستورات پیش‌پردازنده که قبلاً اشاره شد، دیگری در ثابت‌های رشته‌ای که اندکی بعد به آن اشاره خواهیم کرد.

---

1. Write Documents

۳-۶: ورود و خروج اطلاعات در C++

در زبان C ورود و خروج اطلاعات تا حدودی با پیچیدگی همراه بود. به‌عنوان مثال برای ورود اطلاعات باید به‌طور دقیق به کامپایلر بگوییم که چه نوع داده‌ای را باید بخواند. اما یکی از مهمترین مزیت‌های C++ نسبت به C افزوده شدن هدرفایل **iostream.h** بود که به وسیلهٔ اشیای موجود در این هدر فایل می‌توان به‌راحتی هر نوع داده‌ای را خواند و یا هر نوع اطلاعاتی را چاپ کرد.

چاپ اطلاعات به واسطهٔ شئ cout

به واسطهٔ شئ **cout** می‌توان هر نوع اطلاعاتی را به‌سادگی برروی صفحه نمایش چاپ کرد. برای استفاده از شئ **cout** باید هدرفایل **iostream.h** را به برنامه اضافه کنید. شکل کلی این دستور به‌صورت زیر است:

```
cout << عبارت ۱ << عبارت ۲ << ... ;
```

در این دستور *عبارت ۱* و *عبارت ۲* و ... مقادیری هستند که باید در صفحه نمایش چاپ شوند، این مقادیر باید به واسطهٔ عملگر << از یکدیگر جدا شوند. برای چاپ متن‌ها باید آنها را بین یک جفت کوتیشن قرار داد. همچنین می‌توان با استفاده از کاراکترهای کنترلی که لیست آنها در جدول۳-۸ آمده شکل خروجی را بهبود بخشید. به‌عنوان مثال با کاراکترهای کنترلی می‌توان مشخص کرد که آیا تمام اطلاعات در یک سطر چاپ شوند یا در چند سطر، آیا اطلاعات با فاصلهٔ خاصی چاپ شوند یا پشت سر هم. به‌عنوان مثال کاراکتر کنترلی **\n** موجب می‌شود که سطر جاری رد شود و به ابتدای سطر بعدی برویم. اطلاعاتی که پس از **\n** بیاید در سطر جدیدی چاپ خواهد شد. نکتهٔ بسیار مهم در خصوص کاراکترهای کنترلی آن که این کاراکترها حتماً باید داخل یک جفت کوتیشن قرارگیرند و یا در داخل متون ثابت جای داده شوند.

کاراکتر کنترلی	کاری که انجام می‌دهد
\n	سطر جاری را رد می‌کند.
\t	کنترل خروجی را به ابتدای ۸ محل بعدی می‌برد.
\a	بوق سیستم را به صدا در می‌آورد.
\\	کاراکتر \ را چاپ می‌کند.
\"	کاراکتر " را چاپ می‌کند.
\v	کنترل خروجی را به ابتدای ۸ سطر بعدی می‌برد.
\b	کاراکتر قبل از خودش را حذف می‌کند.
\r	عمل **carriage return** را مشخص می‌کند.
\?	کاراکتر ? را چاپ می‌کند.
\:	کاراکتر : را چاپ می‌کند.

جدول ۳-۸: کارکترهای کنترلی

خواندن اطلاعات به وسیلهٔ شئ cin

به‌واسطهٔ شئ **cin** می‌توان هر گونه اطلاعات را از ورودی استاندارد(یعنی صفحه کلید) خواند و در متغیرهای مورد نظر قرار داد. برای استفاده از شئ **cin** نیز نیازمند اضافه کردن هدرفایل **iostream.h** هستید. شکل کلی به‌کارگیری این دستور به‌صورت زیر است:

```
cin>> متغیر ۱ >> متغیر ۲ >> ... ;
```

وقتی چند قلم اطلاعات را توسط دستور **cin** می‌خوانید، هنگام ورود داده‌ها باید آنها را حداقل با یک فاصله[1] از هم جدا کنید و پس از وارد کردن تمام آنها، کلید **Enter** را فشاردهید. نوع داده‌های ورودی باید مشابه با نوع متغیرهایی باشد که در دستور **cin** قرارگرفته اند و گرنه امکان دارد برنامه با خطای زمان اجرا مواجه شود.

تذکر: برخی اوقات افراد تازه کار فراموش می‌کنند که کدام یک از علائم << یا >> را برای شئ **cout** و کدام یک را برای شئ **cin** بکار می‌بردند. برای جلوگیری از این اشتباه این علائم را به مانند یک قیف در نظر بگیرید. وقتی که می‌خواهیم اطلاعات را بر روی صفحه نمایش چاپ کنیم باید اطلاعات را به داخل خروجی بریزیم در این حالت سر قیف باید به سمت خروجی که همان **cout** است باشد. از طرفی هنگامی که اطلاعات را می‌خوانیم باید اطلاعات خوانده شده را به داخل متغیرها بریزیم در این صورت باید سر قیف به سمت متغیرها قرار گیرد.

مثال ۳-۷: برنامه‌ای که شعاع یک دایره را خوانده و مساحت و محیط آن را چاپ می‌کند.

```
1.  //This program shows the usage of cin and cout.
2.  #include <iostream.h>
3.  #define  PI  3.1415927 // تعریف ثابت
4.  int main ()
5.  {
6.      cout<<"Please enter the radius of your Circle :";
7.        float rd;
8.           cin>>rd;
9.             double  s= PI*rd*rd;
10.           double  p= PI*2*rd;
11.     cout<<"\n\n\tThe circle's area = \t"<<s;
12.      cout<<"\n\n\tThe circle's prime = \t"<<p<<endl;
13.    return 0;
14. }
```

توضیح مثال: در مثال فوق با مواردی چون تعریف ثابت، به‌کارگیری کاراکترهای کنترلی، قرار دادن توضیحات در متن برنامه و دستکاری کنندهٔ **endl** روبرو هستید که جزء مورد آخر با تمامی موارد از قبل آشنایی دارید. دستکاری‌کنندهٔ **endl** همانند کاراکتر کنترلی **\n** عمل می‌کند با این تفاوت که نباید در داخل کوتیشن قرارگیرد و همچنین این دستکاری کننده موجب پاک شدن بافر خروجی می‌شود. از این پس در تمامی مثال‌ها برای بهتر ارجاع دادن خوانندگان به خطوط مختلف برنامه، خطوط برنامه‌ها را شماره‌گذاری می‌کنیم، که شماره‌ها جزء برنامه نیستند، و هنگام وارد کردن برنامه در کامپایلر از نوشتن شماره‌ها خودداری کنید.

---

1. space

دستور using

در C++ استاندارد، برنامه‌ها را می‌توان به فضاهای نام مختلفی تقسیم‌بندی کرد. مقصود از فضای نام[۱]، بخشی از برنامه است که در آن، شناسه‌های مشخصی به رسمیت شناخته شده و در خارج آن، این شناسه‌ها به رسمیت شناخته نمی‌شوند. و مقصود از شناسه، اسامی متغیرها، توابع، کلاس‌ها و غیره است. مزیت بکار گیریِ فضاهای نام این است که، حوزه‌هایی در حافظهٔ مورد استفادهٔ برنامه، تشکیل می‌شود و هر حوزه به یک فضای نام اختصاص می‌یابد، تا متغیرها و داده‌های فضای نام مذکور در آن حوزه قرار گیرد در این صورت احتمال برخورد شناسه‌های موجود در حوزه‌های متفاوت با یکدیگر به صفر می‌رسد و از بروز بسیاری از خطاها جلوگیری می‌شود.

پس از استانداردسازی C++ بسیاری از هدرفایل‌ها بازنویسی شدند. هنگامی که می‌خواهید از هدر فایل‌های جدید C++ استفاده کنید باید در هنگام افزودن این هدرفایل‌ها به برنامه، از قرار دادن پسوند **h.** در دنبالهٔ نام فایل سرآیند خودداری کنید. بنابراین به جای استفاده از فایل قدیمی **iostream.h** می‌توانید از هدرفایل جدید **iostream** استفاده کنید. اما اگر برنامهٔ مثال ۳-۷ را یک بار دیگر پس از حذف پسوند **h.** از انتهای فایل سرآیند دوباره کامپایل کنید با خطاهای متعددی برخورد می‌کنید. اولین خطایی که می‌توان مشاهده کرد عبارت است از:

error C2065: 'cout' : undeclared identifier

در حقیقت کامپایلر در پیام فوق به شما می‌گوید که شئ **cout** را شناسایی نکرده است. این عدم شناسایی بدان علت است که پس‌از بازنویسی فایل‌های سرآیند در C++ استاندارد، تمامی شناسه‌های این فایل‌ها در حوزه‌ای با نام **std**[۲] قرار داده شد تا احتمال برخورد شناسه‌های موجود در آنها با سایر قطعات برنامه کم شود. برای استفاده از شناسه های موجود در این فایل‌ها باید قبل از تابع **main** عبارت زیر را تایپ کنید:

```
using namespace std;
```

اگر عبارت فوق را قبل از تابع **main** تایپ کنید و برنامه را دوباره کامپایل و اجرا کنید خواهید دید که دیگر با خطا برخورد نمی‌کنید. در واقع با قرار دادن عبارت فوق قبل از تابع **main** به کامپایلر می‌گویید که شناسه‌هایی نظیر **cout** در این فضای نام قرار دارد. اما می‌توان بدون قرار دادن عبارت فوق نیز کامپایلر را وادار به شناسایی شناسه‌های موجود در فضای نام **std** کرد. برای این منظور باید از *عملگر حوزهٔ تفکیک*[۳] (یعنی **::**) به همراه اسم فضای نام مورد نظر استفاده کنید. به‌عنوان مثال در این حالت خط شمارهٔ ۱۲ مثال ۳-۷ به‌صورت زیر باید بازنویسی شود:

```
std::cout<<"\n\n\tThe circle's area = \t"<<s;
```

---

۱. **Namespace**

۲. **std** مخفف کلمهٔ **standard** است.

۳. **scope resolution**

۳-۷: تبدیل انواع

آیا تا به حال از خود پرسیده‌اید که کامپیوتر چگونه دو عدد را با هم جمع یا تفریق یا ضرب می‌کند؟

جواب این سؤال را باید با توجه به دستورات زبان سطح پایین اسمبلی داد. اگر در زبان اسمبلی [مربوط به پردازنده‌های اینتل] بخواهیم یکی از اعمال چهارگانهٔ ریاضی را بر روی دو عدد صحیح انجام دهیم حتماً باید آن دو عدد با یکدیگر هم اندازه باشند. علت این محدودیت این است که پردازنده برای عملیات برروی دو عدد، ابتدا باید آن دو عدد را در رجیستر[1]های خود قرار دهد، سپس با عمل کردن مدارات خاصی از پردازنده عمل ریاضی موردنظر صورت می‌گیرد. در حقیقت مشکل اصلی از مداراتی است که عمل‌ریاضی موردنظر ما را انجام می‌دهند. در ساختمان بسیاری از پردازنده‌ها اینگونه محدودیت‌ها وجود دارد چرا که طراحی مدارات داخلی آنها چنین محدودیت‌هایی را ایجاب می‌کند. به‌عنوان مثال هیچگاه نمی‌توان دو عدد که در حافظهٔ **RAM** قرار دارند را با هم جمع یا تفریق یا ضرب و یا تقسیم کرد. چرا که مدارات پردازنده‌ها به‌گونه‌ای طراحی شده‌اند که برای عملیات کردن بر روی این دو عدد ابتدا باید حداقل یکی از آن دو عدد را به داخل رجیسترهای پردازنده آورد، تا مدارات قسمت **ALU** بتوانند عملیات مورد نظر ما را انجام دهند. این مثالی کوچکی از محدودیت‌های موجود در کارکردن با پردازنده‌ها است.

به همین صورت مدارات قسمت **ALU** تنها قادر به عملیات برروی دو داده با طول مشابه هستند. مثلاً هر دو عدد باید دارای طولی برابر ۱۶ بیت باشند تا بتوان آنها را در رجیسترهای ۱۶ بیتی پردازنده قرار داد و مدار مربوطه را فراخوانی کرد. حال اگر دو عدد ۳۲ بیتی بودند بایستی از پردازنده‌هایی با طول ۳۲ بیت استفاده کنیم و مدار مربوط به عملیات مورد نظر خود را برای دو رجیستر ۳۲ بیتی فراخوانی کنیم.

اما سؤال بسیار مهمی که در اینجا مطرح می‌شود این است که اگر به‌عنوان مثال بخواهیم دو عدد را که یکی ۱۶ بیتی و دیگری ۳۲ بیتی است را با هم جمع یا ضرب کنیم با توجه به محدودیت‌های موجود در پردازنده‌ها تکلیف چیست؟ آیا باید از چنین اقدامی پرهیز کرد و یا راهی برای این مشکل وجود دارد؟

در پاسخ به این سؤال باید گفت با توجه به آنکه قرار دادن صفر در سمت راست یک عدد هیچ تأثیری بر ارزش آن عدد ندارد، می‌توان عدد ۱۶ بیتی را به عددی ۳۲ بیتی با همان ارزش تبدیل کرد. به‌عنوان مثال دو عدد ۰۰۱۰ و ۱۰ دارای ارزش مشابهی هستند. لذا با توجه به این مثال، می‌توان قاعده‌ای کلی به این صورت عنوان نمود که، در کامپیوتر برای انجام عملیات ریاضی بر روی دو عدد با طول متفاوت، ابتدا باید با قرار دادن صفر در سمت راست عددِ کوچکتر، آن را هم طول عدد بزرگترمی سازد، و سپس قادر خواهد بود عملیات مورد نظر را برروی آن دو عدد انجام دهد. لذا در اکثر زبان‌های برنامه‌نویسی مبحثی تحت عنوان *تبدیل انواع* مطرح می‌شود که در ادامه به تشریح *تبدیل انواع* در **C++** خواهیم پرداخت. شایان ذکر است که تمام زبان‌ها دارای امکاناتی برای شرکت دادن انواع مختلف در عبارات محاسباتی نبوده و زبان **C** یکی از اولین زبان‌هایی بود که این امکان را فراهم نمود.

---

1. register

تبدیل نوع در C++ بر دو گونه است:

۱. تبدیل نوع اتوماتیک که بیشتر در پردازش عبارات محاسباتی و احکام انتساب رخ می‌دهد و به جهت آنکه برنامه نویس هیچ نقشی در انجام این تبدیلات ندارد به آنها تبدیل نوع اتوماتیک گفته‌می‌شود.

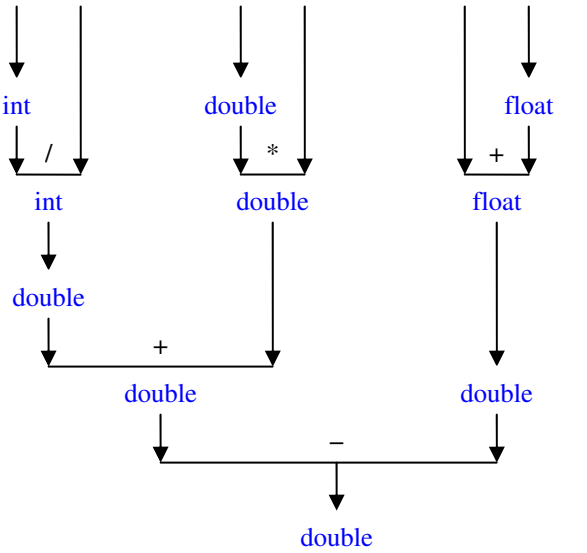
۲. تبدیل نوع موقت که خود دارای حالت‌های متفاوتی است و در ادامه به بررسی آن خواهیم پرداخت.

تبدیل نوع اتوماتیک (تبدیل نوع ضمنی)

در زبان C++ شما قادر هستید انواع مختلف موجود در C++ را بدون هیچ محدودیتی در عبارت محاسباتی به‌کارگیرید. به‌عنوان مثال شما می‌توانید یک عدد از نوع **int** را با یک عدد از نوع **float** جمع کنید. چنانکه پیشتر عنوان شد C++ برای محاسبهٔ حاصل چنین عبارت محاسباتی، ابتدا نوع کوچکتر (**int**) را به نوع بزرگتر (**float**) تبدیل و سپس دو عدد **float** به‌دست آمده را با یک دیگر جمع می‌کند. بنابراین در تبدیل نوع اتوماتیک که در عبارات محاسباتی رخ می‌دهد، در صورتی که دو عملوند اطراف یک عملگر حسابی دارای انواع متفاوتی باشند، ابتدا نوع کوچک‌تر به نوع بزرگتر تبدیل می‌شود، و سپس عملگر برای نوع بزرگتر فراخوانی می‌گردد.

مثال ۳-۷: در قطعه کد زیر متغیر **result** باید از چه نوعی باشد تا حاصل عبارت محاسباتی سمت راست دستور انتساب بدون سرریز در این متغیر ذخیره گردد؟ مراحل پردازش و تبدیل انواع را به‌صورت نموداری رسم کنید؟

```
1.  char   ch;
2.  int     i;
3.  float   f;
4.  double d;
5.  result = (ch / i)   +   (f * d)   -   (f + i) ;
```



شرح مثال: در عبارت محاسباتی فوق پردازش از اولین پرانتز آغاز می‌گردد. چون متغیر **ch** دارای طول یک بایت و متغیر **i** دارای طولی برابر چهار بایت است. ابتدا متغیر **ch** به نوع صحیح تبدیل می‌گردد و سپس عمل تقسیم صورت

می‌گیرد، نتیجهٔ این تقسیم هر چه باشد عددی هم نوع عملوندهای خود، یعنی از نوع صحیح خواهد بود. اگر روند نمودار رسم شده را به همین روش دنبال کنید خواهید دید که در پایان حاصل عبارت محاسباتی از نوع **double** می‌باشد، بنابراین متغیر **result** نیز باید از این نوع باشد.

حالت دیگری از تبدیل نوع اتوماتیک، زمانی رخ می‌دهد که در احکام انتساب دو نوع متفاوت به یکدیگر نسبت داده شوند. به‌عنوان مثال قطعه کد زیر را در نظر بگیرید:

```
short   i=12654;
char    ch='k';
float   f=3.14;
ch=i;
i=f;
f=ch;
f=i;
```

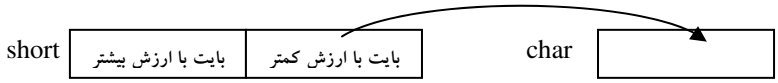
تمامی انواع موجود در **C++** را می‌توان توسط احکام انتساب به یکدیگر نسبت داد. اما دو حالت متفاوت در تبدیل نوع در احکام انتساب وجود دارد که به شرح زیر است:

۱. اگر نوعی با طول کوچکتر به نوعی با طول بزرگتر نسبت داده شود، در این حالت تبدیل نوع معمولاً بدون هیچگونه مشکلی صورت می‌گیرد. زیرا کامپایلر مانند حالت قبل به راحتی با قرار دادن صفر در سمت چپ نوع کوچکتر آن را به نوع بزرگتر تبدیل می‌کند. نمونه‌ای از این تبدیل نوع را در خطوط ۶ و ۷ مثال فوق مشاهده می‌کنید. اما کامپایلر برای برخی از این دست تبدیل نوع‌ها یک هشدار صادر می‌کند. مثلاً در تبدیل نوع **int** به **float** کامپایلر هشداری صادر می‌کند. به نظر شما چنین هشداری به چه دلیلی صادر می‌شود؟

۲. اگر نوعی با طول بزرگتر به نوعی با طول کوچکتر نسبت داده شود، کامپایلر یک پیغام هشدار تحت این مضمون صادر می‌کند که، *"تبدیل کردن نوع بزرگتر به نوع کوچکتر ممکن است همراه با از دست رفتن اطلاعات باشد"*. به‌عنوان مثال اگر خط پنجم از مثال فوق را داخل یک تابع **main** بنویسید و کد خود را کامپایل کنید هشدار زیر را دریافت می کنید:

warning C4244: '=' : conversion from 'float' to 'int', possible loss of data

اما کامپایلر علی‌رغم هشداری که صادر می‌کند این تبدیل را انجام داده و حکم انتساب بدون هیچ مشکلی اجرا می‌گردد. فقط چنانکه گفته شد تبدیل از نوعی با طول بزرگتر به نوعی با طول کوچکتر، ممکن است با از دست رفتن اطلاعات همراه باشد. به‌عنوان مثال خط چهارم قطعه کد فوق را در نظر بگیرید. در این خط متغیری از نوع **short** با طول ۲ بایت به متغیری از نوع **char** با طول ۱ بایت نسبت داده شده است. در این حالت داده‌های موجود در بایت کم ارزش نوع **short** به داخل نوع **char** ریخته می‌شود و بایت با ارزش رهامی‌گردد. حال اگر دادهٔ موجود در نوع **short** بقدری بزرگ باشد که از بایت کم ارزش فراتر رود و اطلاعاتی نیز در بایت با ارزش وجود داشته باشد، آن اطلاعات از بین خواهد رفت.



شکل ۳-۸ : نحوهٔ تبدیل اتوماتیک از نوع بزرگتر به نوع کوچکتر

در جدول ۳-۹ ، تبدیل انواع مختلف به یکدیگر، و اطلاعاتی که ممکن است از بین برود لیست شده است.

نوع منبع	نوع مقصد	اطلاعاتی که ممکن است از بین برود
char	signed char	اگر مقدارx بیش از ۱۲۷ باشد، در مقصد عددی منفی ذخیره می‌گردد. این عدد لزوماً (x-) نیست.
short int	char	۸ بیت با ارزش از بین می‌رود.
long int	char	۲۴ بیت با ارزش از بین می‌رود.
long int	float	اگر طول عدد موجود در مبدأ بیش از ۷ رقم اعشار باشد تبدیل نوع با از دست رفتن اطلاعات همراه است.
long int	short int	۱۶ بیت با ارزش از بین می‌رود.
float	long int	بخش کسری از بین می‌رود.
float	short int	نه تنها بخش کسری بلکه مقداری از قسمت صحیح نیز از بین می‌رود.
double	float	دقت علائم کم می‌شود و نتیجه گرد می‌گردد.
long double	double , int	دقت علائم کم می‌شود و نتیجه گرد می‌گردد.

جدول ۳-۹ : تبدیل انواع در احکام انتساب

شما می توانید در جدول فوق انواع مشابه را نیز جـایگزین کنید و به جـدول کامل تری برسید. به‌عـنوان مثال می‌توان در ردیف دوم، تبدیل نوع **int16__** به نوع **char** را نیز در نظر بگیرید. اما ما به جهت صرفه‌جویی از نوشتن موارد تکراری خودداری کرده‌ایم.

تبدیل نوع موقت (تبدیل نوع صریح)

قبل از ادامهٔ بحث در خصوص تبدیل نوع موقت از شما می‌خواهیم که کار در کلاس زیر را انجام دهید.

کار در کلاس ۳-۲ :

برنامه ای به زبان **C++** *بنویسید که سه عدد را از کاربر دریافت کند، سپس معدل آن سه عدد را چاپ کند.*

احتمالاً پاسخی که شما برای کاردرکلاس ۳-۱ داده‌اید چیزی شبیه برنامهٔ زیر است.

```
1.  //This program calculates average of 3 integer numbers.
2.  #include <iostream.h>
3.  int main ()
4.  {
5.    int x,y,z;
6.     cout<< "Please enter 3 numbers , this "
7.         <<"program calculate average of these numbers:\n";
8.        cout<<"\n\tEnter X=";
9.        cin>>x;
10.         cout<<"\n\tEnter Y=";
11.         cin>>y;
12.           cout<<"\n\tEnter Z=";
13.           cin>>z;
14.       float avg=(x+y+z)/3;
15.        cout<<"\nThe average of these numbers is: "<<avg;
16.     return 0;
17. }
```

برنامهٔ فوق از نظر منطق هیچ مشکلی ندارد. اما اگر این برنامه را در یک کامپایلر بنویسید و اجرا کنید انتظار دارید به‌عنوان مثال برای سه ورودی ۱۵ و ۱۹ و ۱۰ معدل را برابر ۱۴/۶۶۶۷ نشان دهد. اما چیزی که مشاهده خواهید کرد تنها عدد صحیح ۱۴ است. اگر برای هر گونه ورودی دیگری نیز آزمایش خود را تکرار کنید ورودی شما یک عدد صحیح خواهد بود در حالی که شما انتظار دارید جواب شما قسمت اعشاری نیز داشته باشد. به نظر شما اشکال این برنامه در کجاست و چه راه‌حلی برای اصلاح این مشکل وجود دارد؟ قبل از این که پاسخ این پرسش را بدهیم باید مطالبی را در خصوص تبدیل نوع موقت بیان کنیم.

بحث خود را با این سؤال آغاز می‌کنیم که تبدیل نوع موقت چیست و به چه منظوری به‌کار می‌رود؟ اصطلاح تبدیل نوع موقت برای تبدیلاتی به‌کار می‌رود که به‌طور صریح توسط برنامه‌نویس مشخص می‌گردد. به دیگر سخن هرگاه کامپایلر قادر به انجام تبدیل نوع به‌صورت اتوماتیک نباشد، برنامه‌نویس باید خود این تبدیل نوع را انجام دهد که به آن تبدیل نوع صریح یا موقت گفته می‌شود، و در اصطلاح برنامه‌نویس برای این منظور باید از یک **cast** استفاده کند. در **C++** استاندارد، چند مدل تبدیل نوع موقت امکانپذیر است که از جمله می‌توان تبدیل موقت استاتیک[1]، تبدیل موقت داینامیک[2](پویا)، تبدیل موقت تفسیری[3]، و تبدیل موقت ثابت[4] را نام برد. در این بخش تنها به تشریح تبدیل نوع استاتیک خواهیم پرداخت و دیگر انواع تبدیل نوع موقت را در فصول آتی مطرح می‌سازیم.

1. static_cast
2.dynamic_cast
3.reinterpret_cast
4. const_cast

اما به بحث قبلی خود باز گردیم، و مشکل برنامهٔ صفحهٔ قبل را برطرف کنیم. چنانکه پیشتر گفته شد برای انجام اعمال ریاضی برروی دو عملوند باید این دو عملوند از نظر نوع و اندازه با یکدیگر برابر باشند تا پردازنده قادر به انجام عمل ریاضی مورد نظر باشد. از طرفی در عملگر تقسیم هرگاه دو عملوند مربوط به عملگر تقسیم از نوع صحیح باشند، عملیات تقسیم نیز به‌صورت تقسیم صحیح صورت خواهد گرفت. لذا با توجه به آنکه هر دو عملوند عملیات تقسیم در خط ۱۴ از نوع صحیح است تقسیم صورت گرفته نیز از نوع صحیح خواهد بود. اما اگر به شکلی بتوانیم یکی از این دو عملوند را به نوع اعشاری تبدیل کنیم عملگر دیگر به‌طور اتوماتیک به نوع اعشاری تبدیل می‌گردد و در نتیجه به جای آنکه تقسیم صحیح صورت بگیرد تقسیم اعشاری صورت می‌گیرد و قسمت اعشاری پاسخ نیز به‌دست می‌آید. برای این منظور می‌توان به وسیلهٔ تبدیل نوع استاتیک یکی از عملوندها را به‌طور موقت به نوع اعشاری تبدیل کنیم. برای این منظور خط ۱۴ کد صفحهٔ قبل را به‌صورت زیر بازنویسی کنید و دوباره آن را کامپایل و اجرا نمایید. این بار چه نتایجی را در خروجی می‌بینید؟

```
14. float avg=static_cast <float> (x+y+z)/3;
```

در حقیقت با تغییرات فوق کامپیوتر را مجبور می‌سازیم حاصل عبارت داخل پرانتز را به‌طور موقت به نوع **float** تبدیل کند. سپس به‌طور اتوماتیک عدد صحیح **3** نیز به نوع اعشاری تبدیل شده و عملیات تقسیم به‌صورت اعشاری انجام می‌پذیرد. در این صورت نتیجهٔ مورد انتظار را مشاهده خواهید کرد. البته با توجه به مباحث قبلی بهتر است به جای نوع **float** از نوع **double** استفاده کنیم تا یک تبدیل نوع ایمن انجام دهیم.

همیشه در هنگام استفاده از تبدیل نوع استاتیک باید نوع جدید را بین دو علامت < > قرار دهید. همچنین باید متغیر یا عبارتی را که می‌خواهید تبدیل نوع پیدا کند در داخل پرانتز قرار دهید.

کار در کلاس ۳-۳ :

در برنامهٔ زیر با نمونهٔ دیگری از تبدیل نوع استاتیک آشنا خواهید شد. به نظر شما خروجی برنامهٔ زیر چیست؟

```
1.  //This program is a test for signed and unsigned int.
2.  #include <iostream>
3.  using  namespace std;
4.  int main()
5.  {
6.     int int_var=1500000000;              //1,500,000,000
7.      cout<<"\nInt_var1 = "<<int_var;
8.       int_var=(int_var*10)/10;
9.        cout<<"\nInt_var2 = "<<int_var;     //wrong answer
10.      int_var=1500000000;               //1,500,000,000
11.       int_var=(static_cast <double> (int_var)*10)/10;
12.        cout<<"\nInt_var3 = "<<int_var; //right  answer
13.      cout<<endl;
14.    return 0;
15. }
```

اگر برنامهٔ موجود در کاردرکلاس ۳-۲ را در کامپایلر **VC6** کامپایل و اجرا کنید خروجی زیر را مشـاهده خواهیـد کرد:

```
Int_var1 = 1500000000
Int_var2 = 211509811
Int_var3 = 1500000000
```

در خروجی فوق همه چیز قابل پیش‌بینی بود جزء دومین خط خروجی. چرا که ما انتظار داشتیم عدد ۱.۵۰۰.۰۰۰.۰۰۰ چاپ گردد اما متأسفانه خروجی اشتباهی چاپ شده است. علت این امر این است که در خط نهم برنامه با ضرب کردن متغیر **int_var** در عدد ده مقدار آن از محدودهٔ اعداد **int** فراتر می‌رود و در نتیجه منجر به خروجی اشتباه شده است. اما در خط دوازدهم برنامه ابتدا با یک تغییر نوع استاتیک متغیر **int_var** را به‌طور موقت به نوع **double** تبدیل می‌کنیم که قادر است عدد ۱۵.۰۰۰.۰۰۰.۰۰۰ را در خود بگنجاند و در نتیجه خروجی مورد انتظار ما چاپ شده است.

قبل از **C++** استاندارد، تبدیل نوع موقت با استفاده از قالب متفاوتی انجام می‌شد. مثلاً به‌جای نوشتن عبارت:

```
float_var=static_cast <float> (int_var);
```

عبارت:

```
float_var=(float)int_var;
```

و یا عبارت زیر:

```
float_var=float(int_var);
```

نوشته می‌شد. یکی از مزایای استفاده از فرم اول مشاهدهٔ آسان آن در متن یک کد بزرگ است. بنابراین جستجوی آن نیز به‌سادگی صورت می‌گیرد. البته این فرم از تبدیل نوع استاتیک، نیز در **C++** استاندارد قابل قبول است. استفاده از تبدیل نوع موقت امنیت داده را به خطر می‌اندازد. چرا که گاهی منجر به خطاهایی می‌شود که کامپایلر قادر به کشف آن خطاها نیست. بنابراین بهتر است جزء در موارد ضروری از تبدیلات موقت استفاده نکنید.

۳-۸ : نحوهٔ ایجاد فایل اجرایی توسط کامپایلر

قبل از اینکه به نحوهٔ ایجاد فایل اجرایی توسط کامپایلر اشاره کنیم باید اشاره‌ای کوتاه به مقولهٔ توابع و فایل‌های کتابخانه‌ای در C++ بکنیم.

توابع کتابخانه‌ای و فایل‌های کتابخانه‌ای

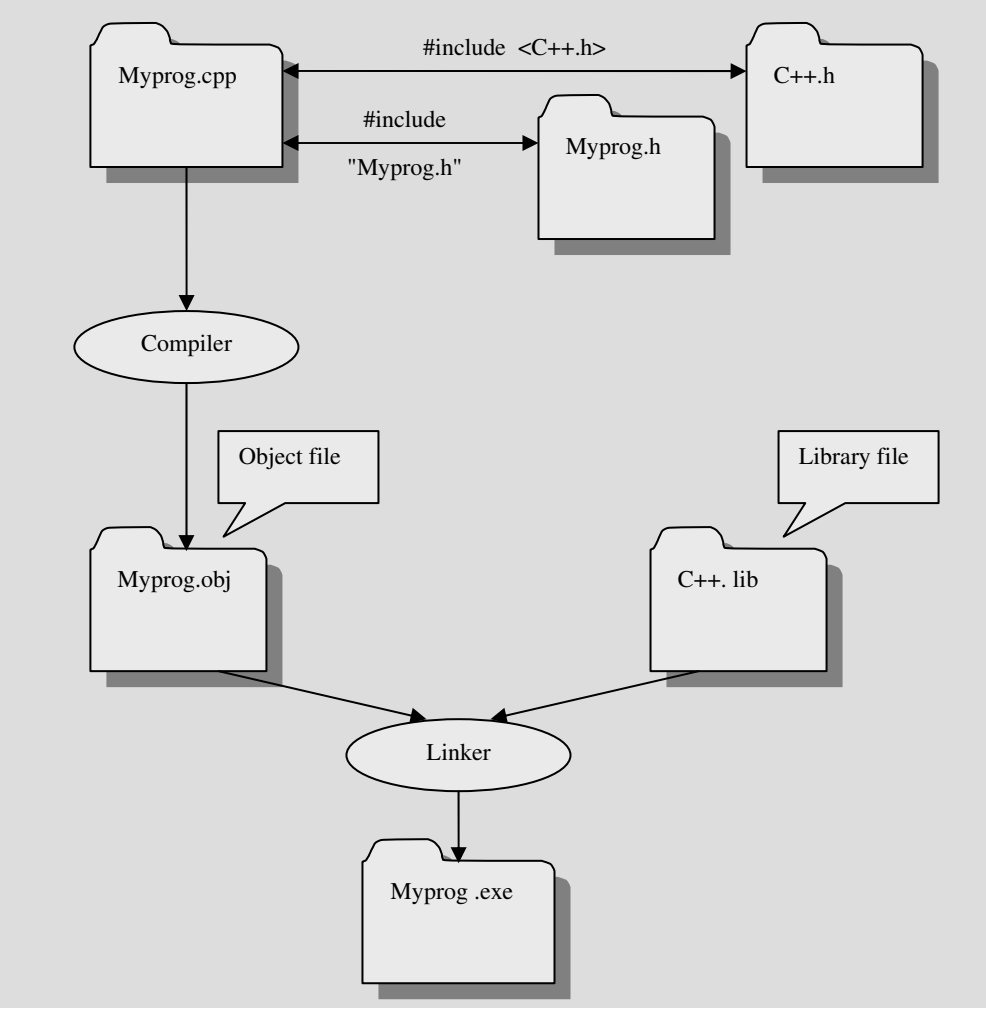
در زبان C++ برای بسیاری از کارها توابعی به‌صورت آماده نوشته شده است که شما می‌توانید از آنها در برنامه‌های خود استفاده کنید. به‌عنوان مثال برای محاسبهٔ جذر تابعی تحت‌عنوان **sqrt** نوشته شده است. این توابع در فایل‌هایی موسوم به فایل‌های کتابخانه‌ای قرار دارند. این فایل‌ها حاوی برنامهٔ اجرایی به زبان ماشین برای توابع کتابخانه‌ای هستند. فایل‌های کتابخانه‌ای دارای پسوند **lib.** هستند و هرکس می‌تواند آنها را تولید کند و در دسترس عموم قراردهد. اما برای استفاده از توابع کتابخانه‌ای باید به کامپایلر مشخصات فایل کتابخانه‌ای حاوی تابع موردنظر خود را ارائه کنیم تا کد ماشین داخل توابع به کد برنامهٔ ما پیوند زده شود. برای این منظور اطلاعات لازم در خصوص توابع کتابخانه‌ای در هدر فایل‌های خاصی قرار گرفته است. به‌عنوان مثال اطلاعات لازم در خصوص فایل کتابخانه‌ای که شامل برنامهٔ اجرایی تابع **sqrt** است در هدرفایل **cmath** قرار دارد، و یا اطلاعات مربوط به توابع کتابخانه‌ای **scanf** و **printf** در سرفایل **stdio.h** قرار دارد. هنگام استفاده از توابع کتابخانه‌ای حتماً باید سرفایل مربوطه به برنامه اضافه شود. همان‌طور که قبلاً دیدید به واسطهٔ دستور **#include** می‌توان انواع هدرفایل‌ها را به برنامه اضافه کرد.

همچنین در آینده خواهید دید که خود قادر به تولید هدرفایل هستید و می‌توانید کلاس‌ها اشیاء و توابع موردنظر خود را در هدر فایل‌هایی تولید و از آنها در برنامه‌های گوناگون استفاده کنید. اگر هدرفایلی را خود نوشته باشید به‌جای استفاده از دو نماد < > در اطراف نام هدرفایل هنگام افزودن آن به برنامه باید از علامت کوتیشن در اطراف نام هدرفایل استفاده کنید. البته می‌توانید از علامت کوتیشن برای تمامی سرفایل‌ها بهره بگیرید.

نحوهٔ تولید فایل اجرایی یک برنامه توسط کامپایلر

پس از آنکه برنامهٔ خود را به‌طور کامل در یک ویرایشگر مثل **Notepad** و یا ویرایشگر خود کامپایلر زبان C++ تایپ کردید باید پسوند این فایل متنی را **cpp.** بنامید. هنگامی که فرمان کامپایل را به کامپایلر می‌دهید کامپایلر ابتدا کلیهٔ هدر فایل‌هایی را که به برنامهٔ خود اضافه کرده‌اید، چه آنهایی که مربوط به خود زبان C++ هستند و چه آنهایی که شما نوشته‌اید را به‌جای دستور **#include** مورد نظر قرار می‌دهد، اما شما تغییری در متن فایل **cpp.** خود نمی‌بینید چرا که این تغییرات در فایل‌های واسط داده می‌شود. سپس فرآیند کامپایل آغاز شده و ابتدا اشکالات نحوی برنامه گرفته می‌شود. اگر خطایی از این نظر پیدا شود دارای عنوان **syntax error** خواهد بود. فراموش نکنید در صورت وجود خطا همواره باید اوّلین خطا را برطرف سازید، سپس دوباره فرآیند کامپایل را از نو اجرا کنید. همواره ممکن است یک خطا موجب اعلام تعدادی خطای نامعتبر دیگر شود در نتیجه همواره اولین خطا برای ما دارای ارزش است، و در اکثر مواقع با رفع کردن اولین خطا و کامپایل مجدد تعداد خطاها به نحو چشم‌گیری کاهش می‌یابد. پس از این مرحله اشکالات منطقی برنامه گرفته می‌شود. به خطاهایی که در این زمان توسط کامپایلر اعلام می‌شود خطای زمان ترجمه(کامپایل) گفته

می‌شود. اگر برنامه از مرحلهٔ کامپایل سربلند(بدون خطا) بیرون آید یک فایل با پسوند **obj.** ساخته می‌شود. سپس باید فرآیند ساخت فایل اجرایی را آغاز کنید. در این مرحله قسمتی از زبان **C++** موسوم به پیوند دهنده[1] شروع به‌کار می‌کند و فایل‌های کتابخانه‌ای لازم را که دارای پسوند **lib.** هستند به کد برنامه پیوند می‌زند. نتیجهٔ این عملیات فایل اجرایی برنامه با پسوند **exe.** است که قابلیت اجرا شدن بر روی کامپیوتر را دارد. فایل **exe** شامل کد برنامه به‌صورت صفر و یک است. در شکل ۳-۹ تمامی این فرآیند را به‌صورت نمودار می‌توانید مشاهده کنید.



شکل ۳-۹ : مراحل ایجاد فایل اجرایی

1. linker

۳-۹: تمرین

۱. تعیین کنید کدام یک از عبارات زیر درست و کدام یک نادرست هستند.

الف- توضیحات در برنامه، موجب می‌شود که متن بعد از // در صفحه نمایش چاپ شود.

ب- تمام متغییرها قبل از استفاده باید اعلان شوند.

ج- از نظر C++ ، متغییرهای **number** و **Number** یکی هستند.

د- تبدیل نوع **int** به **double** و یا **float** ممکن است با از دست رفتن اطلاعات همراه باشد.

و- اعلان متغییرها تنها باید در ابتدای تابع اصلی صورت گیرد.

هـ- عملگر باقیماندهٔ تقسیم، تنها می‌تواند با عملوندهایی از نوع صحیح به‌کار گرفته شود.

۲. در مورد شئ **cerr** تحقیق کنید.

۳. هر یک از خطوط زیر دارای چه اشکالات نحوی (**syntax error**) هستند. (درهر خط ۲ اشکال وجود دارد.)

```
1.  cout>>this is my first program>>'\n';
2.  cin>>67> >number;
3.  int x=87; float f=56.7; x =< f++ ;
4.  float f=45.6,g=0; f%=g;
```

۴. برنامه ای به زبان C++ بنویسید که شعاع یک دایره را بگیرد و قطر و مساحت و محیط آن را چاپ کند.

۵. در مورد خطاهای **fatal error** و خطاهای **nonfatal error** تحقیق کنید.

۶. برنامه‌ای به زبان C++ بنویسید که یک عدد پنج رقمی را بگیرد و تک تک ارقام آن را به‌دست آورد و با فاصلهٔ ۸ کاراکتر از هم چاپ کند. مثلاً عدد **98524** را در یافت کند و خروجی زیر را بدهد:

9 8 5 2 4

۷. ضرورت وجود توضیحات در برنامه را بیان کنید.

۸. تبدیل انواع چیست؟ با ذکر مثال‌هایی تبدیل انواع را به‌طور کامل شرح دهید.

۹. تشریح کنید در چه زمانی ممکن است در تبدیل انواع اطلاعات از بین برود؟

۱۰. چگونگی تولید فایل اجرایی را از یک فایل متنی **cpp.** توضیح دهید.

۱۱. با توجه به مقادیر تعیین شده هریک از عبارات زیر را ارزیابی کنید و مقادیر **k** و **L** را به‌دست آورید.

```
int    x=8,y=10,m=6;
int    k=x/4*y/2*m;
int    L=x/y+ ++y/--m;
```

۱۲. تحقیق کنید که در چه مواردی ممکن است خطای **link** به وجود آید.

۱۳. برنامه‌ای بنویسید که وزن کالا را برحسب گرم بخواند و وزن آن را برحسب کیلوگرم چاپ کند.

۱۴. برنامه‌ای بنویسید که صورت و مخرج دو کسر را بخواند و حاصل جمع و تفریق و ضرب و تقسیم دو کسر را در خروجی بنویسد.

۱۵.در موارد زیر برنامه‌های داده شده را از روی لوح فشردهٔ شمارهٔ یک پیدا کرده و در کامپایلر خود اجرا کنید و خروجی آن را ببینید. سپس در مورد تک تک خطوط هر برنامه توضیح دهید.

I.

```
#include "iostream.h"
int main()
{
    cout<<65<<endl;
     cout<<'A';
      cout<<'\n'<<(int)'A'<<"\a\n";
 return  0 ;
}
```

II.

```
#include <iostream>
using namespace std;
void main()
{
    int m,n;
      m=(n=66)+9;
        cout<<m<<","<<n<<endl;
}
```

III.

```
#include <iostream.h>
#include <limits.h>
int main()
{
 //prints the contents stored in limits.h
 cout<<"\n minimum char = "<<CHAR_MIN;
  cout<<"\n maximum char = "<<CHAR_MAX;
   cout<<"\n minimum short = "<<SHRT_MIN;
    cout<<"\n maximum short = "<<SHRT_MAX;
     cout<<"\n minimum int  = "<<INT_MIN;
      cout<<"\n maximum int  = "<<INT_MAX;
       cout<<"\n minimum long = "<<LONG_MIN;
        cout<<"\n maximum long = "<<LONG_MAX;
  return 0;
}
```

IV.

```
#include "iostream"
using namespace std;
int main()
{
 int x=6,y=8;
 cout<<"x="<<x<<",y="<<y<<endl;
  cout<<"Squared hypotenuse is:"<<x*x+y*y<<endl;
   cout<<"(6/8)*8 is equal to "<<(6/8)*8<<endl ;
 return 0;
}
```

۳-۱۰ : موارد مطالعاتی (Case Studys)

۱. در متن لاتین زیر نحوهٔ کامپایل برنامه‌های C++ از طریق خط فرمان توضیح داده شده است. ضمن مطالعهٔ این متن، مراحل گفته شده در آن را به‌عنوان یک آزمایش اجراکرده و گزارشی از آن تهیه‌کنید.

Command Line Mode:

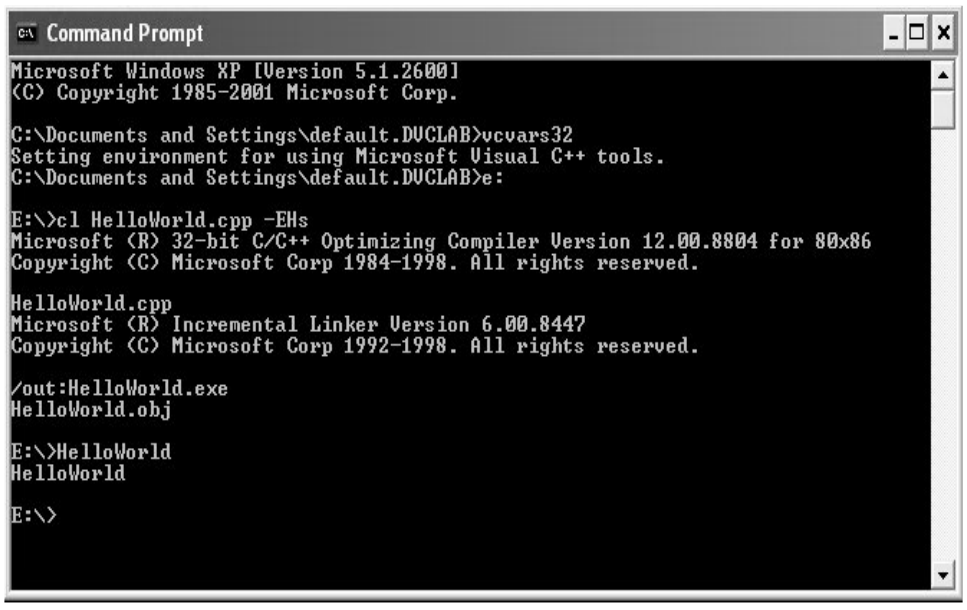
In order to use command line compiling, first copy this file:
C:\Program Files\Microsoft Visual Studio\VC98\bin\vcvars32.bat
to your *C:\WINDOWS* folder. This has already been done on the computer lab PCs.
Use any text editor of your choosing, such as XP's Notepad, or JNotePad. Then when you go to a command prompt to compile and build, enter the command *vcvars32* once to prepare for using the command line editor. When saving CPP files from Notepad, put the filename in quotes to prevent Windows from adding .txt to the filename. *To compile*, use a command like the following (the command is "see-el", *not* "see-one") to create an EXE file:

```
cl HelloWorld.cpp -EHs
```

To compile and build projects consisting of *more than one CPP*, list the CPPs separated by spaces, like this:

```
cl main.cpp Time.cpp -EHs
```

The EXE has the same name as the first (or only) listed CPP, but with the extension "exe". *To run* the program, enter the name of the EXE on the command line -- you may leave off the trailing ".exe".



شکل ۳-۱۰ : نمونه‌ای از کامپایل از طریق خط فرمان

To compile a CPP *without building*, include the `-c` flag -- this produces an OBJ file with the same name as the listed CPP, but with the extension "obj":

```
cl Time.cpp -c -EHs
```

To build an EXE from already-compiled OBJs, list the OBJs and do not use the `-EHs` flag, like this (creating *main.exe*):

```
cl main.obj Time.obj
```

When working with multiple CPP files in a single project, it is recommended to compile each CPP separately, using the `-c` flag during development. This makes debugging easier. Once the program is working, and you are making small code adjustments, then you should go back to compiling and building all in one command.

Using XP's Command Line Buffer

So that you do not have to retype the compile and run commands, use the up and down arrow keys to navigate through previously-typed commands. Use the F7 key to popup a list of commands in the buffer. The usual sequence is to type the compile and build command, followed by the run command. After that, *up–up* returns to the compile and build command, and *down* goes from there to the run command.

۲. در خـصوص نحوهٔ ورود و خروج اطلاعات در زبان **C** که توسط توابع **scanf** و **printf** صورت می‌گیرد تحقیق کنید.

۳. اگر سعی کنید با شئ **cout** متغیری که از نوع **__int64** است را به خروجی ببرید خواهید دید که کامپایلر پیغام خطایی را به شما اعلام می‌کند، این یکی از اشکالات یا به اصطلاح **bug** های موجود در کامپایلر **VC6** می‌باشد که در نسخه‌های بعدی اصلاح شد. اما روشی وجود دارد که می‌توانید این نوع متغیر را به خروجی ببرید. در شبکهٔ جهانی اینترنت در این خصوص تحقیق کنید.

۴. در برخی سیستم‌ها و صفحه کلیدهای قدیمی برخی از کاراکترهای خاص مثل } یا { یا # یا & موجود نبوده است. لذا یک مجموعه کاراکتر به نام **Trigraph** (سه کاراکتر دنبال هم) مخصوص زبان **C** و **Digraph** (دو کاراکتر دنبال هم) مخصوص زبان **C++** طراحی شدند که این نقیصه را برطرف کنند. مثلاً می‌توان به‌جای علامت } از علامت >?? و یا به جای { از علامت <?? استفاده کرد. و یا آنکه می‌توان به جای علامت & از کلمهٔ **bitand** و به جای علامت && از کلمهٔ **and** بهره گرفت. در مستندات **MSDN** و یا اینترنت در این خصوص تحقیق کنید. البته با توجه به آنکه استفاده از این علائم از خوانایی برنامه می‌کاهد و به علاوه اینگونه مشکلات در صفحه کلیدهای جدید برطرف شده است اکثر کامپایلرهای **C++** از کاراکترها و کلمات رزروی تعریف شده در جدول **Digraph** پشتیبانی نمی‌کنند، و تنها از جدول **Trigraph** و آن هم تنها با هدف پشتیبانی کردن از تمامی امکانات زبان **C** پشتیبانی می‌کنند. شایان ذکر است که کلمات تعریف شده در جدول **Digraph** جزء کلمات کلیدی زبان **C++** محسوب نمی‌شوند.

۵. بد نیست بدانید که اگر در تبدیل نوع، نوعی با تعداد بایت کمتر به نوعی با تعداد بایت بیشتر تبدیل شود به آن تبدیل نوع ارتقاء دهنده[1] و در حالتی که نوعی با تعداد بایت بیشتر به نوعی با تعداد بایت کمتر تبدیل می‌شود به آن تبدیل نوع محدود کننده[2] گفته می‌شود.

۶. تحقیق کنید که ساختار یک متغیر **double** و یا **long double** چگونه است؟ و تشریح کنید که چرا با توجه به این ساختار به‌کارگیری کلمهٔ **unsigned** همراه این دو نوع موجب ایجاد نوع بدون علامت؛ و گسترش بازهٔ اعداد مثبت قابل ذخیره در متغیرهای اعشاری دقت مضاعف نمی‌شود.

۷. در C++ ضمن انجام عمل انتساب، عملگر انتساب مقداری را که در عملگر سمت چپ خود جایگزین می‌کند، باز می‌گرداند. در فصل پنجم خواهید دید که چگونه می‌توان از یک عملیات انتساب در داخل عبارات شرطی استفاده کرد.

۸. در مورد این که چگونه اعمال اصلی ریاضی برروی دو عدد ممیز شناور در کامپیوتر اعمال می‌شود تحقیق کنید؟

۹. به خاطر دارید که دقت متغیرهای **float** تا هفت رقم اعشار است. حال فرض کنید که می‌خواهیم حاصل جمع یازده عدد را به خروجی ببریم. از این یازده عدد یکی برابر 10^7 و ده تای دیگر برابر ۱ می‌باشند. سمیرا و سمیه هر یک دو روش متفاوت را برای یافتن این حاصل جمع به کار برده‌اند. سمیرا هر بار یکی از یک‌ها را به عدد بزرگتر که در ابتدا 10^7 می‌باشد می‌افزاید و در نهایت پس از انجام این عمل به تعداد ده مرتبه مقدار حاصل جمع را به خروجی می‌برد. اما سمیه ابتدا ده تا ۱ را با یکدیگر جمع می‌کند و سپس عدد ۱۰ را با عدد 10^7 جمع می‌کند و نتیجه را به خروجی می‌برد. به نظر شما روش کدام یک نتیجهٔ صحیح‌تری را حاصل می‌کند. از این بحث چه نتیجه‌ای گرفتید؟

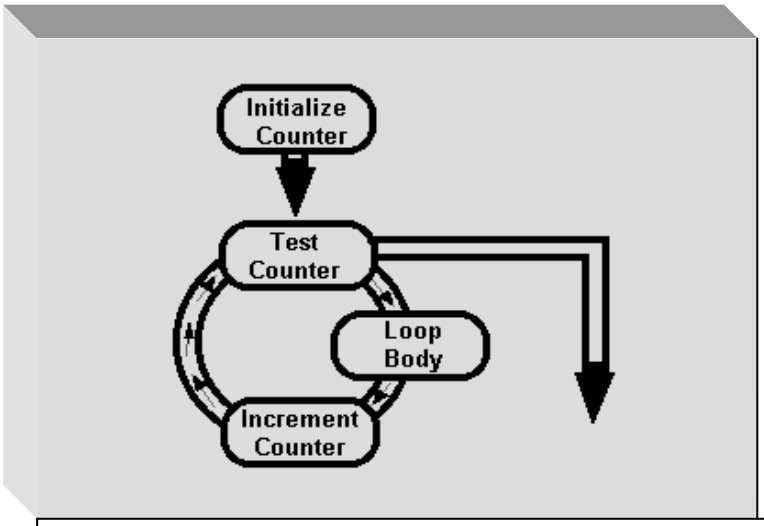
راهنمایی : برنامه‌ای به زبان C++ برای روش سمیرا و سمیه بنویسید و سپس نتایج را با یکدیگر مقایسه کنید.

۱۰. در خصوص گلوگاه وان نویمان[3] که عاملی برای کاهش سرعت در معماری وان نویمان است تحقیق کنید.

۱۱. در مورد انواع روش‌های ترجمه و تولید فایل اجرایی توسط کامپایلرهای متفاوت مثل روش پیاده‌سازی کامپایلری، یا روش تفسیر محض و استفاده از مفسر[4] و یا روش سیستم‌های پیاده‌سازی مختلط[5] تحقیق کنید؟ همچنین از استاد خود بخواهید در خصوص ماشین زمان اجرا مثل ماشین زمان اجرای(مجازی) جاوا (Java Virtual Machine) اندکی اطلاعات در اختیار شما قرار دهد.

---

1. **widening**
2. **narrowing**
3. **Von Neumann bottleneck**
4. **interpreter**
5. **hybrid implementation system**



فصل چهارم

ساختارهای حلقه‌زنی

اهداف فصل و چکیده مطالب :

۱. مهمترین ویژگی الگوریتم‌های ساخت‌یافته
۲. آشنایی با انواع جعبه های تکرار
۳. مثال‌های کاربردی از به‌کارگیری جعبه‌های تکرار
۴. آشنایی با حلقه های تودرتو

۴-۱ : الگوریتم های ساخت یافته

مقدمه

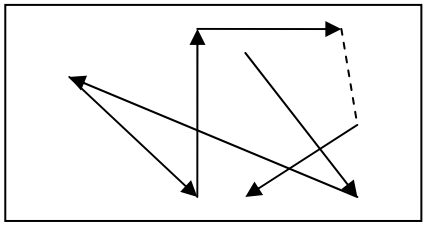
در فصل دوم با انواع جعبهٔ تصمیم و چگونگی ایجاد حلقه به واسطهٔ این جعبه‌ها آشنا شدیم. اما به جهت اهمیت حلقه زنی در الگوریتم‌ها در این فصل چند نماد جدید را برای ایجاد انواع حلقه، معرفی می‌کنیم. اهمیت این ساختارها زمانی مشخص می‌شود که بخواهیم یک الگوریتم را از جهت ساخت یافتگی تحلیل کنیم، و یا بخواهیم از حلقه‌های تودرتو استفاده کنیم. لذا ابتدا به بررسی مهمترین ویژگی الگوریتم‌های ساخت‌یافته می‌پردازیم، تا به جایگاه این ساختارهای حلقه زنی پی‌ببریم. حال این سؤال را مـطرح می‌سازیم که به چه الگوریتم‌هایی ساخت‌یافته و به چه الگوریتم‌هایی غیرساخت یافته گفـته می‌شود؟ و یا به زبان ساده‌تر می‌توان چنین عنوان کرد که کارنمای الگوریتم‌های ساخت‌یافته چه ویژگی بارزی دارد؟ اما قبل از پاسخگویی به این سؤال از شما می‌خواهیم که کار در کلاس زیر را انجام دهید.

کار در کلاس ۴-۱ :

کارنمایی رسم کنید که تعدادی سه تایی **(x, y, z)** *را از ورودی بخواند و ضمن شمارش تعداد سه تایی ها بزرگترین مقدار هر سه تایی را چاپ کند. یک نماد را به‌عنوان نماد انتهای ورود داده در نظر بگیرید. در نهایت نیز تعداد سه تایی ها باید چاپ شود.*

هدف ما از طرح این مسئله این است که شما را به نقش حلقه های تکرار در ساخت یافتگی و یا عدم ساخت‌یافتگی یک الگوریتم واقف‌سازیم. چنانکه از ظاهر این مسئله پیدا است باید یک حلقهٔ تکرار برای دریافت تعداد نامشخصی سه‌تایی مرتب داشته باشیم. این حلقه باید با روش نگهبان کنترل شود اما الگوریتم واحد را چگونه پیاده‌سازی کنیم. یک راه پیاده‌سازی الگوریتم واحد به‌صورتی است که در شکل ۴-۳ مشاهده می‌کنید. کارنمای ارائه شده در این شکل شامل یک الگوریتم غیرساخت‌یافته است. در مقابل در شکل ۴-۴ کارنمای دیگری برای این مسئله ارائه شده که یک الگوریتم ساخت‌یافته را به تصویر کشیده است. به نظر شما چه تفاوت بارزی بین این دو کارنما مشاهده می‌شود؟

همان‌طور که ممکن است شما نیز حدس زده باشید نوعی نظم در کارنمای شکل ۴-۴ به وضوح قابل مشاهده است که در الگوریتم شکل ۴-۳ وجود ندارد. در کارنمای شکل ۴-۳ برای یافتن بزرگترین عدد از بین سه عدد x و y و z مدام به بالا و پایین رفته‌ایم و نوعی بی‌نظمی را بر روندنمای خود تحمیل کرده‌ایم. شاید تعبیر حرکت کاتوره‌ای که در علم فیزیک مطرح می‌شود و در شکل ۴-۱ نشان داده شده بی‌تناسب با روندنماهای غیرساخت‌یافته نباشد.



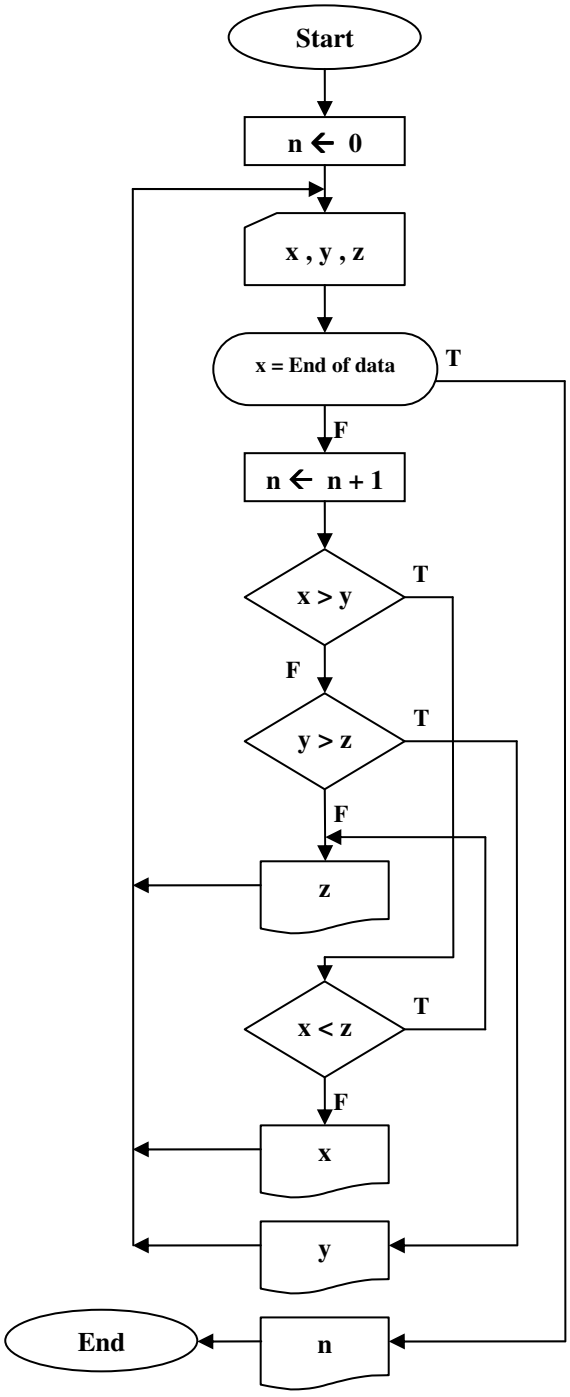
شکل ۴-۱ : نمادی برای حرکت کاتوره‌ای در کارنماهای غیرساخت‌یافته

اما اگر با دقت بیشتری به کارنمای شکل ۴-۴ توجه شود، می‌توان نظم موجود در این کارنما را توصیف‌کرد، و با تعمیم این توصیف به قاعده‌ای جهت تمییزدادن کارنماهای‌ساخت‌یافته از کارنماهای غیرساخت‌یافته دست یافت. بنابراین با توجه به این کارنما قاعدهٔ زیر را به‌عنوان ویژگی بارز کارنماهای ساخت‌یافته مطرح می‌کنیم:

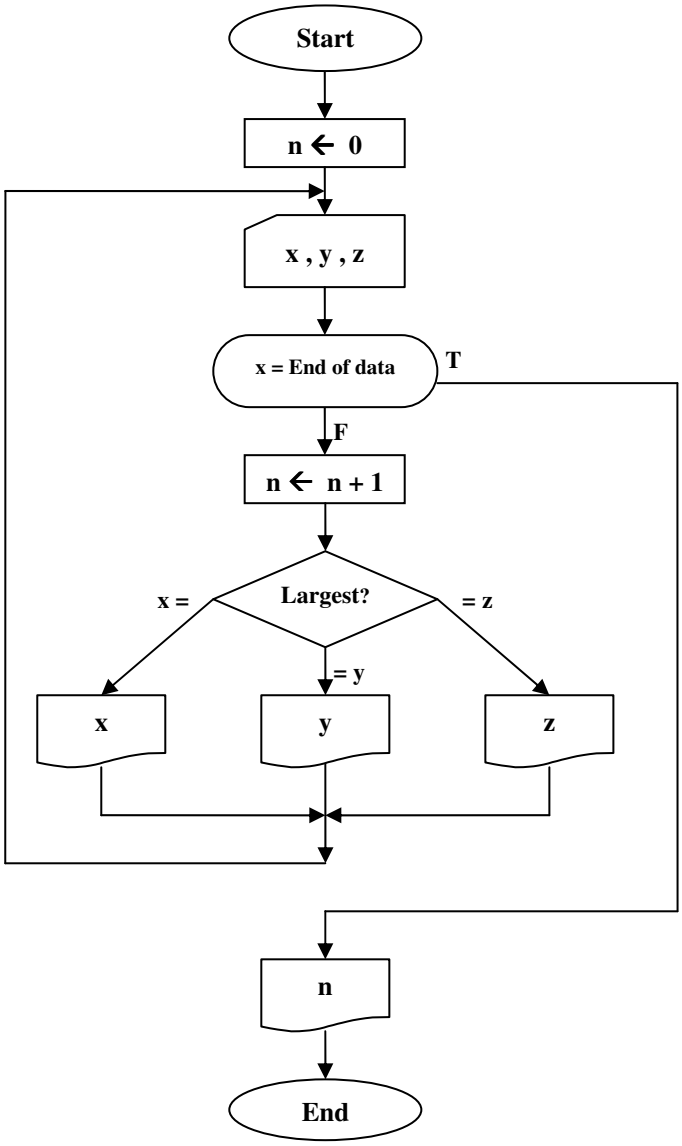
در کارنماهای ساخت‌یافته روند حرکتی کارنما دارای جریانی از سمت بالا به پایین است. البته ممکن است در برخی از نقاط کارنما با حلقه روبروشویم اما در نهایت پس از خروج از حلقه نیز روند روبه پایین کارنما ادامه پیدا می‌کند. بنابراین در الگوریتم ساخت یافته تنها بازگشت به عقب، به‌منظور ساخت حلقه مجاز است.

در شکل ۴-۲ نمادی برای روند روبه پایین کارنماهای ساخت‌یافته آمده است.

شکل ۴-۲ : نمادی برای روند حرکتی کارنماهای ساخت‌یافته



شکل ۴-۳ : الگوریتمی غیرساخت یافته برای کاردرکلاس ۴-۱



شکل ۴-۴ : الگوریتمی ساخت‌یافته برای کاردرکلاس ۴-۱

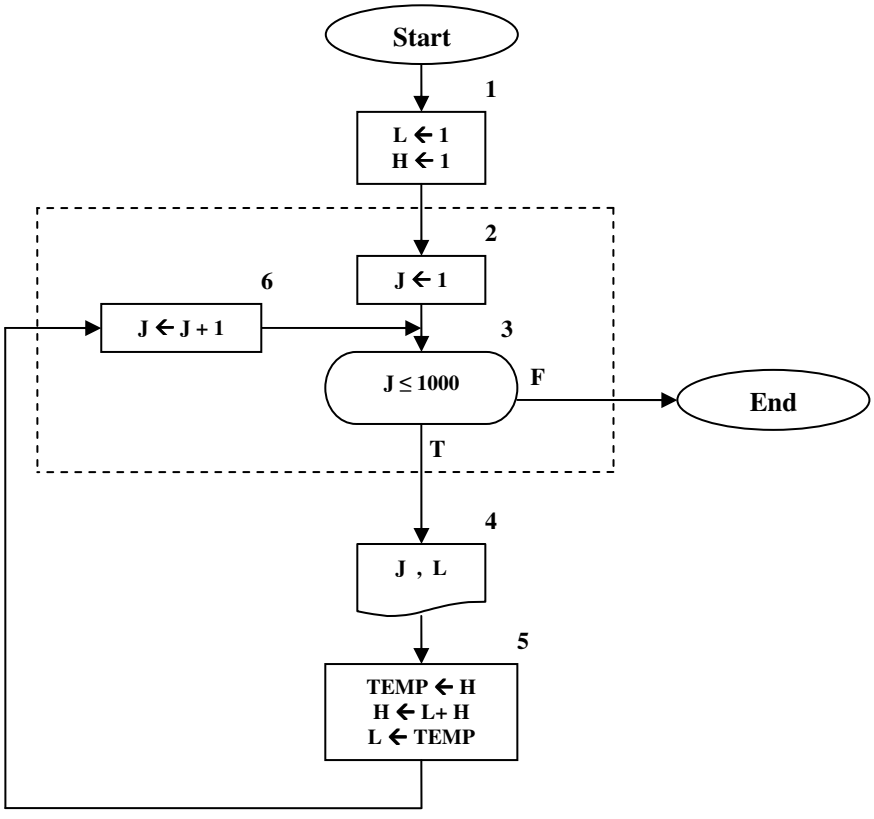
در فصل آتی یکبار دیگر به این دو الگوریتم بازخواهیم‌گشت و در آنجا از شما خواهیم خواست برنامه‌ای به زبان C++ برای این کارنماها بنویسید، تا به تفاوت برنامه‌های ساخت‌یافته با برنامه‌های غیرساخت‌یافته نیز پی ببرید. اما برای آنکه از این پس، بازگشت به عقبی که در نتیجهٔ قراردادن حلقه درکارنماها حاصل‌می‌شود را با حرکت‌های نامنظمی که الگوریتم را از ساخت‌یافتگی خارج می‌کند اشتباه نگیرید و قادر به تشخیص این دو از یکدیگر باشید؛ در بخش ۴-۲ ساختارهایی را برای حلقه‌زنی در کارنماها ارائه می‌کنیم.

۴-۲: جعبه های تکرار

حلقه های دارای شمارنده

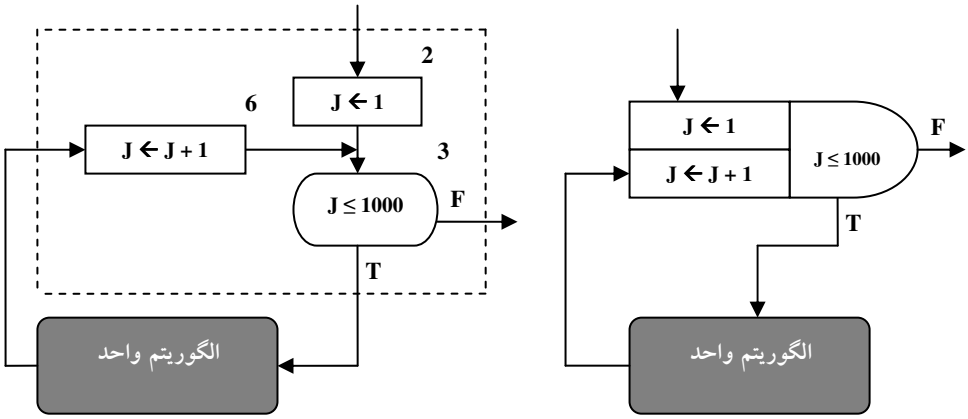
یکبار دیگر به بخش ۲-۴ بازگردید و شکل ۲-۱۳ که در خصوص الگوریتم‌های دارای حلقهٔ تکرار با مکانیسم شمارنده بود را مورد بازبینی قرار دهید. ابتدا تصمیم داریم کارنمایی رسم کنیم که دارای چنین حلقهٔ تکراری باشد سپس جعبه‌ای را برای اینگونه حلقه‌های تکرار ارائه می‌کنیم و سعی خواهیم کرد از این پس کلیهٔ کارنماهای دارای حلقهٔ تکرار با شمارنده را با جعبه‌هایی که برای مکانیسم تکرار در این قسمت ارائه می‌دهیم، رسم کنیم. لذا ابتدا به مثال زیر توجه کنید.

مثال ۴-۱: در زیر کارنمایی ارائه شده که هزار جملهٔ اول دنبالهٔ فیبوناچی را چاپ می‌کند. از آنجا که در این مسئله با تعداد مشخصی جمله روبرو هستیم استفاده از حلقهٔ تکرار دارای شمارنده اجتناب‌ناپذیر است. در دنبالهٔ فیبوناچی هر جمله برابر حاصل جمع دو جملهٔ ما قبل خود است، و جملات اول و دوم برابر یک می‌باشند. با این تفاسیر کارنمای زیر را می‌توان برای این مسئله ترسیم کرد.



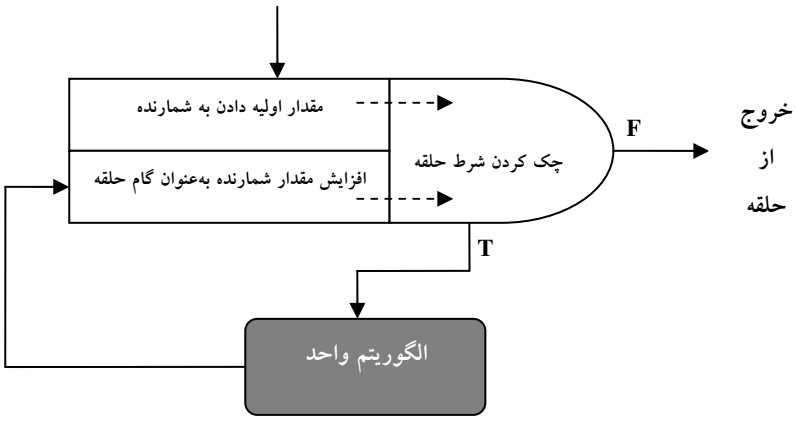
شکل ۴-۵: کارنمای مثال ۴-۱ (چاپ هزار جملهٔ اول دنبالهٔ فیبوناچی)

یکبار دیگر به کارنمای شکل ۴-۵ توجه کنید. در این کارنما تمامی جعبه‌ها به ترتیب روند اجرا شماره‌گذاری شده‌اند. همچنین سه جعبهٔ ۲ و ۳ و ۶ را با خطوط مقطع از دیگر بخش‌های کارنما جدا کرده‌ایم، چرا که این جعبه‌ها قلب عمل تکرار را تشکیل می‌دهند. در حقیقت جعبه‌های ۴ و ۵ را می‌توان به‌عنوان الگوریتم واحد در نظر گرفت. بنابراین تنها جعبه‌های ۲ و ۳ و ۶ هستند که موجب پیدایش گردش در کارنما می‌شوند. در جعبهٔ شماره ۲ ابتدا شمارنده را با عدد یک مقدار اولیه داده‌ایم، در جعبهٔ شمارهٔ ۳ شرط حلقه را چک می‌کنیم و در جعبهٔ شمارهٔ ۶ مقدار شمارنده را یک واحد اضافه می‌کنیم که می‌توان تعبیر گام برداشتن را برای جعبهٔ شمارهٔ ۶ در نظر بگیریم. در شکل ۴-۶ تمامی مراحل مقدار اولیه دادن به شمارنده، چک کردن شرط حلقه و گام برداشتن در جعبه‌ای فشنگی شکل موسوم به *"جعبهٔ تکرار با شمارنده"* شبیه‌سازی شده است.



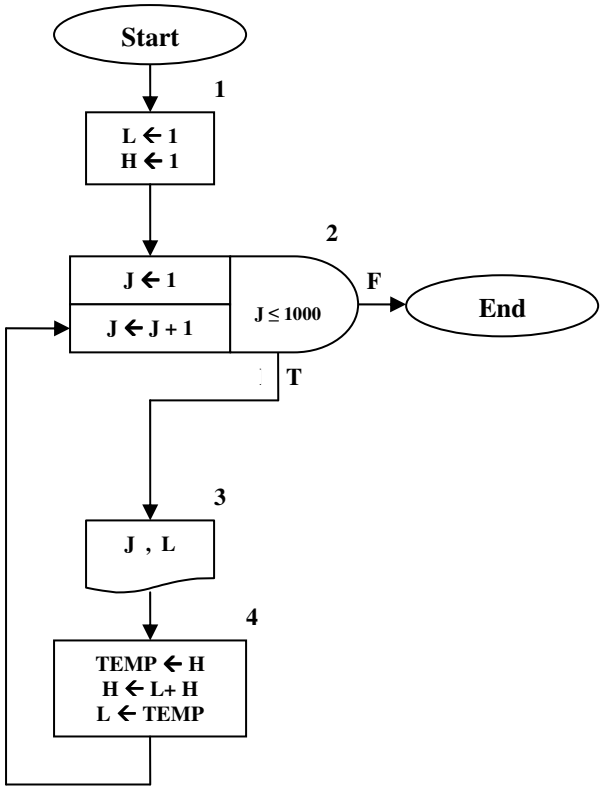
شکل ۴-۶: جعبهٔ تکرار برای حلقه‌های دارای شمارنده

در شکل ۴-۷ مراحل حرکت در یک جعبهٔ تکرار با شمارنده با جزئیات بیشتری نشان داده شده است.



شکل ۴-۷ : ساخت حلقهٔ تکرار به‌عنوان یک قدم واحد

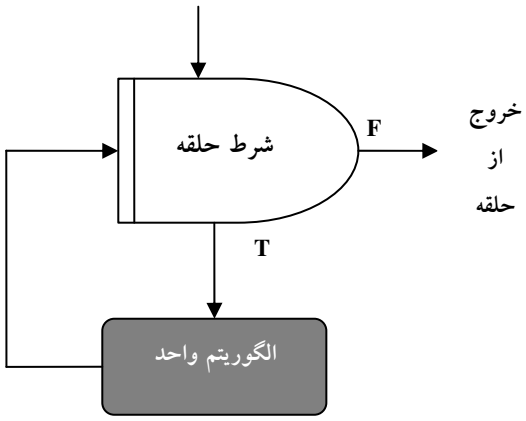
کارنمای مثال ۴-۱ را یکبار دیگر بازنویسی می‌کنیم اما این بار از جعبهٔ تکرار برای ایجاد حلقهٔ تکرار موجود در این الگوریتم استفاده می‌کنیم.



شکل ۴-۸ : کارنمای دنبالهٔ فیبوناچی با جعبهٔ تکرار با شمارنده

در کارنمای فوق پس از ورود به جعبهٔ شمارهٔ یک، به قسمت مقدار اولیهٔ جعبهٔ تکرار وارد می‌شویم. سپس شرط حلقه چک شده و با توجه به درست بودن شرط، الگوریتم واحد، اجرا می‌گردد. پس از آن دوباره به جعبهٔ تکرار باز می‌گردیم، اما این بار در قسمت گام حلقه وارد می شویم. در قسمت گام حلقه، مقدار گام حلقه به شمارنده اضافه می‌شود. توجه داشته باشید که گام حلقه همواره برابر یک نیست، به‌عنوان مثال در برخی موارد که مقدار شمارنده در محاسبات الگوریتم واحد شرکت می‌کند، ممکن است خواستار آن باشیم که مقدار شمارنده دوتا دوتا اضافه گردد. (یعنی گام حلقه را برابر ۲ می‌گیریم). لذا پس از افزایش مقدار گام حلقه به شمارنده دوباره شرط حلقه چک می‌گردد تا در صورت درست بودن شرط حلقه الگوریتم واحد، اجرا گردد و در صورت نادرست بودن شرط حلقه، از جعبهٔ تکرار خارج می‌شویم.

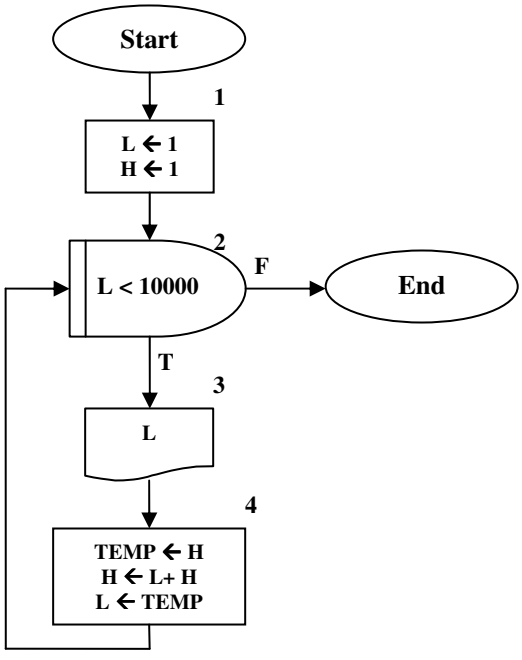
چنانکه در بخش ۲-۴ نیز دیدید نوع دیگری از حلقه‌های تکرار وجود دارد که بدون شمارنده و به واسطهٔ نگهبان کنترل می‌شود. این حلقه‌ها تا زمانی که شرط حلقه درست باشد اجرا می‌شوند. برای اینگونه حلقه‌ها نیز نوعی جعبهٔ تکرار را در شکل ۴-۹ ارائه کرده‌ایم که شباهت زیادی به جعبه‌های تصمیم دارد.



شکل ۴-۹: جعبهٔ تکرار برای حلقه‌های دارای نگهبان

چنانکه پیشتر نیز متذکر شدیم باید شرایطی در داخل دستورات الگوریتم واحد در حلقه‌های دارای نگهبان وجود داشته باشد تا پس از اجرای حلقه به دفعات موردنظر، شرط حلقه نقض گردد و در نتیجه از حلقه خارج شویم. این نوع حلقه در بسیاری از زبان‌های برنامه‌نویسی به حلقه‌های **while** معروف است. در مثال زیر نمونه‌ای از کاربرد جعبهٔ تکرار دارای نگهبان را می‌بینید.

مثال ۴-۲: کارنمایی که جملات دنبالهٔ فیبوناچی را تا جایی که کوچکتر از ۱۰،۰۰۰ باشد چاپ می‌کند.



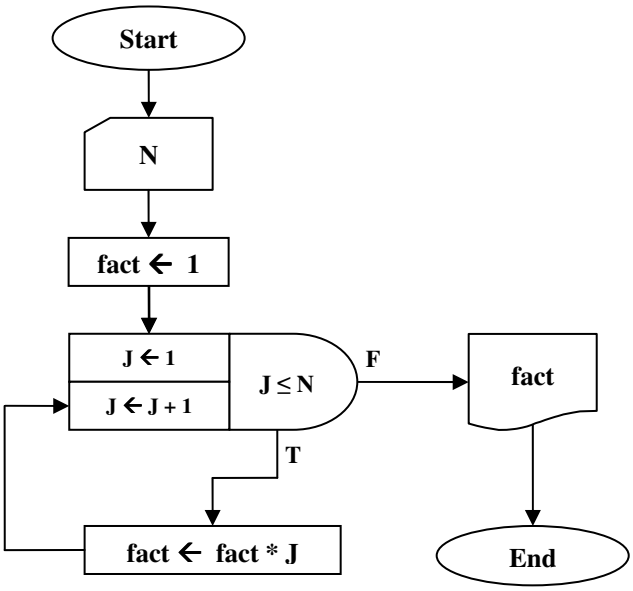
شکل ۴-۱۰ : کارنمای دنبالهٔ فیبوناچی با جعبهٔ تکرار با نگهبان

۴-۳ : مثال هایی بیشتر از به‌کارگیری جعبه های تکرار

حلقه‌های تکرار دارای کاربردهای متعددی خصوصاً در زمینهٔ کار با آرایه‌ها و متغیرهای‌لیستی هستند. اما از آنجاکه بحث در خصوص آرایه ها را به فصل جداگانه‌ای موکول کرده‌ایم، در این بخش تصمیم داریم با مطرح کردن مثال های دیگری از کاربرد حلقه‌های‌تکرار در مسائل گوناگون، نحوهٔ به‌کارگیری جعبه‌های‌تکرار را بیشتر بیاموزیم.

کار با فاکتوریل در الگوریتم‌ها

مثال ۴-۳ : کارنمایی که عدد صحیح و مثبت N را از ورودی خوانده و مقدار **N!** را محاسبه و چاپ می‌کند.



شکل ۴- ۱۱: کارنمای محاسبهٔ فاکتوریل

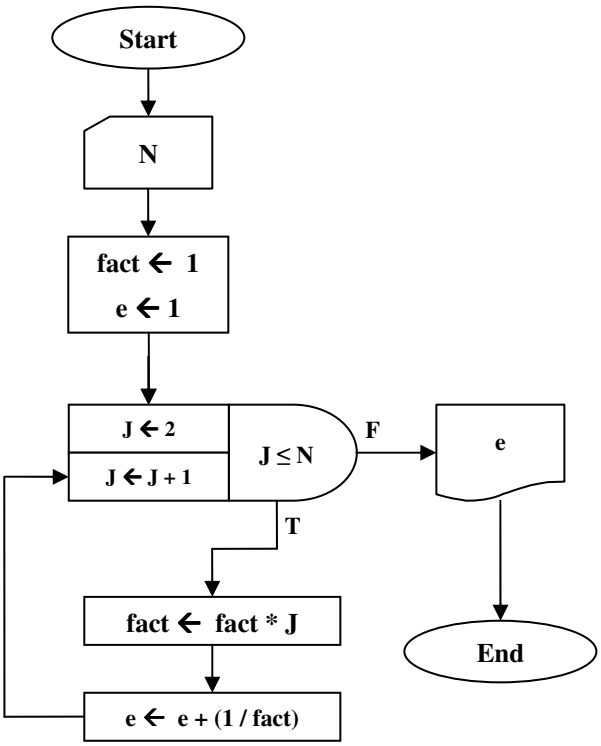
توضیح کارنما : با توجه به آن که مقدار **N!** برابر $N \times \ldots \times ۳ \times ۲ \times ۱$ است. ابتدا متغیر **fact** را با یک مقدار اولیه داده‌ایم. زیرا حتی **0!** هم برابر یک است. اما در صورتی که **N>0** باشد به تعداد دفعات موردنظر، مقدار شمارنده را در متغیر **fact** ضرب می‌کنیم. این عمل تا جایی ادامه پیدا می‌کند که شرط **J≤N** برقرار باشد، و در نتیجه مقدار **N!** محاسبه می‌گردد.

مثال ۴-۴ : کارنمایی که مقدار **e** (پایه لگاریتم نپرین) را با استفاده از رابطهٔ زیر محاسبه می‌کند.

$$e = 1 + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{n!}$$

توضیح الگوریتم : برای محاسبهٔ سری فوق به یک متغیر مانند **e** نیازمندیم که مجموع به‌دست آمده را در خود نگهدارد. از طرفی به یک حلقهٔ تکرار نیازمندیم که هربار عملیات محاسبهٔ یک جمله از این سری را تکرار کند، و در نهایت مقدار

به‌دست آمده را با متغیر **e** جمع کند تا مجموع جدید به‌دست آید. در این کارنما نیز متغیر **fact** مقدار فاکتوریل را برای جمله‌ای که هم اکنون در حال محاسبهٔ آن هستیم، در خود نگه می‌دارد.



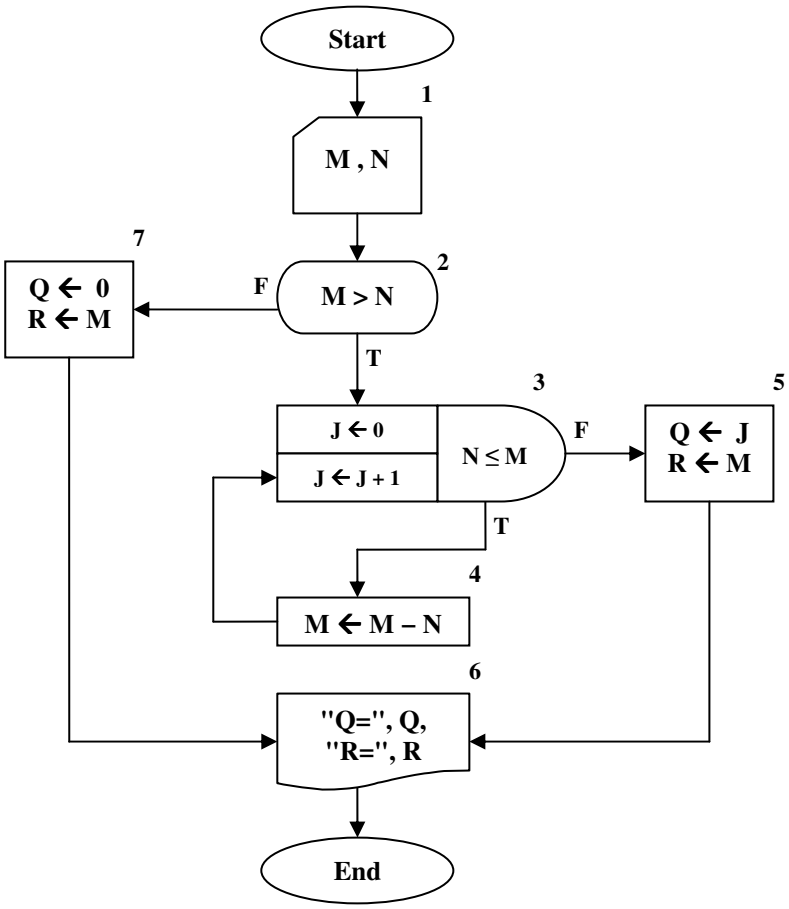
شکل ۴- ۱۲: کارنمای محاسبهٔ عدد e

مسائل مرتبط با ساده‌سازی و تقسیم اعداد

مثال ۴-۵ : کارنمایی که دو عدد صحیح و مثبت را از ورودی خوانده، و آنها را با عمل تفریق بر هم تقسیم می‌کند.

توضیح الگوریتم : هنگامی که عدد **M** را بر عدد **N** تقسیم می‌کنیم، در حقیقت تعداد **N**هایی که می‌توان در عدد **M** جدا کرد را می‌شماریم. بنابراین می‌توانیم هر بار که یک **N** را از **M** جدا می‌کنیم یک واحد به شمارنده بیفزاییم. این عمل تا جایی می‌تواند ادامه پیدا کند که **N** بزرگتر از باقی‌ماندهٔ **M** از مرحلهٔ قبل باشد. هنگامی که **N** از **M** کوچکترشود مقدار شمارنده برابر خارج‌قسمت تقسیم و آنچه که در **M** باقی می‌ماند برابر باقی‌ماندهٔ تقسیم می‌باشد. کارنمای این مثال در شکل ۴-۱۳ ارائه شده است. در این کارنما **R** برابر باقی‌مانده و **Q** برابر خارج‌قسمت تقسیم می‌باشد.

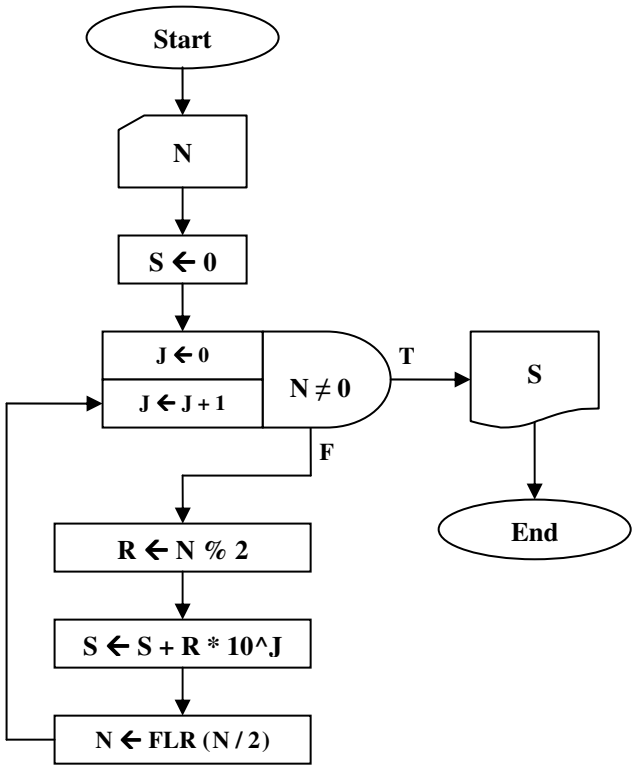
به نظر شما آیا لزومی بر قراردادن جعبه‌های ۲ و ۷ در الگوریتم شکل ۴-۱۳ وجود داشت؟ آیا با حذف این دو جعبه اتفاق خاصی رخ می‌دهد؟ شما چه نتیجه‌ای از این بحث می‌گیرید؟



شکل ۴-۱۳: کارنمای محاسبهٔ تقسیم به واسطهٔ عمل تفریق

مثال ۴-۶: کارنمایی که یک عدد مبنای ۱۰ را از ورودی خوانده، و آن را به مبنای دو می‌برد و نتیجه را چاپ می‌کند.

توضیح الگوریتم: تبدیل یک عدد از مبنای ۱۰ به مبنای ۲، با تقسیم‌های متوالی انجام می‌شود، به‌طوری که باقی‌مانده‌ها باید از آخرین باقی‌مانده به اولین باقی‌مانده چاپ شود. چنانکه می‌بینید اولین رقم خروجی در آخرین مرحله به‌دست می‌آید. یک راه برای چاپ این باقی‌مانده‌ها ذخیرهٔ آنها در یک مکان از حافظه است، تا در نهایت بتوان آنها را به ترتیب از آخر به اول چاپ کرد. اما این کار چندان عقلانی نیست چون ما را به یک تعداد خانه از حافظه محدود می‌کند، و اگر عدد ورودی از یک مقدار حداکثری، بیشتر باشد الگوریتم ما با مشکل مواجه می‌شود. اما یک راه دیگر آن است که نتیجهٔ دودویی خود را به‌صورت یک عدد مبنای ۱۰ فرض کنیم، با این توضیح که این عدد تنها از ارقام صفر و یک تشکیل‌شده است. بنابراین از آنجا که باقی‌ماندهٔ هر مرحله باید به‌عنوان رقم منتهاالیه سمت چپ عدد خروجی، قرارگیرد؛ می‌توان باقی‌ماندهٔ جدید را در توانی از ده ضرب‌کرد و با حاصل مرحلهٔ قبل جمع نمود تا در نهایت نتیجه در مبنای دو به‌دست آید.



شکل ۴- ۱۴: کارنمای تبدیل یک عدد مبنای ۱۰ به عددی در مبنای ۲

هم‌عامل‌های یک عدد صحیح

پیشتر در مثال ۲-۸ الگوریتمی ارائه کردیم که تمامی مقسوم‌علیه‌های یک عدد صحیح را به‌دست می‌آورد. اما در این قسمت تصمیم داریم الگوریتمی ارائه کنیم که تمامی «هم‌عامل‌های» یک عدد صحیح مثل N را چاپ کند. مقصود از هم عامل‌های یک عدد صحیح مثل N ، تمامی جفت اعدادی است که حاصل‌ضرب آنها مساوی خود عدد N گردد. به‌عنوان مثال هم‌عامل‌های عدد ۲۰ شامل سه دوتایی (۱.۲۰) و (۲.۱۰) و (۴.۵) می‌باشد. در حقیقت این اعداد مقسوم‌علیه‌های ۲۰ هستند که به‌صورت جفت‌های دوتایی انتخاب شده‌اند، و یا هم‌عامل‌های عدد ۲۵ عبارت است از (۱.۲۵) و (۵.۵). بنابراین رابطه‌ای تنگاتنگ بین این مسئله و پیدا کردن مقسوم‌علیه‌های عدد N وجود دارد، بنابراین از تجربه‌های خود در مثال ۲-۸ استفاده خواهیم‌کرد. یک راه ساده برای حل این مسئله چنین است: «تمامی مقسوم‌علیه‌های عدد مفروض N را به‌دست آور و هر مقسوم‌علیه را به‌همراه هم‌عاملش چاپ کن.»

قسمت اول مسئله یعنی یافتن مقسوم‌علیه‌های یک عدد را پیشتر حل کرده‌ایم. اما چنانکه در مسئلهٔ شمارهٔ یک بخش ۲-۱۱ نیز مطرح شد، الگوریتم مثال ۲-۸ کارایی لازم را ندارد. چرا که اگر به‌عنوان مثال عدد N را برابر یک میلیون در نظر بگیریم حلقهٔ تکرار کارنمای مثال ۲-۸ یک میلیون بار تکرار می‌شود. ولی عملاً نیمی از این تکرارها بیهوده است. زیرا هنگامی که $M > N / 2$ می‌شود دیگر هیچ مقسوم‌علیه‌ای یافت نمی‌شود جزء خود عدد N که بخش‌پذیری آن بر خودش بدیهی است. بنابراین نباید اشتباهی را که در این مثال مرتکب شدیم در این مسئله نیز تکرار کنیم، و باید با دیدی

باز شرط حلقه را به‌گونه‌ای تنظیم کنیم که از تکرارهای بیهوده جلوگیری شود. بنابراین یکبار دیگر راه‌حل خود و استراتژی حل مسئله را به‌طور موشکافانه مورد ارزیابی قرارمی‌دهیم.

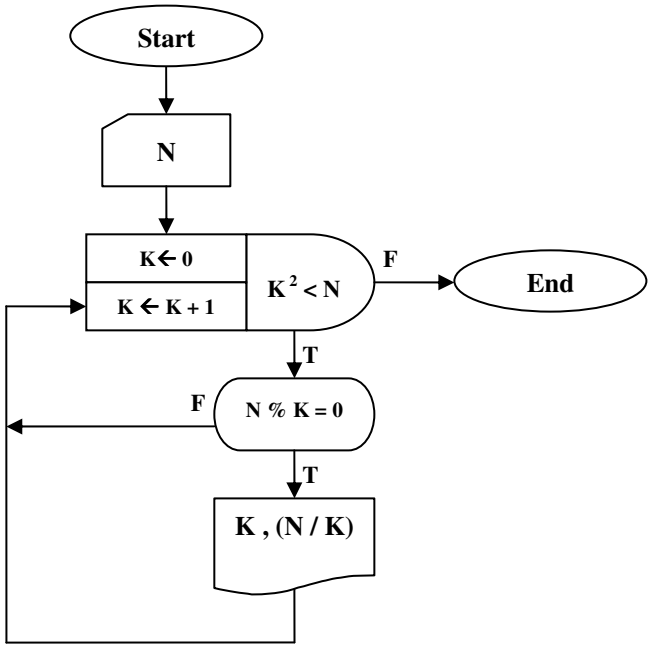
اگر فرض‌کنیم عدد صحیحی مثل **K** یک مقسوم‌علیه عدد **N** باشد خود به خود هم عامل آن با توجه به رابطهٔ روبه رو به‌دست می‌آید:

$$N = K \times \frac{N}{K}$$

بنابراین در این مسئله تعداد بسیار کمتری از مقسوم‌علیه‌های عدد مفروض **N** را باید به‌دست بیاوریم. اما چگونه تعیین‌کنیم که روندِ یافتنِ مقسوم‌علیه‌های عدد **N** را تا کجا باید ادامه پیدا کند؟ پاسخ این سؤال بسیار ساده است، عدد **K** و هم‌عاملش هردو نمی‌توانند کوچکتر از $\sqrt{N}$ باشند. چرا که طبق رابطهٔ زیر اگر هر دوی آنها کوچکتر از $\sqrt{N}$ باشند به تناقض می‌رسیم:

$$K < \sqrt{N} , \frac{N}{K} < \sqrt{N} \ \Rightarrow N = K \times \frac{N}{K} < \sqrt{N} \times \sqrt{N} = N$$

بنابراین تنها کافی‌است تمامی مقسوم‌علیه‌های کوچکتر از $\sqrt{N}$ عدد مفروض **N** را به‌دست‌آوریم در این صورت مقسوم‌علیه دیگر که بزرگتر از $\sqrt{N}$ می‌باشد خود به خود به‌دست می‌آید. حال به کارنمای این الگوریتم توجه کنید.



شکل ۴- ۱۵: کارنمای پیدا کردن هم‌عامل‌های عدد مفروض N

اگر به شرط حلقهٔ تکرار توجه کرده باشید به جای عبارت $K < \sqrt{N}$ از عبارت $K^2 < N$ استفاده کردیم و این بدان جهت است که محاسبهٔ جذر در کامپیوتر همراه با تقریب است و می‌تواند دقّت محاسبات ما را کاهش دهد در حالی که محاسبهٔ توان دوم یک عدد، با هیچگونه تقریبی همراه نیست.

الگوریتم اقلیدس (یافتن ب.م.م دو عدد)

الگوریتم اقلیدس در کتاب پنجم اقلیدس آمده است و تاریخ آن حداقل به ۳۰۰ سال قبل از میلاد مسیح می‌رسد. این الگوریتم روشی برای پیدا کردن بزرگترین مقسوم‌علیه مشترک دو عدد صحیح است. قبل از بیان الگوریتم اقلیدس باید به ذکر برخی مقدّمات بپردازیم.

با «بزرگترین مقسوم‌علیه مشترک» (ب.م.م) دو عدد در دروس ریاضی دورهٔ راهنمایی آشنا شدید. ساده‌ترین راهی که می‌توان ب.م.م دو عدد را تعیین کرد بدین‌صورت است که ابتدا کلیهٔ مقسوم‌علیه‌های هر دو عدد را تعیین کنیم. سپس مجموعهٔ مقسوم‌علیه‌های مشترک دو عدد را یافته و در نهایت بزرگترین عضو این مجموعه را تعیین می‌کنیم. به‌عنوان مثال ب.م.م دو عدد ۵۴ و ۷۲ مطابق با این روش عبارت است از:

S = مجموعه مقسوم‌علیه‌های ۵۴ = (۵۴.۲۷.۱۸.۹.۶.۳.۲.۱)

T = مجموعه مقسوم‌علیه‌های ۷۲ = (۷۲.۳۶.۲۴.۱۸.۱۲.۹.۸.۶.۴.۳.۲.۱)

اشتراک این دو مجموعه برابر است با:

$S \cap T = \{۱.۲.۳.۶.۹.۱۸\}$

و از آنجا که بزرگترین عضو این مجموعه برابر ۱۸ می‌باشد، ب.م.م دو عدد ۵۴ و ۷۲ برابر ۱۸ می‌باشد.

اما روش دیگری که غالباً از آن برای تعیین ب.م.م دو عدد استفاده می‌شود روش نردبانی است که از قبل با آن آشنایی دارید. اساس روش نردبانی بر قاعدهٔ ریاضی زیر استوار است. فرض کنید سه عدد صحیح X و Y و Z دارای رابطهٔ زیر باشند:

$$Z = Y + X$$

حال اگر هر دوی این اعداد بر a بخش‌پذیر باشند، عدد سوم نیز بر a بخش‌پذیرخواهدبود. به‌عنوان مثال فرض‌کنید $Y=ay$ و $Z=az$ باشند آنگاه داریم:

$$a(z-y) = X \Rightarrow az = ay + X$$

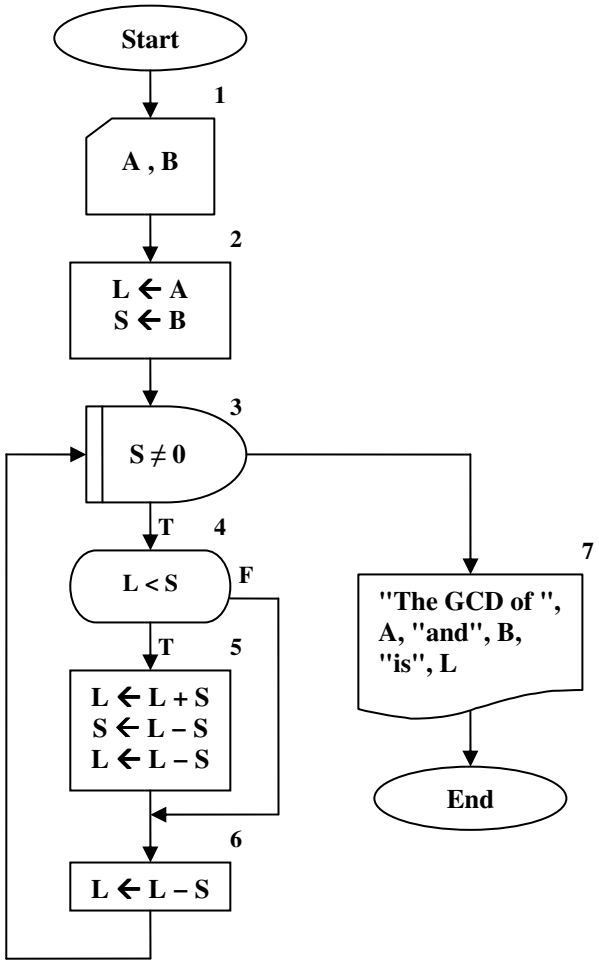
بنابراین عدد X نیز بر عدد a بخش‌پذیر می‌باشد. با توجه به رابطهٔ اخیر هر دو عددی را که از میان این سه عدد انتخاب کنیم، مقسوم‌علیه‌های مشترک آنها همانند مقسوم‌علیه مشترک هر دو عدد دیگری است که از بین آنها انتخاب کنیم. برای روشن شدن مسئله سه عدد ۹۴۳ و ۴۳۷ و ۵۰۶ را در نظر بگیرید. این اعداد دارای رابطهٔ زیر هستند:

$$۹۴۳ = ۵۰۶ + ۴۳۷$$

با توجه به حکم فوق دو عدد ۵۰۶ و ۴۳۷ دارای همان مقسوم‌علیه‌هایی هستند که اعداد ۹۴۳ و ۵۰۶ دارند. حال برای یافتن مقسوم‌علیه مشترک دو عدد ۹۴۳ و ۵۰۶ می‌توانیم مقسوم‌علیه مشترک دو عدد ۵۰۶ و ۴۳۷ را بیابیم. این روند می‌تواند تا جایی ادامه پیدا کند که محاسبهٔ مقسوم‌علیه مشترک دو عدد به‌سادگی امکان‌پذیر باشد. یعنی می‌توانیم برای محاسبهٔ مقسوم‌علیه مشترک دو عدد ۵۰۶ و ۴۳۷، مقسوم‌علیه مشترک دو عدد ۴۳۷ و ۶۹ را بیابیم. زیرا:

$$۵۰۶ - ۴۳۷ = ۶۹$$

اثبات می‌شود که مقسوم‌علیه مشترکی که از این روش به‌دست می‌آید همان بزرگترین مقسوم‌علیه مشترک دو عدد است. در شکل ۴-۱۶ کارنمایی ارائه شده که روند فوق را برای پیدا کردن ب.م.م دو عدد پیاده‌سازی می‌کند.



شکل ۴- ۱۶: کارنمای الگوریتم اقلیدس با استفاده از تفریق

کار در کلاس ۴-۲ :

۱. آیا می توانید بگویید چه عملی در جعبهٔ شمارهٔ ۵ انجام می شود؟

۲. روندنمای فوق را برای ورودی های زیر امتحان کنید.

- **A = 177 , B = 379**
- **A = 3363 , B = 4465**

کار در کلاس ۴-۳ :

کارنمای شکل ۴-۱۶ را به‌گونه ای بازنویسی کنید که الگوریتم اقلیدس را به واسطهٔ تقسیم پیاده سازی کند. سپس الگوریتم خود را تعمیم دهید تا ک.م.م دو عدد را نیز محاسبه و چاپ کند.

راهنمایی : برای محاسبهٔ ک.م.م ساده ترین راه استفاده کردن از فرمول زیر است :

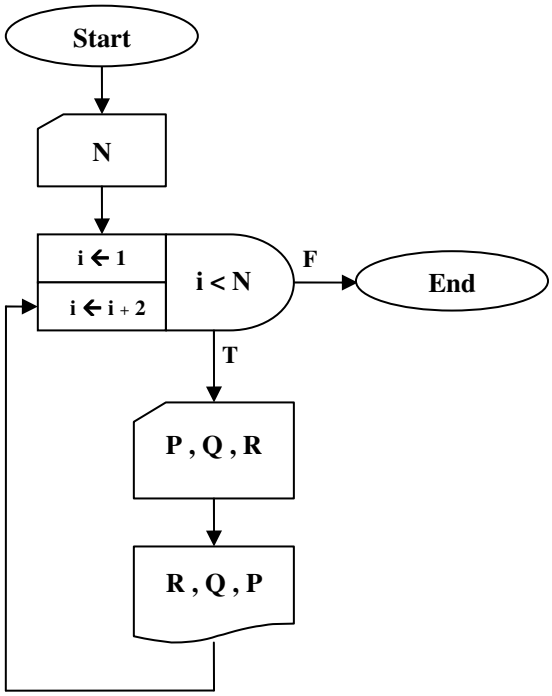
حاصل ضرب دو عدد = ک.م.م × ب.م.م

۴-۴ : تمرین

۱. تفاوت الگوریتم‌های ساخت‌یافته با الگوریتم‌های غیرساخت‌یافته چیست؟

۲. یک روندنما و دنباله‌ای از داده های ورودی به شما داده شده است. وظیفهٔ شما این است که تعیین کنید چه مقادیری به خارج داده خواهد شد.

دنبالهٔ داده‌های ورودی: ۷ ، ۱ ، ۲ ، ۳ ، ۴ ، ۵ ، ۶ ، ۷ ، ۸ ، ۹ .



شکل ۴-۱۷ : کارنمای تمرین ۲

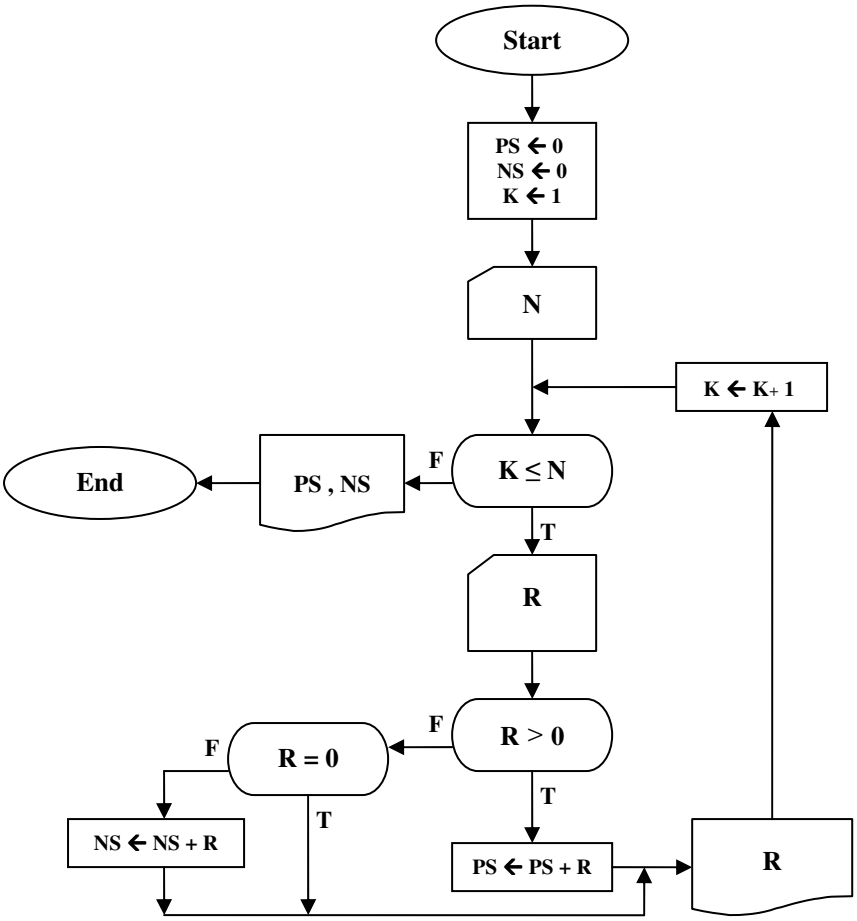
۳. کارنمایی رسم کنید که ب.م.م دو عدد را به روش زیر محاسبه کند.

فرض کنید N از M کوچکتر است. مقسوم‌علیه‌های N را از بزرگ به کوچک به‌دست آورید و آزمایش کنید که آیا M نیز بر آن مقسوم‌علیه بخشپذیر است یا نه. اولین مقسوم‌علیه N که M نیز بر آن بخشپذیر باشد برابر ب.م.م دو عدد M و N است.

۴. کارنمایی رسم کنید که مجموع ارقام عدد مفروض N را به‌دست آورد.

۵. کارنمایی رسم کنید که تعداد N عدد صحیح را از ورودی بخواند، و اعدادی را که بر ۶ قابل قسمت هستند را به همراه تعدادشان چاپ کند.

۶. روندنمایی را که در شکل ۴-۱۸ نشان داده شده است مجدّداً به‌گونه‌ای رسم کنید که، حلقه‌ای که توسط شمارندهٔ K کنترل شده است، تحت کنترل یک جعبهٔ تکرار باشد.



شکل ۴- ۱۸ : کارنمای تمرین ۶

۷. عدد طبیعی N مفروض است. کارنمایی رسم کنید که معین کند N چند رقم دارد.

راهنمایی : برای حل این مسئله از دو طریق می‌توانید عمل کنید. یکی از طریق عملگر باقی‌ماندهٔ تقسیم و جدا کردن ارقام N و شمارش تعداد ارقام آن. دوم از طریق رابطهٔ زیر که ثابت می‌شود عدد طبیعی N دارای k رقم است.

$$(10^{(k-1)} \leq N < 10^{k})$$

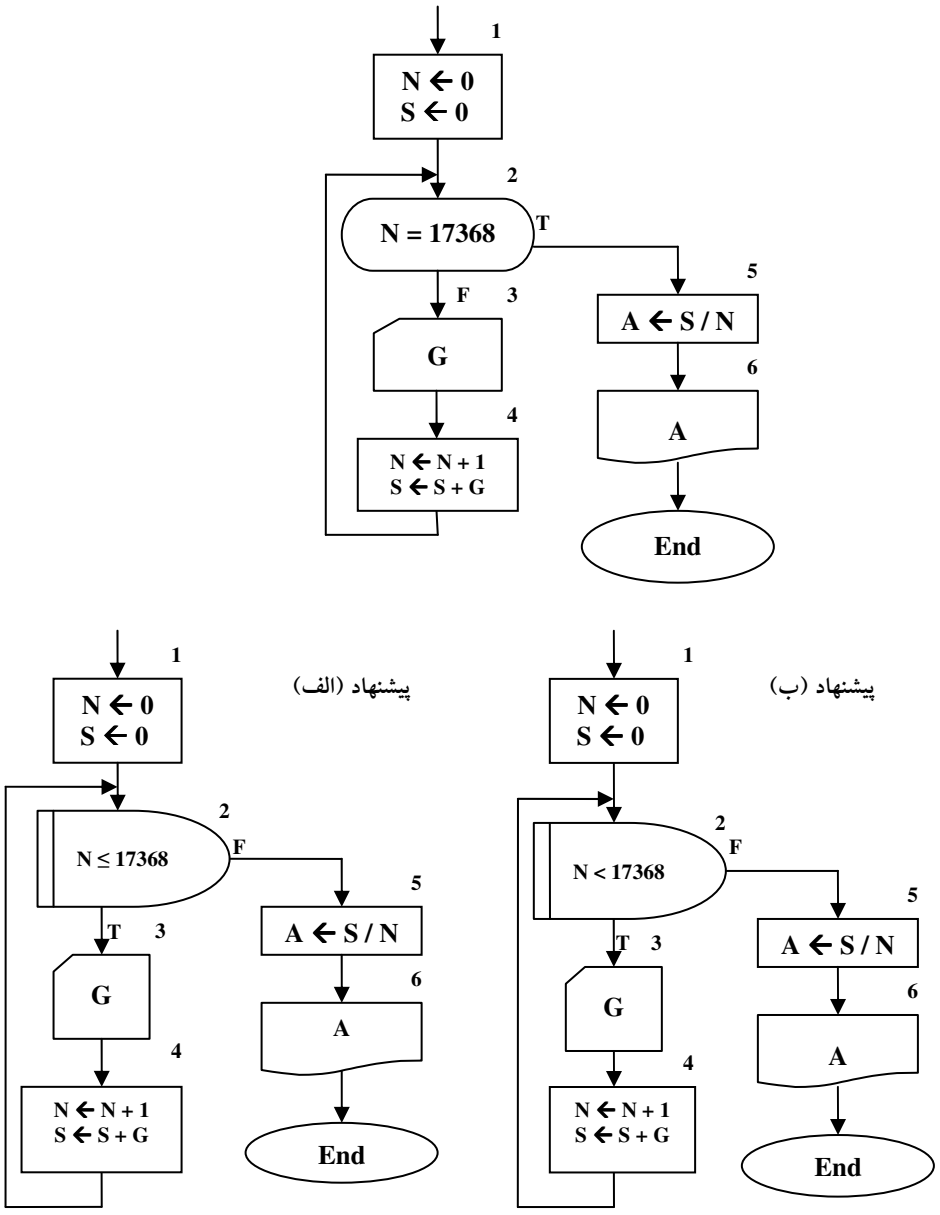
۸. مسئلهٔ قبل را به‌گونه‌ای تعمیم دهید که برای اعداد منفی و در نتیجه برروی اعداد صحیح نیز جواب صحیح بدهد.

۹. کارنمایی رسم کنید که مقلوب عدد مفروض N را به‌دست‌آورد. مقلوب عدد ۳۴۲۸ برابر ۸۲۴۳ می‌باشد.

۱۰.کارنمایی رسم کنید که بدون محاسبهٔ باقی‌ماندهٔ تقسیم بخشپذیر بودن عدد مفروض N را بر عدد ۱۱ به‌دست آورد؟

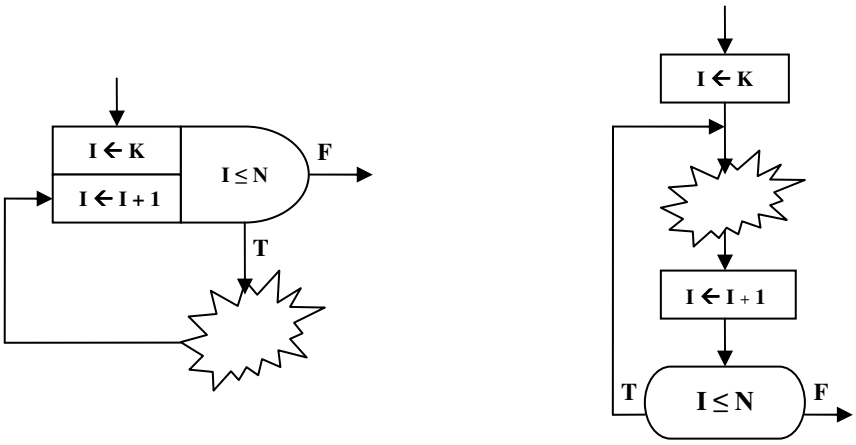
۱۱.الگوریتمی بنویسید که با دریافت یک عدد N اول بودن یا نبودن آن را تشخیص دهد.

۱۲.در این مسئله در ابتدا قطعه روندنمایی به شما داده شده است که دارای یک حلقهٔ کنترل شده به وسیلهٔ شمارنده است. سپس چند قطعهٔ روندنما، که در آنها از جعبهٔ تکرار **While** استفاده شده، برای انجام همان کار پیشنهاد شده است. از شما می‌خواهیم تا پیشنهادی را که واقعاً همان کار را انجام می‌دهد انتخاب کنید.



شکل ۴- ۱۹ : کارنمای تمرین ۱۲

۱۳.دو قطعهٔ روندنما در زیر نشان داده شده و فرض بر این است که محتوای قسمت ابری در هر دو یکی است. تحت چه شرایطی این دو روندنما معادل یکدیگر نخواهند بود؟



شکل ۴-۲۰ : کارنمای تمرین ۱۳

۱۴.یک توزیع‌کننده، تعداد ۱۰.۰۰۰ اسباب‌بازی کوچک را که در جعبه‌های مکعب مستطیلی با ابعاد گوناگون بسته‌بندی شده است خریداری کرده است. وی خیال دارد به‌منظور مخفی نگه‌داشتن اسباب‌بازی‌ها و در نتیجه جلب‌توجه بیشتر بچه‌ها، این جعبه‌ها را در کُره‌های پلاستیکی قراردهد و مجدداً به فروش برساند. از این رو او باید بداند که چند عدد کُره با قطرهای مشخص (۱۰، ۱۵، ۲۰، ۲۵، ۳۰ سانتیمتر) احتیاج دارد. چون بزرگترین اندازهٔ خارجی یک مکعب مستطیل به ابعاد **A** ، **B** و **C** قطر آن است، لذا توزیع‌کننده باید با استفاده از رابطهٔ زیر:

$$D = \sqrt{A^2 + B^2 + C^2}$$

قطر جعبه‌ها را حساب کند. سپس باید تعیین کند که تعداد جعبه‌هایی که قطرشان ۱۰ سانتیمتر یا کمتر است چقدر است، تعدادی که قطرشان بین ۱۰ و ۱۵ سانتیمتر است چقدر است، و به همین ترتیب. تصمیم توزیع‌کننده بر این قرار می‌گیرد که قطر هر ۱۰.۰۰۰ جعبه را حساب کند. هر جعبه دارای یک شمارهٔ مشخصهٔ **ID** و مجموعهٔ اندازه‌های **A** ، **B** و **C** است. کار شما ترسیم روندنمایی است برای تهیهٔ یک جدول چاپی که هر خط آن دارای پنج قلم شامل مقادیر ورودی **ID**، **A**، **B** ، **C** و مقدار حساب‌شدهٔ **D** باشد. هر خط از جدول باید نظیر یکی از بسته های اسباب‌بازی باشد. از جعبهٔ تکرار استفاده کنید. تعداد جعبه های مورد نیاز روندنما، بدون احتساب جعبه های شروع و پایان، فقط چهار تا است.

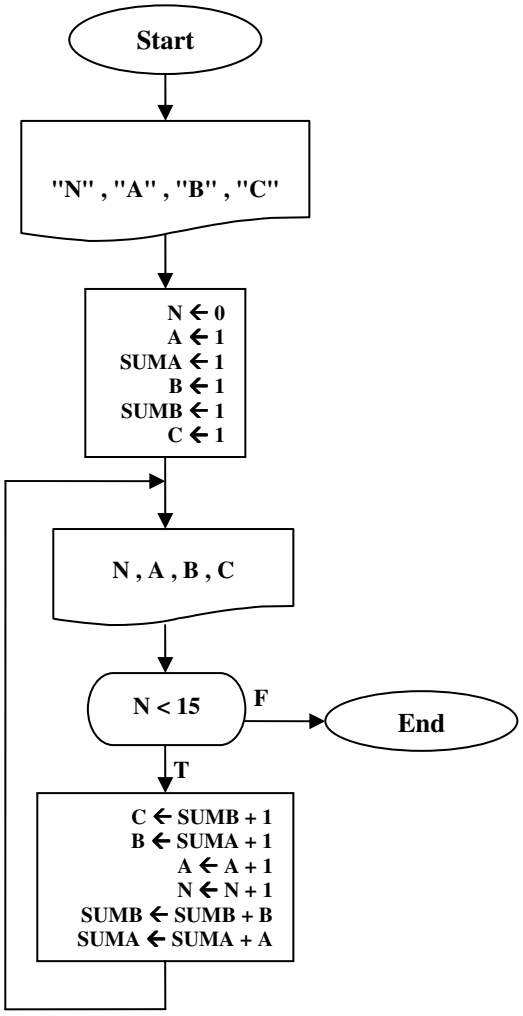
۱۵.الگوریتمی بنویسید که تعداد نامشخصی عدد را از ورودی بخواند و مشخص کند که آیا این اعداد مرتب هستند یا خیر؟ و اگر مرتب هستند، ترتیب صعودی است یا نزولی؟ الگوریتم باید در هنگام ورود داده‌ها مرتب بودن یا نبودن آنها را تشخیص دهد. آخرین ورودی نیز $ است.

۱۶.این مسئله دنبالهٔ مسئله ۱۴ است. فرض کنید توزیع‌کنندهٔ اسباب‌بازی متوجه می‌شود که برای تعیین تعداد کره‌های پلاستیکی موردنیاز از هر قطر، راه ساده‌تری از پویش فهرست ۱۰.۰۰۰ خطی وجود دارد. روندنمایی را که در مسئله ۱۴ تهیه کرده‌اید چنان تغییر دهید که به‌جای چاپ آن جدول طولانی خود الگوریتم، حساب تعداد کره‌های مورد نیاز از هر سایز را نگه دارد و پس از پردازش تمامی ۱۰.۰۰۰ داده، تعداد کره‌های موردنیاز از هر سایز را چاپ کند.

۱۷. روند نمای شکل ۴-۲۱ را مطالعه کنید و به سؤال‌های زیر پاسخ دهید.

الف- مقادیری را که بر روی پنج خط اول خروجی الگوریتم چاپ می‌شود نشان دهید.

ب- این روندنما را از نو با استفاده از جعبهٔ تکرار رسم کنید.



شکل ۴- ۲۱ : کارنمای تمرین ۱۷

۱۸.روندنمایی بسازید که جدولی شامل چهار ستون را محاسبه کند و در خروجی بنویسد. ستون اول باید شامل اعداد از ۱ تا ۱۰۰ باشد. ستون دوم باید شامل مربعات اعداد نظیر خود در ستون اول باشد. ستون‌های سوم و چهارم باید شامل مکعب‌ها و توان‌های چهارم اعداد ستون اول باشد. برای مثال، دومین خط خروجی باید چنین باشد:

۲ ۴ ۸ ۱۶

۱۹.الگوریتمی بسازید که اولین ۵۰۰ عدد صحیح را آزمایش کند و فقط آنهایی را که کامل هستند در خروجی بنویسد. عدد کامل عددی است که مساوی حاصل‌جمع همهٔ عامل‌های خود باشد. کوچکترین عدد کامل ۶ است.

۶ = ۱ + ۲ + ۳

(بنابه تعریف، عدد یک، جزو اعداد کامل نیست.)

۲۰.لیست **X** که شامل **N** مقدار X_1 ، X_2 ، ... ، X_N و یک مقدار **A** است داده شده است.

الف- روند نمایی برای محاسبهٔ **NUM** ، که تعریف ریاضی آن به‌صورت حاصل ضرب **N** جمله‌ای زیر داده شده است، ترسیم کنید.

$$NUM = (X_1 - A) \times (X_2 - A) \times (X_3 - A) \times ... \times (X_N - A)$$

روندنمای شما باید شامل خواندن و نوشتن کلیهٔ داده‌های مورد نیاز و نتایج حاصله باشد. شما نیز یک متغیر با زیرنویس **i** بگیرید و در یک حلقهٔ دارای شمارنده مقادیر را برای متغیر دارای زیرنویس مذکور بخوانید و یا بنویسید.

ب- مانند قسمت (أ) ولی با این تفاوت که جملهٔ **K**- اُم از **N** جملهٔ حاصل‌ضرب حذف شده باشد.

۲۱.الگوریتمی بنویسید که تعدادی عدد را بخواند و هر عدد که بر ۹ بخشپذیر باشد را در خروجی چاپ کند.(از روش مجموع ارقام استفاده کنید.)

۲۲.الگوریتمی را بنویسید که عددی را با هر طول دلخواه بخواند و مشخص کند که آیا عدد متقارن است یا خیر. به‌عنوان مثال عدد ۱۲۳۲۱ متقارن است.

۲۳.الگوریتمی بنویسید که عدد **N** را از ورودی بخواند و ۱۰ مضرب اول **N** را چاپ کند.

۲۴.الگوریتمی بنویسید که بدون محاسبهٔ ب.م.م مقدار ک.م.م دو عدد را محاسبه و چاپ کند.

۲۵.الگوریتمی بنویسید که ابتدا دو عدد **a** و **b** را به‌عنوان دو کران بازهٔ **[a, b]** از کاربر بگیرد، سپس با دریافت عدد مفروض **N** این بازه را به تعدادی زیر بازه با طول‌های مساوی تقسیم و هر زیر بازه را به‌طور جداگانه چاپ کند. به‌عنوان مثال برای بازهٔ **[0,1]** و عدد **n=4** باید مقادیر زیر را چاپ کند.

[0, 0.25] , [0.25, 0.5] , [0. 5, 0.75] , [0.75, 1]

۲۶. کارنمایی رسم کنید که یک عدد و مبنای آن را دریافت و آن را از مبنای ۸ به مبنای ۱۰ و از مبنای ۱۰ به مبنای ۸ ببرد. این کار را برای تبدیل بین مبنای ۱۰ و ۱۶ نیز تکرار کنید.

۲۷. کارنمایی رسم کنید که عمل تبدیل بین مبناهای ۲ و ۸ و ۱۶ را به یکدیگر انجام دهد. در این کارنما باید یک عدد و مبنای آن دریافت و معادل تبدیل یافتهٔ عدد در دو مبنای دیگر چاپ گردد.

۲۸. در مسئله ۱۹ از مجموعهٔ تمرین ۲-۱۱، از شما خواسته شده بود تا چرخ بازی را N بار بچرخانید. در اینجا می‌خواهیم مفهوم شبیه‌سازی بازی به کمک کامپیوتر را که به‌منظور پیش‌بینی نتیجه انجام می‌گیرد، به‌طوری جدّی‌تر ارائه دهیم. بازی موردنظر عبارت است از «بازی چرخ گردان». چنین تصور کنید که یک مقدار ورودی برای S، محل عقربه در شروع بازی و یک منبع تمام‌نشدنی از مقادیر برای تعداد قطاع‌های طی شده، m، به شما داده شده است.

این داده‌ها به نوعی نمایش‌گر آن چیزی هستند که یک شخص در صورتی که قرار باشد به دفعات و بدون دخالت بازیکن دیگری چرخ را بچرخاند، عملاً تجربه می‌کند. ما چنین تعریف می‌کنیم که یک «بازی» شامل یک سری چرخاندن توسط یک بازیکن معین است و پایان بازی موقعی است که قدرمطلق امتیازات وی، |SUM|، از یک مقدار بحرانی معین CV بیشتر شود. تعداد چرخاندن‌های انجام شده در سری مزبور را «طول» بازی می‌نامیم. سؤالی که واقعاً می‌خواهیم بکنیم این است: «قبل از آن که CV ≤ |SUM| شود، به‌طور متوسط چند بار می‌توان انتظار داشت که یک بازیکن چرخ را بچرخاند؟»

شما در این تمرین زمینه را، برای پاسخی که بعداً به سؤال فوق خواهید داد، آماده می‌کنید. وظیفهٔ فعلی شما این است که روندنمایی برای شبیه‌سازی یک بازی کامل ترسیم نمایید. در زیر خطوط کلی یک راه‌حل داده شده است ولی تا هنگامی که طرح ساخت خودتان را آزمایش نکرده‌اید به آن مراجعه نکنید.

الف- مانند تمرین‌های قبل، ابتدا چهار مقدار تشکیل‌دهندهٔ «قاعده محاسبهٔ امتیازات» را که باید از آن استفاده شود، بخوانید.

ب- بعد مقداری برای CV، مقدار بحرانی، بخوانید.

ج- سپس مقداری برای S و یک سری از مقادیر m را بخوانید.

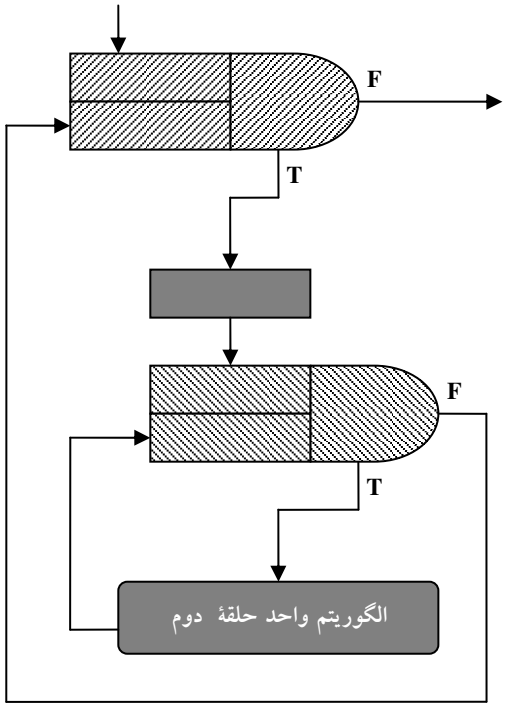
د- پس از خواندن هریک از مقادیر m ، مقدار جدید امتیازات حاصله (SUM) را محاسبه کرده و شمارندهٔ تعداد چرخاندن‌ها (L) را به هنگام در بیاورید.

و- هرگاه قدرمطلق SUM بیش از CV شود، مقادیر L ، CV و SUM را چاپ کرده و کار را متوقف کنید.

هـ- به‌منظور اجتناب از به‌وجود آمدن حلقهٔ بی‌پایان، هرگاه مقدار L بیش از ۱۰۰۰ شود ضمن چاپ یک پیغام خطا، کار را متوقف کنید. استفاده از جعبهٔ تکرار در جایی که به نظرتان موجب سادگی ساخت روندنما خواهد شد، از یاد نبرید.

۴-۵ : حلقه های تودرتو

اصطلاح حلقه‌های تودرتو[1] به الگوریتم‌هایی اطلاق می‌شود که مانند آنچه در شکل ۴-۲۲ نشان داده شده دارای حلقه‌ای در داخل حلقهٔ دیگر هستند. به‌عبارت دیگر اگر در داخل الگوریتم واحد یک حلقه از حلقهٔ تکرار دیگری استفاده شود به مجموعهٔ این حلقه‌ها «حلقه‌های تودرتو» می‌گویند.



شکل ۴- ۲۲ : نمایی از حلقه‌های تودرتو

کاربرد حلقه‌های تودرتو و مثال‌های متنوعی از این مبحث را پس از مطرح کردن مبحث متغیرهای لیستی و آرایه‌ها خواهید دید. اما در حال حاضر صرفاً جهت تکمیل مبحث حلقه‌ها، این بحث را مطرح ساختیم و بحث مفصل در خصوص کاربرد حلقه‌های تودرتو را به فصل آرایه‌ها موکول می‌سازیم. لذا در ادامه به طرح مثال مقدّماتی در این زمینه می‌پردازیم.

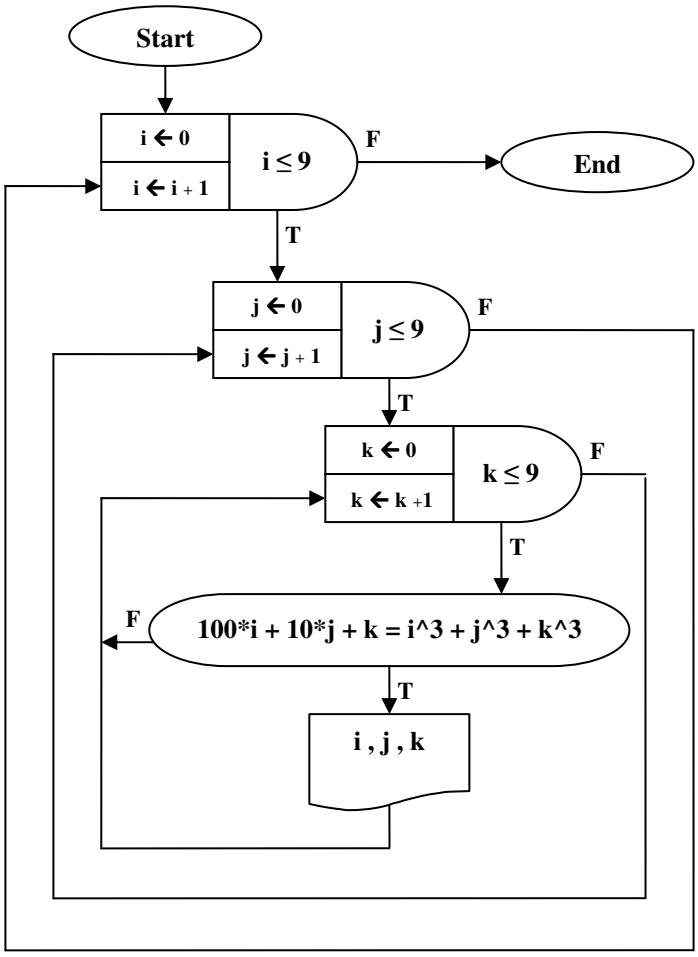
1. Nested loop

مثال ۴-۷ : کارنمایی که تمام اعداد محدودهٔ ۰۰۰ تا ۹۹۹ را که برابر مجموع مکعب‌های ارقام‌شان هستند پیدا کنید.

توضیح الگوریتم : اگر ارقام اعداد مورد نظر را با **i** و **j** و **k** نشان دهیم، آنگاه مسئله تبدیل می‌شود به پیدا کردن تمام ارقام سه‌گانهٔ مذکور که در شرط زیر صدق کنند.

$$100*i + 10*j + k = i^3 + j^3 + k^3$$

در حقیقت این تنها محاسبه‌ای است که در این الگوریتم انجام می‌شود. بقیهٔ الگوریتم مقادیر مختلف را برای متغییرهای **i** و **j** و **k** به واسطهٔ حلقه‌های تودرتو مهیا می‌کند. به‌ازای هربار گذر از حلقهٔ **i** ، حلقهٔ **j** به تعداد ده بار و به‌ازای مقادیر ۰ تا ۹ تکرار می‌شود. همچنین به‌ازای هربار گذر از حلقهٔ **j**، حلقهٔ **k** نیز به تعداد ۱۰ مرتبه اجرا می‌گردد. حال به کارنمای این الگوریتم توجه کنید.



شکل ۴- ۲۳ : کارنمای مثال ۴-۷

۴-۶: تمرین

شکل ۴-۲۳ را بررسی کنید و به سؤالات ۱ تا ۶ پاسخ دهید.

۱. از دکمهٔ شروع تا پایان چند عمل ضرب صورت می‌گیرد؟

۲. از دکمهٔ شروع تا پایان چند مقدار مختلف i^3 محاسبه می‌شود؟

۳. از دکمهٔ شروع تا پایان چند مقدار مختلف j^3 محاسبه می‌شود؟

۴. الگوریتم را طوری تغییر دهید که هیچ مقدار i^3 بیش از یک مرتبه و هیچ مقدار j^3 بیش از ده مرتبه محاسبه نشود.

۵. چگونه می‌توانیم الگوریتم را تغییر دهید به قسمی که هیچ مقدار i^3 ، j^3 و k^3 بیش از یک مرتبه محاسبه نشود؟

۶. با توجه به آن که عمل ضرب عملی وقت‌گیر برای کامپیوتر است، عملی پسندیده است که سعی کنید ضمن استفاده از حداکثر نه جعبه مختلف در روندنما، تعداد ضرب‌ها را به ۱۲۰ عدد تقلیل دهید. سپس در تلاش بعدی تعداد ضرب‌ها را به ۱۸ عدد کاهش دهید. آیا قادرید تعداد ضرب‌ها را از ۱۸ عدد هم کمتر کنید؟

۷. با استفاده از حلقه‌های تودرتو الگوریتمی طراحی کنید که یک اسکناس ۱۰۰ دلاری را به واسطهٔ اسکناس‌های ۵۰ ، ۲۰، ۱۰، ۵ و ۲ دلاری طوری خرد کند که تمام حالت‌های ممکن را شامل شود.

۸. الگوریتمی طراحی کنید که عددی را دریافت کند و بگوید آن عدد اول است یا خیر؟

۹. الگوریتمی طراحی کنید که مجموع N جملهٔ اول دنبالهٔ زیر را به‌ازای عدد حقیقی x محاسبه کند.

$$s = \frac{1}{x^1} - \frac{1}{x-2x^2} + \frac{1}{x-2x^2+3x^3} - \frac{1}{x-2x^2+3x^3-4x^4} + \cdots$$

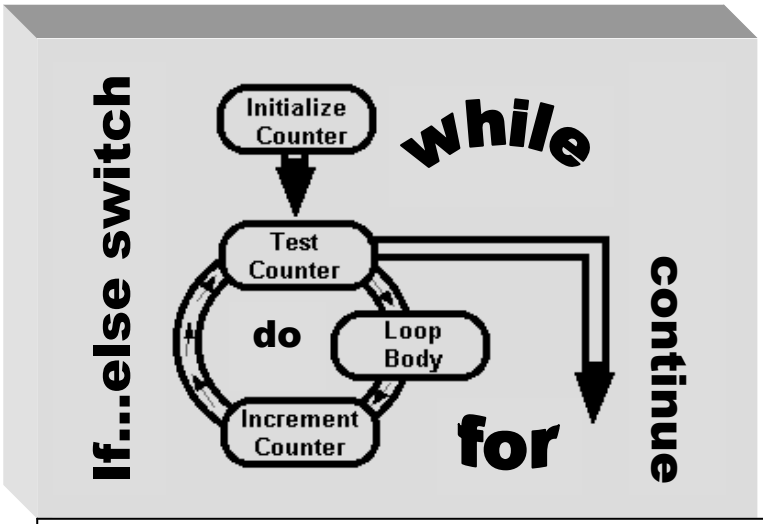
۱۰. الگوریتمی‌بنویسید که مقدار زاویهٔ x را برحسب رادیان بخواند و مقدار sin(x) را به واسطهٔ رابطهٔ زیر تا ۱۰ رقم اعشار تعیین کند. (برای تشخیص تعداد ارقام اعشار چه راه حلی را پیشنهاد می‌کنید!؟)

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

۱۱. فرض کنید سکه‌های رایج کشور عبارتند از ۱ و ۲ و ۵ و ۱۰ و ۲۵ و ۵۰ تومانی. الگوریتمی بنویسید که یک اسکناس N تومانی را دریافت کند و با توجه به این سکه‌ها آن اسکناس را با حداقل تعداد سکه خرد کند.

۱۲. منحنی $f(x) = x^2$ مفروض است. سطح تقریبی زیر منحنی را در فاصلهٔ a تا b به روش زیر محاسبه کنید. (توجه کنید که انتگرال‌گیری مد نظر نیست. اعداد a و b و N را از کاربر دریافت می‌کنید.)

فاصلهٔ a تا b را به N قسمت مساوی تقسیم‌کنید و با رسم خطوط موازی محور y ها N ذوزنقه به‌دست خواهدآمد. مساحت هر یک از ذوزنقه‌ها را حساب کنید و مجموع مساحت آنها را به‌عنوان سطح زیر منحنی چاپ کنید.



فصل پنجم

ساختارهای تکرار و تصمیم‌گیری در C++

اهداف فصل و چکیده مطالب:

۱. کاربرد ساختار تکرار for
۲. کاربرد ساختارهای تکرار while و do…while
۳. کاربرد دستورات break و continue
۴. آشنایی با دستور goto
۵. برنامه‌های ساخت‌یافته و غیر ساخت‌یافته
۶. کاربرد ساختار تصمیم‌گیری if
۷. کاربرد ساختار تصمیم‌گیری else if
۸. کاربرد ساختار switch
۹. شبیه‌سازی توابع gotoxy و clrscr
۱۰. تولید اعداد تصادفی و شبیه‌سازی توابع random و randomize

۵-۱ : ساختارهای تکرار در C++

مقدمه

بیشتر مطالب کلیدی لازم برای کارکردن با ساختارهای تکرار و تصمیم‌گیری را در فصول دوم و چهارم آموختید. لذا در این فصل سعی خواهیم کرد از تکرار مطالب قبلی خودداری کنیم، و بیشتر با ذکر مثال‌های متنوع به ذکر نکات برنامه نویسی در این زمینه بپردازیم. پس از آنکه بحث خود را در خصوص ساختارهای تکرار به پایان رساندیم، به بررسی برنامه‌های ساخت‌یافته و غیرساخت‌یافته خواهیم پرداخت.

به‌طور معمول دستورات برنامه باید از اولین دستور تا آخرین دستور به‌طور متوالی و پشت سرهم اجرا گردند. ولی چنانکه پیشتر نیز دیدید، گاهی نیازمند تکرار یک الگو و یا عدم انجام یک الگو تحت یک شرایط خاص و اجرای الگوی دیگری در شرایطی دیگر هستیم. لذا در تمامی زبان‌های برنامه‌نویسی دستوراتی جهت تصمیم‌گیری و یا ایجاد حلقه در برنامه وجود دارد. در C++ نیز دستورات متنوعی جهت عملیات تکرار و تصمیم‌گیری تدارک دیده شده که در ادامه به بررسی تک‌تک آنها خواهیم پرداخت.

ساختار تکرار for

پیشتر در فصل چهارم با ساختارهای تکرار دارای شمارنده آشنا شدید. دقیقاً ساختار تکرار **for** را می‌توان معادل ساختارهای تکرار دارای شمارنده در کارنماها در نظر گرفت. بنابراین هرگاه تعداد دفعات تکرار حلقه را از قبل بدانیم می‌توانیم از این ساختار استفاده کنیم. شکل کلی این ساختار به‌صورت زیر است:

```
for (مقدار اولیهٔ شمارنده ; شرط حلقه ; گام حرکت شمارنده )
    {
                دستور یکم ;
                    ⋮
                دستور n- اُم ;
    }
```

در این ساختار متغیری وجود دارد که شمارنده یا اندیس حلقه نام دارد، و تعداد دفعات تکرار حلقهٔ تکرار را در خود نگه می‌دارد. چنانکه می‌بینید در داخل پرانتز سه قسمت جداگانه وجود دارد که با سمیکالن از هم جدا می‌شوند. در اولین قسمت باید شمارندهٔ حلقه را مقدار اولیه داد. در قسمت دوم باید یک عبارت رابطه‌ای ساده و یا مرکب را به‌عنوان شرط حلقه قرار داد، و در آخرین قسمت باید به وسیلهٔ یک عبارت محاسباتی مقدار شمارنده را به میزان گام حلقه افزایش دهیم، و چنانکه می‌بینید دستورات الگوریتم واحد نیز به ترتیب از اولین دستور تا آخرین دستور بین دو نماد آکولاد باز و بسته قرار می‌گیرند، به عبارت دیگر دستورات مربوط به حلقه در داخل یک بلاک قرار می‌گیرند. قرار گرفتن دستورات حلقه در بلاک به ما این امکان را می‌دهد که متغیرهایی را در داخل خود بلاک تعریف کنیم به‌طوری که در بیرون از بلاک قابل شناسایی نباشند. در یک حلقهٔ تکرار که تنها دارای یک دستور است قرار دادن آکولاد باز و بسته الزامی نیست.

هنگامی‌که کنترل اجرای برنامه به دستور **for** می‌رسد ابتدا به اولین قسمت حلقه رفته و شمارنده را مقدار اولیه می دهد. متغیر شمارنده می‌تواند قبل از حلقه اعلان شده باشد. در این صورت می‌توان شمارنده را مقدار اولیه نداد و مقدار کنونی شمارنده را به‌عنوان مقدار اولیهٔ شمارنده پذیرفت، در نتیجه در این حالت هیچ چیزی در قسمت اول حلقه نمی‌نویسیم. همچنین می‌توان شمارنده را در همین قسمت تعریف کرد و مقدار اولیه داد. در این صورت برای اعلان یک شمارنده به نام **i** می‌توان به‌صورت زیر عمل کرد:

```
int i=0;
```

همچنین در صورت لزوم می‌توانید در این قسمت بیش از یک متغیر را تعریف کنید. در این صورت باید متغیرها را با علامت کاما از یکدیگر جدا کنید. متغیرهای تعریف شده در این قسمت برخلاف متغیرهای تعریف شده در بلاک حلقه، از علامت { به بعد نیز قابل شناسایی هستند. نکتهٔ قابل توجه آنکه قسمت اول حلقه تنها یک بار اجرا می‌گردد و در دفعات بعدی که حلقه تکرار می‌شود شمارنده مقداردهی اولیه نمی‌شود.

پس از مقدار اولیه دادن به شمارنده کنترل اجرای برنامه شرط حلقه را چک می‌کند و در صورت درست بودن این شرط دستورات حلقه به ترتیب از اولین دستور تا آخرین دستور اجرا می‌گردد. پس از پایان یافتن آخرین دستور حلقه، کنترل اجرای برنامه به قسمت گام حلقه رفته و عبارت محاسباتی آمده در این قسمت را اجرا می‌کند. دوباره شرط حلقه چک شده و در صورت درست بودن آن، دستورات داخل بلاک اجرا می‌گردد و در صورت نادرست بودن شرط حلقه کنترل اجرای برنامه به خط بعد از آکولاد بسته می‌رود.

درصورتی که در قسمت شرط حلقه هیچگونه عبارتی نوشته نشود، کامپایلر مقدار **true** را به‌طور پیش‌فرض به جای شرط حلقه قرار می‌دهد، که در این صورت حلقهٔ تکرار به یک حلقهٔ بی‌نهایت تبدیل می‌شود و متوقف نخواهد شد. برای خاتمه دادن به اجرای حلقهٔ تکرار بی‌نهایت باید دکمه‌های **Ctrl** و **Break** را به‌طور همزمان فشار داد. همچنین اگر در داخل قسمت شرط یک عبارت همیشه درست بنویسید، حلقهٔ بی‌نهایت ایجاد می‌شود.

مثال ۵-۱: در زیر نمونه‌های متنوعی را از تعریف حلقه‌های **for** مشاهده می‌کنید.

۱. ساده‌ترین تعریف از یک حلقهٔ **for** که ۱۰۰ بار اجرا می‌گردد:

```
for(int i=0;i<100;i++)
```

۲. شمارنده را از ۱۰۰ تا ۱۰ در گام‌هایی به طول ۲ کم می‌کند:

```
for(int i=100;i>=10;i=i-2)
```

۳. سه حلقهٔ **for** بی‌نهایت:

```
for(int i=1,j; ;i++,j*=i%k)
for(;;)
for(k=8;7<=12;k++)
```

۴. حلقهٔ تکرار با یک عبارت شرطی مرکب:

```
for(int i=10,j=75;
(i<=105&&j>50)||k!=42;
i++,k=k/i)
```

مثال ۵-۲ : برنامه‌ای که مربعات و مکعبات اعداد بین یک تا ۱۰ را چاپ می‌کند.

```
1.  //This program calculates square and cube of some numbers
2.  #include "iostream"
3.  using namespace std;
4.  int main()
5.  {
6.       for(int i=1;i<11;i++)
7.        cout<<i<<"\t"<<i*i<<"\t"<<i*i*i<<endl;
8.     return 0;
9.  }
```

توضیح مثال : در این برنامه تنها از یک حلقهٔ **for** ساده استفاده شده است. به دلیل وجود تنها یک دستور از قرار دادن آکولاد باز و بسته خودداری کردیم.

مثال ۵-۳ : برنامه‌ای که با دریافت یک عدد فاکتوریل آن را حساب می‌کند.

```
1.  //This program calculates factorial of an integer numbers
2.  #include <iostream>
3.  using namespace std;
4.  int main()
5.  {
6.     short n;
7.      cout<<"Please enter an integer number : ";
8.        cin>>n;
9.          unsigned int result=1;
10.           for(int fact=n;fact>0;fact--)
11.                result*=fact;
12.      cout<<"\nThe factorial of "<<n<<"is = "<<result;
13.    return  0;
14. }
```

مثال ۵-۴ : برنامه‌ای که به‌کارگیری شمارندهٔ اعشاری را در حلقهٔ **for** نشان می‌دهد.

```
1.  //This program shows how you can use float number as
2.  //index of loop.
3.  #include "iostream"
4.  using namespace std;
5.  int main()
6.  {
7.     int j=2;
8.      for(float f=5.0 ; f<100 ; cout<<f*j<<"\n" ,f *=1.5);
9.         {
10.             cout<<"In the block.\n";
11.        }
12.       cout<<"Out of  block.\n";
13.    return  0;
14. }
```

توضیح مثال : اصولاً در حلقه‌های **for** از شمارنده‌هایی از نوع **int** استفاده می‌شود. چرا که سرعت دستیابی به نوع **int** از هر نوع دیگری بیشتر است و این خود می‌تواند سرعت اجرای برنامه را افزایش دهد. پیشنهاد می‌شود حتی در مواردی که نوع **char** یا **short** برای شمارنده کافی است نیز از نوع **int** استفاده کنید زیرا تفاوت سرعت دستیابی حتی اگر چند میلیونیم ثانیه هم باشد با توجه به وجود تکرار در حلقه‌ها، می‌تواند در مجموع این تفاوت، تا چند ثانیه هم افزایش یابد. اما گاهی هم پیش می‌آید که به دلیل محاسبات داخلی حلقهٔ تکرار و به‌کار گرفته شدن شمارنده در این محاسبات مجبور هستیم شمارنده‌هایی از نوع اعشاری را به‌کار ببریم، لذا استفاده از انواعی غیر از نوع صحیح نیز به‌عنوان شمارندهٔ حلقه امکانپذیر است.

نکتهٔ قابل توجه دیگر در این مثال به‌کارگیریِ دستور خروجی **cout** در داخل خود حلقهٔ تکرار است. چنانکه می‌بینید در هر بار گذر از حلقه این دستور نیز اجرا می‌گردد. همچنین هرگاه پس از پرانتز بسته در دستور **for** علامت سمیکالن قرار دهیم دستورات داخل بلاک بی‌اعتبار شده و در تکرارهای حلقه اجرا نمی‌گردند. اما پس از خارج شدن از حلقه دستورات داخل بلاک تنها یکبار اجرا می‌شود. با توجه به نکات مطرح شده خروجی این برنامه به‌صورت زیر است:

```
10
15
22.5
33.75
50.627
75.9375
113.906
170.859
In the block.
Out of block.
```

مثال ۵-۵ : برنامه‌ای که جدول کدهای اسکی را به همراه نماد مربوط به هر کد چاپ می‌کند.

```
1.  //This program prints ASCII code table.
2.  #include "iostream.h"
3.  int main()
4.  {
5.    for(char ch=-128;ch<127;ch++)
6.     cout<<"character="<<ch
7.         <<"\tASCII Code="<<(int)ch<<"\n";
8.    cout<<"character="<<ch<<"\tASCII Code="<<(int)ch<<endl;
9.    return 0 ;
10. }
```

توضیح مثال : برای به‌دست آوردن کد اسکی داخل یک متغیر از نوع **char** باید آن متغیر را به یک عدد از نوع **int** نسبت داد، و یا آنکه مانند مثال فوق از تبدیل نوع استاتیک استفاده کرد. نکتهٔ دیگر آنکه چون دامنهٔ اعداد **char** از ۱۲۸- تا ۱۲۷ می‌باشد باید کلیهٔ کاراکترهای این بازه را چاپ کنیم، اما اگر شرط حلقه را به‌صورت **(ch<=127)** بنویسیم با حلقهٔ بی‌نهایت روبرو خواهیم شد.!؟ لذا مجبور به چاپ کد اسکی ۱۲۷ در یک خط جداگانه (خط هشتم) شده‌ایم. این برنامه را اجرا کنید و خروجی آن را ببینید.

حلقه‌های تودرتو

چنانکه می‌دانید اگر در داخل بدنهٔ یک حلقه از حلقهٔ دیگری استفاده شود می‌گوییم که حلقه‌های تودرتو ایجاد شده‌اند. برای به‌کارگیری یک حلقهٔ **for** در داخل حلقهٔ **for** دیگر بهتر است نام شمارنده‌ها را متفاوت در نظر بگیریم. چنانکه حلقهٔ داخلی تنها دستور حلقهٔ خارجی باشد، نیازی به قرار دادن آکولاد باز و بسته برای حلقهٔ خارجی نیست. در زیر مثال ساده‌ای از کاربرد حلقه‌های تودرتو را می‌بینید.

مثال ۵-۶: برنامه‌ای که جدول ضرب را در صفحه نمایش چاپ می‌کند.

```
1.  //This program generates multiplication chart.
2.  #include "iostream"
3.  using namespace std;
4.  int main()
5.  {
6.        for(int i=1;i<=10;i++)
7.          {
8.             for(int j=1;j<=10;j++)
9.                  cout<<i*j<<"\t";
10.            cout<<endl;
11.          }
12.    return 0;
13. }
```

توضیح مثال: در این برنامه از حلقه‌های تودرتو استفاده شده است. به ازای هر بار تکرار حلقهٔ **i**، حلقهٔ درونی به تعداد ۱۰ بار اجرا می‌گردد. در خط نهم برای منظم کردن خروجی پس از چاپ هر عدد به میزان هشت خانه به جلو می‌رویم. همچنین پس از چاپ شدن هر ۱۰ عدد خروجی در یک خط، در خط دهم کد دستکاری‌کنندهٔ **endl** خط جاری را رد می‌کند، تا ده عدد بعدی در خط جدیدی چاپ شود.

کار در کلاس ۵-۱:

به نظر شما خروجی برنامهٔ زیر چیست؟

```
1.  //This program shows how we use counter in the complex loop.
2.  #include "iostream"
3.  using namespace std;
4.  int main()
5.  {
6.          for(int i=1;i<=10;i++){
7.              for(int i=1;i<=10;i++)
8.                  cout<<i*i<<"\t";
9.              cout<<"\n";
10.           }
11.    return 0 ;
12. }
```

ساختار تکرار while

با نماد حلقه‌های **while** در الگوریتم‌نویسی در فصل قبل آشنا شدید. در **C++** نیز ساختاری با نام **while** برای ایجاد حلقه‌های دارای نگهبان وجود دارد. وقتی کنترل اجرای برنامه به دستور **while** می‌رسد، ابتدا شرط حلقه چک شده و در صورت درست بودن شرط، اجرای دستورات داخل بلاک آغاز می‌شود، و تا زمانی‌که شرط حلقه دارای ارزش درستی باشد، این روند ادامه پیدا می‌کند. شکل کلی این دستور به‌صورت زیر است:

```
while ( شرط حلقه )
      {
            دستور یکم ;
                ⋮
            دستور n-اُم ;
      }
```

در این ساختار نیز اگر تنها یک دستور در بدنهٔ حلقه داشته باشیم، نیازی به قرار دادن آکولاد باز و بسته نیست، و همچنان‌که می‌دانید در حلقه‌هایی که از مکانیسم نگهبان استفاده می‌کنند، برای توقف حلقه باید شرایطی را مهیا کرد که شرط حلقه در داخل خود حلقه نقض گردد. اگر شرط نقض نشود با حلقهٔ بی‌نهایت روبرو می‌شویم.

مثال ۵-۷ : برنامه‌ای که جملات دنبالهٔ فیبوناچی را تا آنجایی که ممکن باشد محاسبه می‌کند. این برنامه قبل از آنکه با سرریز شدن داده مواجه شود، متوقف می‌گردد.

```
1.  //This program uses WHILE loop for Fibonacci series.
2.  #include <iostream>
3.  #include <limits.h>
4.  using namespace std;
5.  int main()
6.  {
7.      unsigned long next=0,last=1;
8.         while(next < ULONG_MAX / 2)
9.         {
10.              cout<<last<<endl;
11.               unsigned long sum=next+last;
12.                 next=last;
13.                 last=sum;
14.           }
15.     return 0;
16. }
```

توضیح مثال : چنانکه می‌بینید شرط حلقهٔ تکرار را به‌گونه‌ای انتخاب کرده‌ایم که قبل از بروز سرریز محاسبات متوقف گردد. مقدار **ULONG_MAX** در سرفایل **limits.h** تعریف شده است. نکتهٔ دیگر آنکه تعریف یک متغیر در داخل یک حلقهٔ تکرار موجب نمی‌شود، هر بار که کنترل اجرای برنامه به اعلان متغیر می‌رسد متغیر جدیدی تعریف کند. بلکه تنها در اولین اجرای حلقه متغیر **sum** تعریف می‌شود.

مثال ۵-۸ : برنامه‌ای که دو عدد را به‌عنوان صورت و مخرج یک کسر متعارفی دریافت کرده و خارج قسمت و باقی‌ماندهٔ آن دو عدد را به واسطهٔ عملگرهای / و % به خروجی می‌برد. به نحوهٔ کنترل حلقهٔ تکرار توجه کنید.

```
1.  //This program demonstrates WHILE loop.
2.  #include <iostream>
3.  using namespace std;
4.  int main()
5.  {
6.    long int dividend, divisor;
7.     char ans='y';
8.      while(ans=='y'||ans=='Y')
9.       {
10.       cout<<"Enter dividend : ";
11.         cin>>dividend;
12.       cout<<"Enter divisor : ";
13.         cin>>divisor;
14.       cout<<"Quotient is : "<<dividend / divisor
15.           <<" ,reminder is : "<<dividend % divisor<<endl;
16.       cout<<"Do you want to try again?(Y/N):";
17.         cin>>ans;
18.      }
19.   return  0 ;
20. }
```

توضیح مثال : تنها نکتهٔ قابل توجه در این برنامه خطوط ۱۴ و ۱۵ است که به شما نشان می‌دهد می‌توانید یک دستور را در بیش از یک خط نیز بنویسید چرا که کامپایلر C++ نسبت به فضاهای خالی بی‌توجه است.

ساختار تکرار do...while

ساختار تکرار **do...while** دقیقاً مانند ساختار تکرار **while** عمل می‌کند با این تفاوت که شرط حلقه در پایان حلقه می‌آید، بنابراین دستورات داخل بلاک حداقل یکبار قبل از چک شدن شرط حلقه اجرا می‌گردد. شکل کلی این ساختار به‌صورت زیر است:

```
do {
        دستور یکم ;
            ⋮
        دستور n-اُم ;
   } while ( شرط حلقه ) ;
```

در این ساختار نیز هرگاه تنها یک دستور داشته باشیم نیازی نیست که آکولاد باز و بسته قرار دهیم. اما نکتهٔ بسیار مهمی که اکثر دانشجویان آن را فراموش می‌کنند علامت سمیکالنی است که در انتهای این دستور آمده است.

مثال ۵-۹: برنامه‌ای که تعداد نامشخصی عدد را از ورودی خوانده مقلوب آنها را به خروجی می‌برد. به چگونگی به‌کارگیری حلقه‌های تودرتو در این مثال توجه کنید.

```
1.  //This program calculates inverted of some numbers.
2.  #include <iostream>
3.  using namespace std;
4.  int main()
5.  {
6.     int number, digit;
7.      while(true)
8.        {
9.           cout<<"\nEnter an integer number for convert :";
10.            cin>>number;
11.          cout<<"Inverse is : ";
12.             do{
13.                  digit=number % 10;
14.                     cout<<digit;
15.                   number /=10;
16.                }while(number!=0);
17.         }//end of first loop
18.    return 0;
19. }
```

توضیح مثال: چنانکه می‌بینید در این برنامه از حلقه‌های تودرتو استفاده شده است. حلقهٔ بیرونی دارای شرطی است که همواره درست است، بنابراین یک حلقهٔ تکرار بی‌نهایت را ساخته و برای پایان اجرای این حلقه باید دکمه‌های **Ctrl+Break** را بفشارید. برای نمایش مقلوب عدد نیز تک‌تک ارقام آن را از سمت راست به‌دست آورده و چاپ می‌کنیم، در نتیجه تنها مقلوب عدد در صفحه چاپ می‌شود. آیا می‌توانید مثال فوق را به‌گونه‌ای تغییر دهید که ابتدا عدد مقلوب را در داخل یک متغیر صحیح ذخیره کند و در نهایت آن متغیر را چاپ کند.

چه وقت بهتر است از کدام یک از این حلقه‌ها استفاده کنیم؟

اگر از قبل تعداد دفعات مورد نیاز اجرای حلقه را بدانیم، بهتر است از ساختار **for** برای ایجاد حلقه استفاده کنیم. گرچه می‌توان از حلقهٔ **while** نیز برای ایجاد حلقهٔ تکرار با شمارنده استفاده کرد!؟ اما از انجام این کار به شدّت خودداری کنید، زیرا از خوانایی برنامه می‌کاهد. اما هرگاه تعداد دفعات تکرار حلقه را ندانیم باید به دنبال یک نگهبان برای ساخت شرط حلقه باشیم تا بتوانیم از یکی از دو ساختار **while** یا **do...while** استفاده کنیم. همچنین هرگاه نیاز داشته باشیم دستورات داخل بلاک حلقه حداقل یک بار اجرا گردد از ساختار **do...while** استفاده می‌کنیم، در غیر این صورت از ساختار **while** باید استفاده کرد.

نکتهٔ پایانی که باید متذکر شویم آن که بسیاری از دانشجویان به اشتباه به جای علامت **&&** در قسمت شرط حلقهٔ **for** از علامت کاما استفاده می‌کنند، و یا آنکه به جای علامات سمیکالن در حلقه‌های **for** از علامت کاما استفاده می‌کنند. هر دوی این موارد را در یکی از مثال‌های فوق اعمال کنید و خطای صادر شده را مشاهده نمایید.

۵-۲: انتقال کنترل غیر شرطی در C++

در C++ دو دسته دستور وجود دارند که به واسطهٔ آنها می‌توان کنترل اجرای برنامه را به خطوط مختلف کد منتقل کرد. دسته‌ای از این دستورات با چک کردن شرط یا شروطی مبادرت به انتقال کنترل اجرای برنامه می‌کنند که به ساختارهای تصمیم‌گیری یا انتقال کنترل شرطی معروف هستند، و دسته‌ای دیگر بدون چک کردن هیچ شرطی کنترل اجرای برنامه را به نقطه‌ای خاص انتقال می‌دهند که به کنترل غیرشرطی موسوم شده‌اند. در این قسمت به معرفی این کنترل‌ها می‌پردازیم اما مثال‌های متنوع آنها را در صفحات آتی خواهید دید.

دستور break

این دستور در داخل حلقه های تکرار قرار گرفته و موجب خروج از حلقهٔ تکرار می شود. نحوهٔ کاربرد این دستور به‌صورت زیر است:

```
break;
```

اگر چند حلقهٔ تودرتو وجود داشته باشد این دستور موجب خروج از داخلی‌ترین حلقه می‌شود. کاربرد دیگر این دستور در ساختار **switch** است که در ادامه بحث خواهد شد.

دستور continue

این دستور نیز در حلقه‌های تکرار به‌کار می‌رود و موجب انتقال کنترل اجرای برنامه به ابتدای حلقهٔ تکرار می‌شود. این دستور به‌صورت زیر به‌کار می‌رود:

```
continue;
```

پس از انتقال کنترل به ابتدای حلقه، شرط حلقه چک شده و در صورت درست بودن شرط مذکور، اجرای دستورات داخل حلقه از اولین دستور بلاک از سرگرفته می‌شود. به عبارت دیگر اگر دستور یا دستوراتی پس از دستور **continue** وجود داشته باشد هیچگاه اجرا نخواهند شد. بنابراین اصولاً هر دو دستور **break** و **continue** باید در داخل ساختارهای تصمیم به‌کار گرفته شوند در غیر این صورت اجرای دستورات حلقهٔ تکرار با مشکل مواجه خواهد شد.

مثال ۵-۱۰: در زیر برنامه‌ای ارائه شده که عملکرد دستور **continue** را نشان می‌دهد.

```
1.  //This program demonstrates continue statement.
2.  #include <iostream.h>
3.  void main()
4.  {
5.    int i=0;
6.     do{
7.             i++;
8.              cout<<"before the continue\n";
9.               continue;
10.          cout<<"after the continue, should never print\n";
11.       }while(i<3);
12.    cout<<"after the do loop\n";
13. }
```

توضیح مثال : در این برنامه با توجه به‌وجود دستور **continue** خط دهم برنامه هیچ‌گاه اجرا نخواهد شد، و خروجی این برنامه به‌صورت زیر خواهد بود:

```
before the continue
before the continue
before the continue
after the do loop
```

دستور goto

این دستور سبب انتقال کنترل اجرای برنامه به نقطه‌ای می‌شود که با یک برچسب[1] مشخص شده است. روش کاربرد این دستور به‌صورت زیر است:

```
goto    برچسب ;
```

برچسب عبارت است از یک نام دلخواه که برای انتخاب آن باید از روش نامگذاری متغیرها پیروی کرد. هر خطی را که بخواهیم برچسب‌گذاری کنیم باید در ابتدای آن خط نام برچسب را بیاوریم و سپس یک علامت کولن (:) پس از نام برچسب قرار دهیم. به‌عنوان مثال می‌توان هریک از عبارات زیر را به‌عنوان برچسب در جلوی یک خط از برنامه به‌کار ببریم:

```
Loop: یا  Label: یا  G1:
```

به دلیل این که استفاده از دستور **goto** می‌تواند برنامه را از ساخت‌یافتگی خارج کند، استفاده از این دستور در تمامی منابع برنامه‌نویسی نهی شده است. حتی برای ایجاد حلقه‌های تکرار هم به هیچ‌عنوان نباید از دستور **goto** استفاده نمود. لذا ما نیز به دلیل عدم کاربرد این دستور در برنامه‌های ساخت‌یافته به آن نمی‌پردازیم.

کار در کلاس ۵-۲ :

با استفاده از دستور **goto** *برنامهٔ معادل کارنمای شکل ۴-۳ را به زبان* **C++** *بنویسید.*

1. Label

۵-۳ : ساختارهای تصمیم در C++

در هر زبان برنامه‌نویسی ساختارهایی را جهت اعمال تصمیم گیری در برنامه‌ها که معادل جعبه‌های تصمیم در کارنما هاست فراهم می‌کنند. در C++ نیز تعدادی ساختار جهت اِعمال تصمیم‌گیری در برنامه‌ها فراهم شده است که در ادامه به بررسی این ساختارها می‌پردازیم.

ساختار تصمیم‌گیری if

ساختار تصمیم گیری **if** دارای دو شکل کلی است. چنانکه در زیر می‌بینید در حالت اول اگر شرط ساختار **if** دارای ارزش درستی باشد دستورات داخل بلاک اجرا می‌گردد و در صورت نادرست بودن شرط مذکور، کنترل اجرای برنامه به بعد از علامت } می‌رود. اما حالت دیگر نحوهٔ عملکرد این ساختار بدین گونه است که اگر شرط ساختار **if** درست باشد دستورات داخل بلاکِ پس از دستور **if** اجرا می‌گردد، و در صورت نادرست بودن شرط، دستورات داخل بلاکِ پس از کلمهٔ **else** اجرا می‌گردد. فرم کلی این ساختار به‌صورت‌های زیر است:

أ- صورت کلی اول:

```
if ( شرط )
    {
            دستور یکم ;
                ⋮
            دستور n-اُم ;
  }
```

ب- صورت کلی دوم:

```
if ( شرط )
    {
            دستور یکم ;
                ⋮
            دستور n-اُم ;
  }
else
    {
            دستور یکم;
                ⋮
           دستور n-اُم ;
    }
```

در این ساختار نیز در صورت وجود تک دستور در یک بلاک نیازی به قرار دادن آکولاد باز و بسته نیست.

مثال ۵-۱۱ : برنامه‌ای که از کاربر رمز ورود به برنامه را می‌خواهد و درصورتی که رمز به‌طور صحیح وارد شود پیغام خوش‌آمد دریافت می‌کند.

```
1.  //This program demonstrates IF statement.
2.  #include "iostream"
3.  using namespace std;
4.  int   main ( )
5.  {
6.     const int password=62541893;
7.      int ans;
8.       cout<<"Please enter the password for login : ";
9.        cin>>ans;
10.        if(ans==password)
11.         {
12.            cout<<"Welcome to MJZ program.\n";
13.              return 0;
14.         }
15.      cout<<"Unfortunately you entered a wrong password.";
16.       cout<<"\nLogin failed.\n";
17.    return 0;
18. }
```

توضیح مثال : اولین نکتهٔ بسیار مهم در این مثال تعریف یک متغیر از نوع صحیح به‌صورت ثابت است. که تضمین می کند مقدار **password** در طول برنامه سهواً تغییر نمی‌کند.

نکتهٔ قابل توجه در این برنامه وجود دو دستور **return** در خطوط ۱۳ و ۱۷ برنامه است. چنانکه در فصل سوم در خصوص دستور **return** عنوان شد این دستور نه تنها مقداری را به فراخواننده تابع برمی‌گرداند بلکه کنترل اجرای برنامه را نیز به فراخوانندهٔ تابع برمی‌گرداند. لذا اگر دستوری پس از دستور **return** قرار داشته باشد هیچ‌گاه اجرا نخواهد شد. درصورتی که کاربر رمز صحیح، یعنی عدد **62541893** را وارد کند، رمز مورد قبول واقع شده و پیغام **"Welcome to MJZ program."** را در صفحه می‌بیند و با رسیدن به دستور **return** کنترل اجرای برنامه نیز به سیستم عامل برگردانده می‌شود و لذا پیغام خطا چاپ نخواهدشد. اما در صورت واردکردن هر عدد دیگری پیغام زیر در صفحه نمایش مشاهده می‌شود.

"Unfortunately you entered a wrong password. Login failed."

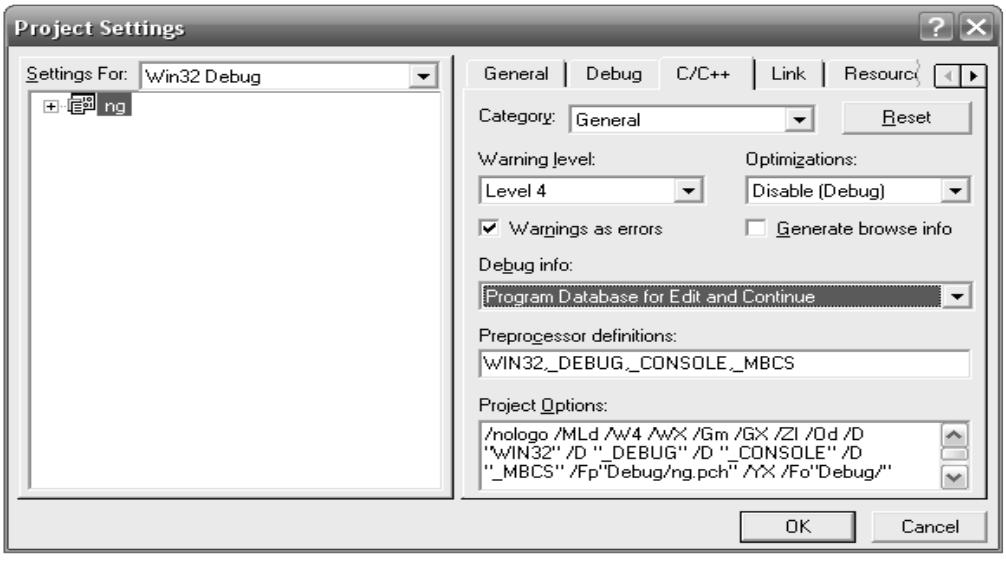
کار در کلاس ۵-۳ :

مثال ۵-۱۱ را یک بار دیگر به واسطهٔ **if...else** *به‌گونه ای بازنویسی کنید که نیازی به دو دستور* **return** *نداشته باشد.*

تذکر مهم: بسیاری از دانشجویان به اشتباه در عبارت شرطی مربوط به دستور **if** به جای استفاده از علامت == برای نشان دادن برابری از علامت مربوط به انتساب یعنی= استفاده می‌کنند. مسلماً چنین عملی موجب می شود با نتایج غیرمنتظره‌ای روبرو بشویم. به‌عنوان یک آزمایش در خط دهم همین برنامه علامت == را با علامت = جایگزین کنید و سپس یکبار دیگر برنامه را کامپایل و اجرا کنید. چه نتیجه‌ای مشاهده می‌کنید؟

نتیجه‌ای که مشاهده خواهید کرد این است که با ورود هرگونه عدد شما پیغام خوش‌آمد را مشاهده می‌کنید. یعنی رمزی که ما برای ورود به برنامهٔ خود گذاشتیم هیچگاه عمل نمی‌کند. *این در یک برنامهٔ بزرگ یعنی یک فاجعه!* در حقیقت یک عمل انتساب صورت می‌گیرد و در صورت موفقیت‌آمیز بودن این عمل مقدار **true** به‌جای عبارت انتساب قرار می‌گیرد، و از آنجا که عمل انتساب همواره با موفقیت همراه است همیشه شرط دستور **if** برقرار است و در نتیجه پیغام خوش‌آمد چاپ می‌شود.

اما متأسفانه اگر دقت کرده باشید کامپایلر حتی هیچگونه هشداری هم در خصوص این خطا نمی‌دهد. با این اوصاف چگونه می‌توان چنین مشکل بزرگی را در یک برنامهٔ چند هزار خطی تشخیص داد؟ جواب این سؤال در تنظیمات کامپایلر است. برای این که کامپایلر بتواند چنین خطاهایی را تشخیص دهد باید سطح هشداردهی کامپایلر را بر روی **maximum warning** قرار دهید. نحوهٔ انجام این تنظیم در کامپایلرهای مختلف متفاوت است. برای این منظور می‌توانید در مستندات مربوط به کامپایلر خود، عبارت **warning level** را جستجو کنید. اما در محیط **Visual Studio 6.0** می‌توانید از طریق انتخاب گزینهٔ **setting** از منوی **project** پنجره‌ای مطابق شکل ۵-۱ موسوم به پنجرهٔ **Project Setting** را باز کنید. در این پنجره صفحهٔ **C/C++** را انتخاب کنید. در این صفحه سطح هشدارها را برروی **level 4** تنظیم کنید و جلوی گزینهٔ **warning as error** علامت تیک بزنید. حال یکبار دیگر کد قبلی را کامپایل کنید این بار چه نتیجه‌ای می‌بینید.



شکل ۵-۱: پنجرهٔ تنظیمات سطح هشدار

مثال ۵-۱۲: برنامه‌ای که با دریافت یک پاراگراف تعداد حروف و کلمات آن را می‌شمارد. انتهای ورود داده ها نیز با زدن کلید **enter** مشخص می‌شود.

```
1.  //This program counts the number of words and characters.
2.  #include <iostream>
3.  #include <conio.h>  //included for getche() and getch()
4.  using namespace std;
5.  int main()
6.  {
7.     int char_counter=0, word_counter=0;
8.      char ch=0;
9.     cout<<"Enter a paragraph(and press Enter for end):\n";
10.      while((ch=getche())!='\r')
11.        {
12.            char_counter++;
13.             if(ch==' ')
14.                word_counter++;
15.         }//end of while
16.      cout<<"\nChar count= "<<char_counter
17.          <<" ,Word count= "<<word_counter+1;
18.     getch();
19.    cout <<endl;
20.   return 0;
21. }
```

توضیح مثال : در این برنامه توسط تابع **getche** شروع به گرفتن کاراکترهای ورودی می‌کنیم. این تابع در هدر فایل **conio.h** قرار دارد و یک کاراکتر را از کاربر گرفته و در صفحه نمایش نشان می‌دهد. هر بار که حلقه اجرا می‌شود ابتدا این تابع اجرا می‌گردد. همچنین تابع **getche** کداسکی کاراکتر دریافتی را برمی‌گرداند. لذا در این مثال کداسکی برگردانده شده توسط این تابع، را به داخل یک متغیر از نوع **char** ریخته‌ایم. کاراکتر کنترلی **\n** کلید **enter** را مشخص می‌کند. کداسکی کلید **enter** برابر ۱۳ می‌باشد و تا زمانی که این کداسکی توسط تابع برگردانده نشود حلقه به تکرار خود ادامه می‌دهد. می‌توان به جای **'\n'** عدد **13** را قرار داد. هر کاراکتر که خوانده می‌شود یک واحد به شمارندهٔ حروف اضافه می‌گردد. اما از آنجا که کلمات با فضای خالی (**space**) از یکدیگر جدا می‌شوند در خطوط ۱۳ و ۱۴ برنامه درصورتی که کاراکتر ورودی برابر فضای خالی باشد یک واحد نیز به شمارندهٔ کلمات اضافه می‌شود. اما آخرین نکته در خصوص این برنامه قرار دادن تابع **getch** در انتهای برنامه است. این تابع نیز یک کاراکتر را از کاربر می‌گیرد ولی در صفحه چاپ نمی‌کند. برای جلوگیری از خاتمهٔ برنامه و ایجاد یک وقفه جهت مشاهدهٔ نتایج از این تابع استفاده کرده‌ایم. با آنکه در کامپایلر **VC6** یک عبارت **press any key to continue** پس از اجرای برنامه قرار داده می‌شود، و شما می‌توانید نتایج را به‌راحتی مشاهده کنید، اما در صورت اجرا کردن برنامه‌ها پس از مرحلهٔ کامپایل این عبارت در انتهای برنامه نمی‌آید و در نتیجه برنامه با رسیدن به دستور **return** خاتمه می‌یابد. لذا این دستور یک وقفه قبل از اجرای دستور **return** ایجاد می‌کند. هنگام استفاده از سرفایل **conio.h** در کامپایلر **VC6** به هیچ عنوان از **iostream.h** استفاده نکنید.

مثال ۵-۱۳: برنامه‌ای که مثال ۵-۱۱ را به‌واسطهٔ توابع **getch** و ساختار **if...else** پیاده‌سازی می‌کند.

```
1.  //This program demonstrates IF…ELSE statement.
2.  #include "iostream"
3.  #include <conio.h>
4.  using namespace std;
5.  int main()
6.  {
7.     const int password=62541893;
8.      int ans=0;
9.       char ch=0;
10.       cout<<"Please enter the password for login : ";
11.        while((ch=getch())!= 13)
12.            if(ch>='0' && ch<='9')
13.              {
14.                 cout.put('*');
15.                 ans *=10;
16.                 ans +=(ch-48);
17.              }
18.    if(ans==password)
19.      cout<<"\nWelcome to MJZ program.\n";
20.    else
21.     cout<<"\nUnfortunately you entered a wrong password."
22.           <<"\nLogin failed.\n";
23.   return 0;
24. }
```

توضیح مثال: با عبارتی که در خط ۱۱ آمده در مثال قبل آشنا شدید. اما به جهت آنکه می‌خواهیم رمز وارد شده توسط کاربر برروی صفحه نمایش چاپ نشود از تابع **getch** استفاده کرده‌ایم. این تابع نیز کد اسکی کلید فشرده شده را برمی‌گرداند و در نتیجه این کد داخل **ch** قرار می‌گیرد. در خط ۱۲ چک می‌شود که اگر کد اسکی داخل کاراکتر **ch** شامل کداسکی یکی از ارقام **0** تا **9** می‌باشد، وارد بلاک مربوط به دستور **if** بشویم. توجه کنید که ارقام **0** تا **9** هنگامی که به‌صورت یک کاراکتر فرض می‌شوند دارای کدهای اسکی بین ۴۸ تا ۵۷ هستند لذا در خط ۱۶ برای رسیدن به رقم ریاضی معادل کاراکترهای **0** تا **9** مقدار ۴۸ را از کاراکتر **ch** کم کرده‌ایم. اما قبل از آن، در خط ۱۵ متغیر **ans** را در ۱۰ ضرب کرده‌ایم، تا به این واسطه هنگامی که در خط ۱۶ رقم وارد شدهٔ جدید را با **ans** جمع می‌کنیم این رقم در منتها الیه سمت راست عدد منظور شود.

اما در خط ۱۴ پس از آنکه کاربر یک رقم را به‌عنوان ورودی وارد کرد یک علامت * برروی صفحه چاپ می‌کنیم تا کاربر بداند رقم وارد شده با موفقیت ثبت شده است. این درحالی است که اگر به‌عنوان مثال کاربر حرفی را واردکند علامت * چاپ نمی‌شود. برای چاپ این علامت از تابع **put** که یک تابع عضو شئ **cout** می‌باشد استفاده کرده‌ایم. این تابع با دریافت کد اسکی یک کاراکتر آن کاراکتر را برروی صفحه چاپ می‌کند، و دارای شکل کلی زیر است:

cout.put (کد اسکی یک کاراکتر **);**

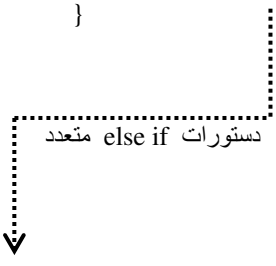
ساختار تصمیم‌گیری else if

اگر در داخل قسمت **else** یک دستور **if**، دستور **if** جدیدی به‌کار ببریم به ساختار **else if** می‌رسیم. این ساختار به ما کمک می‌کند تا شروط مختلفی را چک کنیم. شکل کلی این دستور به‌صورت زیر است:

```
if ( شرط ۱ )
 {
      نخستین بلاک
  }
else
    if ( شرط ۲ )
     {
        دومین بلاک
      }
     else
        {
            آخرین بلاک
         }
```

هنگامی که کنترل اجرای برنامه به اولین دستور **if** می‌رسد شرط آن چک می‌شود، اگر شرط دارای ارزش درستی باشد دستورات داخل نخستین بلاک اجرا شده و سپس کنترل اجرای برنامه به بعد از آکولاد بستهٔ آخرین بلاک می‌رود. اما اگر شرط درست نباشد به قسمت **else** آمده و براساس درست یا غلط بودن شرط ۲ یکی از بلاک‌های دوم یا سوم اجرا می‌گردد. این روند یعنی قرار دادن یک دستور **if** در داخل قسمت **else** یک دستور **if** دیگر می‌تواند تا هر جایی که نیاز ما را برطرف سازد و تمامی شروط ما را پوشش دهد ادامه یابد. اصولاً دو دستور **else** و **if** را در ساختار فوق در یک خط می‌نویسند تا به خوانایی برنامه افزوده شود. بنابراین داریم:

```
if ( شرط ۱ )
 {
      نخستین بلاک
  }
else if ( شرط ۲ )
     {
        دومین بلاک
      }
```




```
else
   {
      آخرین بلاک
    }
```

پاک کردن صفحه نمایش

در کامپایلرهای شرکت بورلند امکان استفاده از چهار تابع مفید و پرکاربرد وجود دارد که در VC6 امکان استفاده از آنها مهیا نیست، چرا که کامپایلر VC6 از این چهار تابع که به ترتیب عبارتند از: clrscr، gotoxy، random و randomize پشتیبانی نمی‌کند. از بین این چهار تابع دو تابع اول بسیار مهم و دارای کاربردهای متعددی هستند، که نمونه‌ای از کاربرد آنها را در همین فصل خواهید دید و در مورد کاربرد دو تابع دوم اندکی بعد صحبت خواهیم‌کرد. با توجه به احساس نیاز برنامه‌نویسان به این توابع تلاش‌هایی برای پیاده‌سازی این توابع در VC6 صورت گرفته که در این قسمت به ذکر آنها برای پیاده‌سازی clrscr و gotoxy می‌پردازیم. ذکر جزئیات و عملکرد کدهای ارائه شده در این قسمت فراتر از معلومات فعلی شماست. لذا فعلاً تنها به نحوهٔ پیاده‌سازی این توابع و چگونگی به‌کارگیری و استفادهٔ این توابع توجه کنید.

در کامپایلرهای شرکت بورلند با اضافه کردن سرفایل conio.h قادر به استفاده از تابع clrscr برای پاک کردن صفحه نمایش هستید. این تابع به‌صورت زیر به‌کار می‌رود:

```
clrscr();
```

هنگامی که کنترل اجرای برنامه به این تابع می‌رسد محتویات صفحه نمایش را پاک می‌کند. اما اگر چنین دستوری را در داخل VC6 بنویسید با خطا روبرو می‌شوید. برای آنکه بتوانید این دستور را در VC6 نیز همانند کامپایلرهای بورلند استفاده کنید مراحل زیر را باید انجام دهید.

۱. پیش از تابع main و بعد از دستور using خط زیر را تایپ کنید:

```
void clrscr(void);
```

۲. پس از آکولاد بستهٔ تابع main نیز کد زیر را تایپ کنید:

```
void clrscr()
{
    system("cls");
}
```

در هر کجای کد هر برنامه‌ای که در حالت کنسول نوشته می‌شود می‌توانید دستورات سیستم عامل Dos را توسط تابع system به خط فرمان فرستاده، و اجرا کنید. به‌عنوان مثال در عبارت فوق دستور cls که یکی از فرامین سیستم عامل Dos می‌باشد و موجب پاک شدن محتویات صفحه نمایش می‌گردد را به خط فرمان داس فرستاده‌ایم تا صفحه نمایش را برای ما پاک کند. نکتهٔ بسیار مهم آنکه دستور موردنظر را باید به‌صورت یک رشته (در داخل گیومه) بنویسید. به‌عنوان مثالی دیگر اگر بخواهیم رنگ پس زمینه را به رنگ آبی و رنگ متن نوشتاری را به رنگ سبز در بیاوریم کافی است در ابتدای تابع main دستور زیر را تایپ کنید:

```
system("color 12");
```

برای یافتن اطلاعات بیشتر در خصوص فرامین Dos باید به قسمت help سیستم عامل Dos مراجعه کنید. برای این منظور ابتدا در گزینهٔ Run از منوی Start ویندوز خود عبارت cmd را تایپ کرده و کلید enter را بزنید تا پنجرهٔ command prompt باز شود. سپس در جلوی خط فرمان عبارت help را تایپ کرده و enter بزنید تا مستندات سیستم‌عامل Dos نمایان گردد. در این قسمت فرامین بیشتری را می‌توانید بیابید.

انتقال مکان‌نما در صفحه نمایش

اصولاً برای ایجاد یک خروجی مناسب و شکیل در صفحه نمایش نیازمند آن هستیم که به راحتی مکان نما را در صفحه حرکت داده و در هر نقطه که لازم باشد خروجی چاپ شود. این امکان در کامپایلرهای شرکت بورلند به‌واسطهٔ تابع **gotoxy** فراهم شده است. برای استفاده از این تابع باید هدر فایل **conio.h** را به برنامه اضافه کرد. این تابع دارای شکل کلی زیر است:

```
gotoxy(int X,int Y);
```

در حالت کنسول صفحه نمایش به‌صورت ۸۰ ستون و ۲۵ ردیف فرض می‌شود. که گوشهٔ بالا سمت چپ، معادل نقطهٔ **(0, 0)** و گوشهٔ پایین سمت راست، معادل نقطهٔ **(80, 25)** در نظر گرفته می‌شود. نکتهٔ قابل توجه آن که در دستور **gotoxy** ابتدا شمارهٔ ستون و سپس شمارهٔ ردیف ذکر می‌گردد. به‌عنوان مثال دستور **gotoxy(5, 10) ;** مکان‌نما را به ستون پنجم و ردیف دهم منتقل می‌کند.

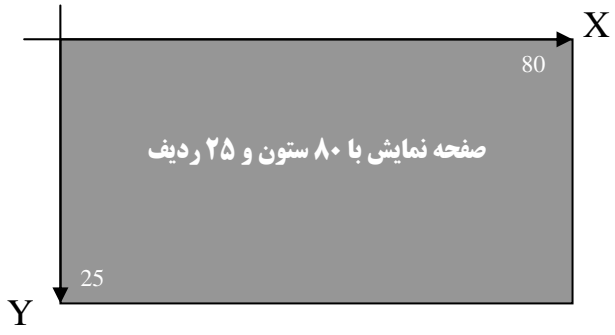
اما برای آنکه بتوانید در **VC6** نیز از این تابع استفاده کنید باید مراحل زیر را انجام دهید:

۱. قبل از تابع **main** و پس از دستور **using** خطوط زیر را تایپ کنید:

```
#include <windows.h>
void gotoxy(int,int);
```

۲. پس از آکولاد بستهٔ تابع **main** نیز دستورات زیر را تایپ کنید:

```
//function definition -- requires windows.h
void gotoxy(int x, int y)
{
  HANDLE hConsoleOutput;
  COORD dwCursorPosition;
  cout.flush();
  dwCursorPosition.X = x;
  dwCursorPosition.Y = y;
  hConsoleOutput = GetStdHandle(STD_OUTPUT_HANDLE);
  SetConsoleCursorPosition(hConsoleOutput,dwCursorPosition);
}
```



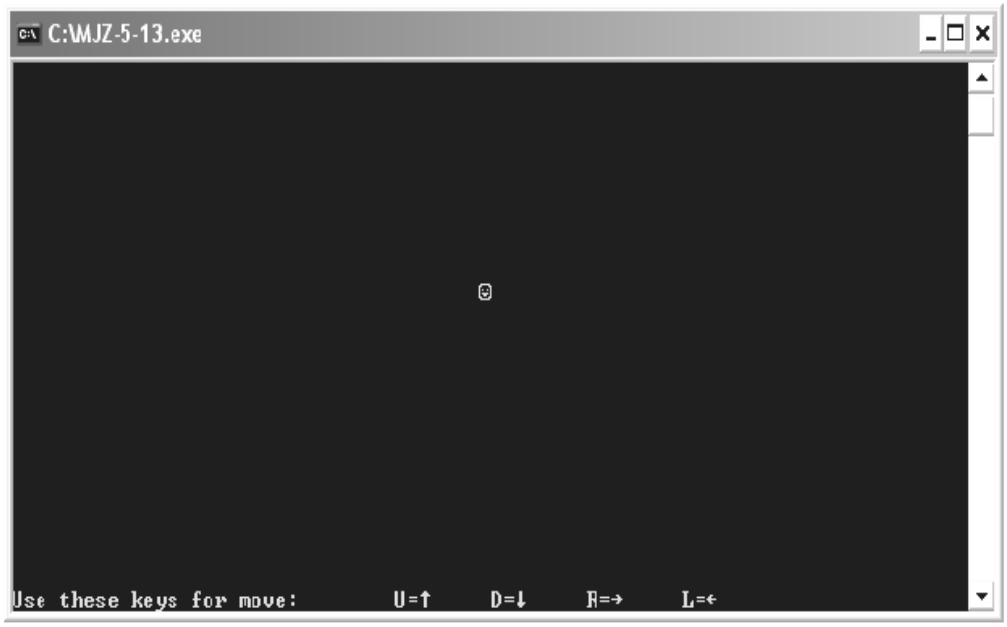
شکل ۵-۲: نمای صفحهٔ کنسول

مثال ۵-۱۴ : برنامه‌ای که با فشردن دکمه‌های U و D و L و R یک علامت ☺ را در صفحه نمایش به ترتیب به سمت بالا، پایین، چپ و راست می‌برد. هدف از این برنامه نشان دادن نحوهٔ به‌کارگیری ساختار **else...if** و توابع **clrscr** و **gotoxy** است.

```
1.  //This program demonstrates ELSE…IF statement
2.  #include <iostream>
3.  #include <conio.h>
4.  using namespace std;
5.  #include <windows.h>
6.  void gotoxy(int, int); //prototype
7.  void clrscr(); //prototype
8.  int main()
9.  {
10.  //move ☺ in to four corners of the screen.
11.   clrscr(); //function call
12.    gotoxy(0,25);//move to begin of the last line
13.     char U=24,D=25,R=26,L=27;//ASCII code of 4 vectors
14.      cout<<"Use these keys for move:\t"
15.          <<"U="<<U<<"\tD="<<D<<"\tR="<<R<<"\tL="<<L;
16.   char sg=1,ch=0;//sg has the ASCII code of ☺
17.    int x=39,y=11;//x and y store position of cursor.
18.     gotoxy(x,y);//move to center of screen.
19.      cout.put(sg);
20.       gotoxy(x,y);//return to previous position.
21.       while((ch=getch())!=27)
22.           {
23.              //start getting command of user.
24.              if((ch=='U'||ch=='u') && y>1)
25.                {
26.                   cout.put(' ');
27.                    gotoxy(x,--y);
28.                   cout.put(sg);
29.                    gotoxy(x,y);
30.                }
31.              else if((ch=='D'||ch=='d') && y<24)
32.                {
33.                   cout.put(' ');
34.                    gotoxy(x,++y);
35.                   cout.put(sg);
36.                    gotoxy(x,y);
37.                }
38.              else if((ch=='R' || ch=='r') && x<79)
39.                {
40.                    cout.put(' ');
41.                     gotoxy(++x,y);
42.                    cout.put(sg);
43.                     gotoxy(x,y);
44.                }
```

```
45.                 else if((ch=='L'||ch=='l') && x>0)
46.                   {
47.                       cout.put(' ');
48.                        gotoxy(--x,y);
49.                       cout.put(sg);
50.                        gotoxy(x,y);
51.                   }
52.              }//end of while
53.          clrscr();
54.     return 0;
55. }
56. // function definition -- requires windows.h
57. void gotoxy(int x, int y)
58. {
59.   HANDLE hConsoleOutput;
60.    COORD dwCursorPosition;
61.     cout.flush();
62.   dwCursorPosition.X = x;
63.    dwCursorPosition.Y = y;
64.     hConsoleOutput = GetStdHandle(STD_OUTPUT_HANDLE);
65.      SetConsoleCursorPosition(hConsoleOutput,dwCursorPosition);
66. }
67. //function definition.In some compilers require process.h
68. void clrscr()
69. {
70.   system("cls");
71. }
```

توضیح مثال : در این قسمت از خط ۹ تا ۵۳ که مربوط به تابع اصلی است را توضیح می‌دهیم. از خط ۱۱ تا ۱۴ ابتدا مکان‌نما را به ابتدای آخرین خط برده و توضیحاتی را برای کاربر چاپ می‌کنیم، که از چه کلیدهایی قادر است استفاده کند. متغیر **sg** حاوی کد اسکی علامت ☺ است و برای چاپ این علامت از **sg** استفاده می‌کنیم. متغیرهای **x** و **y** حاوی آخرین موقعیت مکان‌نما هستند. در خطوط ۱۶ تا ۱۸ مکان‌نما را به وسط صفحه برده و علامت ☺ را چاپ می‌کنیم. در خط ۱۹ مکان‌نما را دوباره به محل چاپ علامت دایره‌ای هدایت می‌کنیم تا در دفعهٔ بعد که می‌خواهیم این علامت را حرکت دهیم ابتدا آن را پاک کنیم و سپس علامت را در مکان جدیدی چاپ کنیم. در حلقهٔ **while** تا زمانی که کاربر کلید **Esc** را فشار ندهد فرامین کاربر را دریافت می‌کنیم. کد اسکی کلید **Esc** برابر ۲۷ می‌باشد. از بین بلوک‌های **else...if** به دلیل تشابه کد تنها خطوط ۲۲ تا ۲۸ را توضیح می‌دهیم. در خط ۲۲ ابتدا چک می‌کنیم که اگر کاربر کلید **U** را فشار داده است و مکان‌نما به بالای صفحه نرسیده است، وارد بلوک نخست می‌شویم. در خط ۲۴ این بلوک ابتدا علامت را از مکان قبلی آن پاک می‌کنیم و سپس با کاهش مقدار **y** و هدایت مکان‌نما به یک خانه بالاتر علامت ☺ را در محل جدید چاپ می‌کنیم. و در خط ۲۷ دوباره اشاره‌گر را به زیر علامت دایره‌ای هدایت می‌کنیم.



شکل ۵-۳: نمونهٔ خروجی مثال ۵-۱۴

تذکر: آخرین نکته‌ای که در خصوص دستورات **if** بیان خواهیم کرد این است که، همیشه یک دستور **else** با آخرین دستور **if** که دارای هیچ **else** ای نیست متناظر می‌شود. به‌عنوان مثال شما فکر می‌کنید با اجرا شدن قطعه کد زیر کدام یک از خطوط ۴ یا ۶ اجرا می‌گردد؟

```
1.  int a=5, b=7, c=7;
2.   if(a==b)
3.     if(b==c)
4.        cout<<"a and b and c are the same.\n";
5.     else
6.        cout <<"a and b are different.\n";
```

احتمالاً به این نتیجه رسیده‌اید که خط ۶ اجرا می‌شود. اما در حقیقت هیچ یک از دو پیغام فوق چاپ نمی‌گردد. چرا که **else** موجود در خط ۵ با **if** موجود در خط ۳ متناظر می‌شود، نه **if** موجود در خط ۲. برای آنکه **else** موجود در خط ۵ با **if** موجود در خط ۲ متناظر گردد باید از بلوک‌بندی استفاده کنیم. در این صورت باید بنویسیم:

```
1.  int a=5, b=7, c=7;
2.   if(a==b)
3.     {
4.        if(b==c)
5.          cout<<"a and b and c are the same.\n";
6.     }
7.   else
8.     cout<<"a and b are different.\n";
```

تولید اعداد تصادفی

در C++ تابع کتابخانه‌ای به نام **rand** وجود دارد که با فراخوانی آن یک عدد تصادفی از نوع صحیح تولید کرده و عدد مذکور را برمی‌گرداند. الگوی این تابع در فایل **stdlib.h** قرار دارد که باید به برنامه اضافه گردد. شکل کلی به‌کارگیری این تابع به‌صورت زیر است.

```
int magic = rand();
```

اما چنانکه پیشتر گفته شد در بورلند C++ دو تابع **random** و **randomize** وجود دارند که در **VC6** قابل استفاده نیستند. تابع **random** موجب می‌شود که عدد تصادفی تولید شده در یک بازهٔ خاص قرار داشته باشد. شکل کلی به‌کارگیری این تابع به‌صورت زیر است:

```
int magic = random(num);
```

عدد تصادفی تولید شده توسط این تابع در بازهٔ صفر تا **num** قرار دارد. **num** باید یک عدد غیر اعشاری باشد. همچنین الگوریتم تولید اعداد تصادفی نیاز به یک عدد مبنا دارد. اگر تابع **random** را بدون تعیین مبنای تولید اعداد تصادفی بکار ببرید هر بار که از این تابع استفاده می‌کنید نتایج مشابهی را مشاهده خواهید کرد. زیرا عدد مبنا به‌طور خودکار برابر صفر فرض می‌شود. لذا تابع اعداد تصادفی همواره نتایج مشابهی را به‌دست خواهد داد. اما برای تعیین یک مبنا برای تولید اعداد واقعاً تصادفی از تابع **randomize** استفاده می‌شود. اصولاً قبل از تولید اعداد تصادفی با اجرای تابع **randomize** یک عدد تصادفی برحسب زمان سیستم تولید می‌شود و به‌عنوان مبنای تولید اعداد تصادفی بعدی قرار می‌گیرد. این تابع نیز به‌صورت زیر به‌کار می‌رود:

```
randomize();
```

اما برای شبیه‌سازی این توابع در محیط **VC6** باید خطوط زیر را در ابتدای برنامه اضافه کنید:

```
#include <cstdlib>
#include <ctime>
#define randomize() (srand(time(0)))
#define random(x) (rand()%x)
```

مثال ۵-۱۵ : برنامه‌ای که نحوهٔ استفاده از این توابع را نشان می‌دهد. این برنامه را چندین بار اجرا کنید و نتایج آن را با هم مقایسه نمایید. خروجی خط ۹ همواره ثابت اما خروجی خط ۱۲ متغیر است.

```
1.  //This program generates random number.
2.  #include <cstdlib>
3.  #include <ctime>
4.  #define randomize() (srand(time(0)))
5.  #define random(x) (rand()%x)
6.  #include <iostream.h>
7.  int main ( )
8.  {
9.      for(int i=0;i<10;cout<<random(25)<<endl ,i++);
10.      randomize();
11.       cout<<"After call randomize function.\n";
12.     for(i=0;i<10;cout<<random(25)<<endl ,i++);
13.   return 0;
14. }
```

ساختار switch

یکی از ساختارهایی که می‌تواند معادل جعبه‌های تصمیم چندگانه یا درخت‌های انتخاب به‌کار رود ساختار **switch** است. با این تفاوت که در ساختارهای چند انتخابی قبلی مانند **else if** قادر بودیم هرگونه شرطی را مورد ارزیابی قرار دهیم ولی ساختار **switch** تنها برای ارزیابی *"مساوی بودن"* یک عبارت با مقادیر گوناگون، به‌کار می‌رود. شکل کلی این ساختار به‌صورت زیر است:

```
switch  ( عبارت )
{
        case   مقدار ۱ :
                    مجموعه دستورات ۱
                    break;
        case   مقدار ۲ :
                    مجموعه دستورات ۲
                    break;
                .
                .
                .
        case مقدار آخر :
                    مجموعه دستورات آخر
                    break;
        default:
                    مجموعه دستورات حالت استثناء
}
```

در ساختار فوق در جلوی دستور **switch** یک عبارت محاسباتی یا نام یک متغیر قرار می‌گیرد. نتیجهٔ حاصل از ارزیابی عبارت محاسباتی یا متغیری که در بین دو پرانتز قرار می‌گیرد باید از نوع عدد صحیح باشد، مثلاً می‌تواند **int** یا **char** باشد ولی از انواع ممیز شناور مثل **float** نمی‌تواند باشد. در جلوی هر یک از کلمات **case** باید یک عدد صحیح یا یک کد اسکی قرار گیرد. هنگامی که کنترل اجرای برنامه به دستور **switch** می‌رسد مقدار داخل پرانتز ارزیابی می شود، سپس این مقدار با مقدار اولین **case** مقایسه شده در صورت برابر بودن مجموعه دستورات ۱ اجرا شده و با رسیدن به دستور **break** از ساختار **switch** خارج می‌شویم. درصورتی که مقدار داخل پرانتز با مقدار جلوی اولین **case** برابر نباشد، کنترل اجرای برنامه به سراغ **case** بعدی می‌رود و عمل مقایسهٔ عبارت داخل پرانتز را با مقدار جلوی این **case** انجام می‌دهد. این روند تا آخرین **case** ادامه پیدا می‌کند تا آنکه بالاخره یک **case** مشابه مقدار داخل پرانتز پیدا شود. اگر مقدار هیچ یک از **case**ها با عبارت داخل پرانتز برابر نباشد و ساختار **switch** دارای قسمت **default** باشد دستورات داخل این قسمت اجرا شده و در نهایت از ساختار **switch** خارج می‌شویم. اگر تمایل دارید بلوک واحدی از دستورات برای **case**های متفاوتی اجرا شود کافی است از قرار دادن **break** خودداری کنید تا مقادیر **case**ها با هم **or** بشوند.

مثال ۵-۱۶: برنامه‌ای که با تولید اعـداد تصادفی چگونگی ریخته شدن تاس را شبیه‌سازی می‌کند. در این برنامه تا زمانی که کاربر کلید **enter** را بزند، تاس ریخته شده و عدد مربوط به آن نمایش داده می‌شود. با زدن کلید **Esc** از برنامه خارج می‌شویم.

```
1.  //This program can imagery dice rolling.
2.  #include <cstdlib>
3.  #include <ctime>
4.  #define  randomize()(srand(time(0)))
5.  #define  random(x)(rand()%x)
6.  #include <conio.h>
7.  #include <iostream>
8.  using namespace std;
9.  void main()
10. {
11.  cout<<"Press Enter for roll the dice and press Esc for exit.";
12.    int n=1;
13.      do{
14.          char ch=getch();
15.           if(ch==13)
16.             {
17.                system("cls");
18.                randomize();
19.                 switch(random(6)+1)
20.                {
21.                    case 1:
22.                           cout<<n++<<"- The dice is 1.";
23.                           break;
24.                    case 2:
25.                           cout<<n++<<"- The dice is 2.";
26.                           break;
27.                    case 3:
28.                           cout<<n++<<"- The dice is 3.";
29.                           break;
30.                    case 4:
31.                           cout<<n++<<"- The dice is 4.";
32.                           break;
33.                    case 5:
34.                           cout<<n++<<"- The dice is 5.";
35.                           break;
36.                    default :
37.                           cout<<n++<<"- The dice is 6.";
38.                  }//end of switch
39.              }//end of if
40.           else if(ch==27)
41.             break;//break the loop if user press the Esc.
42.        }while(1);//end of DO…WHILE loop.
43. }
```

توضیح مثال : در این مثال در یک حلقهٔ تکرار بی‌نهایت در خط ۱۳ شروع به دریافت فرامین کاربر کرده‌ایم. اگر کاربر کلید **enter** را بفشارد در خطوط ۱۸ و ۱۹ یک عدد تصادفی بین ۱ تا ۶ تولید می‌شود. سپس در **case**های مختلف بر حسب اینکه عدد تصادفی تولید شده کدام یک از اعداد ۱ تا ۶ باشد پیغام مناسب، چاپ می‌شود. هرگاه کاربر کلید **Esc** را بفشارد دستور **break** موجود در خط ۴۱ موجب شکستن حلقهٔ تکرار شده و اجرای برنامه خاتمه می‌یابد.

مثال ۵-۱۷ : برنامه‌ای که طریقهٔ **or** کردن **case**های مختلف دستور **switch** (ایجاد **case**های چندگانه) را نشان می‌دهد. در این برنامه نیز یک عدد تصادفی بین ۱ تا ۶ به‌عنوان عدد تاس ریخته شده تولید می‌شود و برنامه تنها زوج یا فرد بودن عدد را اعلام می‌کند.

```
1.  //This program shows what happen when you delete BREAK.
2.  #include <cstdlib>
3.  #include <ctime>
4.  #define  randomize()(srand(time(0)))
5.  #include <iostream>
6.  using namespace std;
7.  int main()
8.  {
9.       randomize();
10.      switch (rand()%6+1)
11.      {
12.         case 1:
13.         case 3:
14.         case 5:
15.                cout<<"The dice is odd.\n";
16.                break;
17.         case 2:
18.         case 4:
19.         case 6:
20.                cout<<"The dice is even.\n";
21.                break;
22.      }
23.   return 0;
24. }
```

عملگر شرطی (عملگر سه‌تایی Ternary Operator)

در C++ می‌توان به واسطهٔ عملگر ? یک ساختار تصمیم‌گیری ساده ولی پرکاربرد را پیاده‌سازی کرد که به این واسطه در کدنویسی صرفه‌جویی می‌شود. شکل کلی به‌کارگیری این عملگر به‌صورت زیر است:

متغیر = عبارت شرطی ? عبارت محاسباتی ۱ : عبارت محاسباتی ۲ ;

عملکرد این عملگر به این صورت است که اگر عبارت شرطی دارای ارزش درستی بود آن‌گاه مقدار عبارت محاسباتی ۱ ارزیابی شده و در متغیر سمت چپ دستور انتساب قرار می‌گیرد، و الا اگر عبارت شرطی دارای ارزش نادرستی باشد مقدار عبارت محاسباتی ۲ ارزیابی شده و در متغیر سمت چپ دستور انتساب قرار می‌گیرد.

به عبارت دیگر عملگر سه‌تایی معادل کد زیر عمل می‌کند:

```
if ( عبارت شرطی )
    متغیر  =  عبارت محاسباتی ۱  ;
else
    متغیر  =  عبارت محاسباتی ۲  ;
```

مثال ۵-۱۸ : برنامه‌ای که نحوهٔ عملکرد عملگر سه‌تایی را نشان می‌دهد. در این برنامه می‌توانید ستون‌بندی پرشی (Tab Stop) را در صفحه نمایش ملاحظه کنید.

```
1.  //This program prints 'X' every 8 columns
2.  //by means of conditional operator.
3.  #include <iostream>
4.  using namespace std;
5.  int main()
6.  {
7.       for(int j=0;j<80;j++)
8.         {
9.            char ch=(j%8) ? ' ' : 'X';
10.            cout<<ch;
11.         }
12.     return 0;
13. }
```

توضیح مثال : در این برنامه هرگاه باقی‌ماندهٔ **j** بر ۸ برابر صفر گردد کاراکتر **X** و در غیر این صورت فضای‌خالی در صفحه نمایش چاپ می‌شود.

کار در کلاس ۵-۴ :

مثال ۵-۱۴ را به وسیلهٔ دستور **switch** *بازنویسی کنید. توجه داشته باشید که در این مورد مقادیر* **case** *های مختلف کد اسکی هستند.*

مثال ۵-۱۹ : برنامه‌ای که یک عملگر و دو عملوند صحیح را از ورودی گرفته و عملگر را برروی عملوندهای خود اجرا می‌کند.

```
1.  //This program uses SWITCH with char variable.
2.  #include <iostream>
3.  #include <conio.h>
4.  using namespace std;
5.  int main()
6.  {
7.   int num1, num2;
8.   bool  control=true;
9.   char opt;
10.  while(control)
11.   {
12.     system("cls");
13.      cout<<"Enter num1, operator , num2:";
14.       cin>>num1>>opt>>num2;
15.     switch(opt)
16.    {
17.       case '+' :
18.             cout<<"\n sum = "<<num1+num2;
19.             break;
20.       case '-' :
21.             cout<<"\n minus = "<<num1-num2;
22.             break;
23.       case '/' :
24.       case '\\':
25.             cout<<"\n division = "<<(float)num1/num2;
26.             break;
27.       case '*' :
28.             cout<<"\n multiply = "<<num1*num2;
29.             break;
30.       default:
31.     cout<<"\nOperator is illegal.Press a key to end.";
32.       control=0;//It's similar to write control=false;
33.    }//end of switch
34.    getch();
35.   }//end of while
36.  return 0;
37. }
```

توضیح مثال : اولین نکتهٔ جدید این مثال استفاده از یک متغیر از نوع **bool** به‌عنوان نگهبان حلقهٔ تکرار است. زمانی که کاربر کاراکتر اشتباهی را به‌عنوان عملگر وارد کند، متغیر کنترلی مقدار **false** به خود گرفته و موجب اتمام کار حلقه می شود. همچنین در خطوط ۲۳ و ۲۴ برنامه هر دو کاراکتر / و \ به‌عنوان عملگر تقسیم با **or** کردن دو **case** متوالی پذیرفته شده است. همچنین به به‌کارگیریِ تبدیل نوع استاتیک در خط ۲۵ توجه کنید.

۵-۴ : تمرین

۱. خطای قطعه کدهای زیر را بیان کنید.

I.

```
1.   if(x=y)
2.      cout<< ++(x+y);
3.   else;
4.       x--;
```

II.

```
1.   int z = 2, sum;
2.    while(z >= 0)
3.         sum += z;
```

III.

```
1.   switch(value%2)
2.   {
3.         case 0:
4.              cout<<"Even integer."<<endl;
5.         case 1:
6.              cout<<"Odd integer."<<endl;
7.   }
```

۲. کدام یک از عبارات زیر درست و کدام یک نادرست است.

الف- وجود حالت **default** در دستور **switch** الزامی است.

ب- وجود دستور **break** در دستور **default** الزامی است.

ج- دستور **default** حتماً باید به‌عنوان آخرین قسمت دستور **switch** قرارگیرد.

د- اگر متغیرهایی را داخل بلاک تعریف کنیم در آن صورت خارج از بلاک قابل دسترسی نیستند.

۳. برنامه‌ای به زبان **C++** بنویسید که یک مربع و یک لوزی توخالی را به‌واسطهٔ دستورات **gotoxy** و **for** در صفحه رسم کند. سپس برنامه را به‌گونه‌ای گسترش دهید که طول ضلع شکل را هم دریافت کند.

۴. برنامه‌ای بنویسید که دو عدد تصادفی تولید کند و سپس براساس آن دو عدد یک علامت مثل ☻ را در صفحه جابه‌جا کند و به محلی که آن دو عدد تصادفی اعلام می‌کند ببرد. دو عدد تصادفی باید در محدودهٔ مجاز صفحه باشند.

۵. برنامه‌ای بنویسید که یک عدد مبنای ۱۰ را دریافت کند. سپس عدد دریافتی را به هر مبنای مورد نظری ببرد.

۶. برنامه‌ای بنویسید که با دریافت یک عدد در مبنایی غیر از ۱۰، آن را به مبنای ۱۰ ببرد و نتیجه را چاپ کند.

۷. برنامه‌ای به زبان C++ بنویسید که مقدار عدد π را از رابطهٔ زیر محاسبه و چاپ کند.

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \ldots$$

۸. خروجی برنامهٔ زیر را مشخص کنید.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.      int count =1;
6.        while(count <= 10)
7.              {
8.                cout<<(count % 2 ? "****" : "++++++++")
9.                    <<endl;
10.               count++;
11.             }
12.    return 0;
13. }
```

۹. برنامه‌ای به زبان C++ بنویسید که مقدار e^x را با استفاده از فرمول‌های زیر تا ۹ رقم اعشار حساب کند.

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

۱۰. با استفاده از قوانین دمورگان معادل هریک از عبارات منطقی زیر را بیان کنید.

الف- `!(x<5)&&(y==32)`

ب- `!(a==b)||!(h!=8)`

ج- `!((v<=43)&& f>3)`

د- `!(i>3 || a<=3)`

۱۱. یکی از اشکالات دستور **break** و دستور **continue** آن است که این دستورها، ساخت یافته نیستند. در عمل، دستورهای **break** و **continue** را می توان به‌صورت دستورهای ساخت یافته شبیه‌سازی کرد که می تواند با اشکالاتی نیز همراه باشد. در حالت کلی توضیح دهید چگونه می‌توانید هر دستور **break** یا **continue** در یک برنامه را از حلقهٔ تکرار حذف کنید در عین حالی که روند اجرائی حلقه با مشکلی مواجه نشود.

۱۲. شرکتی می‌خواهد انتقال داده‌ها را از طریق خطوط تلفن انجام دهد اما این کار تنها می تواند از طریق داده‌های چهار رقمی انجام شود. برنامه‌ای بنویسید که داده‌های شرکت را به رمز در آورد و آنها را به شکل مطمئن‌تری انتقال دهد. برنامه باید یک عدد صحیح چهار رقمی را بخواند و آن را به این صورت رمز کند که به جای هر رقم، باقیمانده تقسیم بر ۱۰ آن عدد را قرار دهد، سپس جای رقم اول را با رقم سوم و رقم دوم را با رقم چهارم جابه‌جا

کند. و نتیجه را در خروجی چاپ کند. همچنین برنامه را به اینگونه گسترش دهید که قادر باشد یک عدد رمز شده را رمزگشایی کند.

۱۳. برنامه‌ای به زبان C++ بنویسید که ب.م.م و ک.م.م دو عدد دریافتی را حساب کند.

۱۴. برنامه‌ای بنویسید که حرف B را به‌صورت زیر در خروجی بنویسید.

```
* * * * *
*       *
*       *
* * * * *
*       *
*       *
* * * * *
```

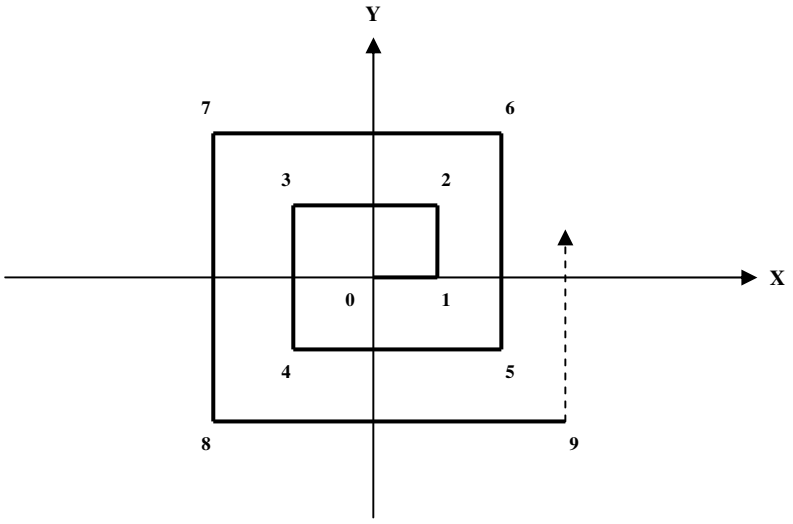
۱۵. برنامه‌ای به زبان C++ و با استفاده از دستور sizeof بنویسید که مشخص کند یک سیستم کامپیوتری برای هر یک از ۱۲ نوع پایه‌ای موجود در C++ چقدر فضا برحسب بیت استفاده می‌کند.

۱۶. برنامه‌ای بنویسید که با دریافت عدد صحیح مثبت n مثلث متساوی الساقینی با قاعدهٔ 2n–1 ستاره و ساق‌هایی در n خط در خروجی چاپ کند. بهتر است از عملگر سه‌تایی در حل این مسئله استفاده‌کنید.

۱۷. برنامه‌ای بنویسید که شمارهٔ دانشجویی و معدل تعداد n دانشجو را از ورودی بخواند، و دانشجویی را که دومین معدل را از نظر بزرگی دارد، پیدا کند و به خروجی ببرد. (برای حل این مسئله نباید از آرایه‌ها استفاده کنید.)

۱۸. برنامه‌ای بنویسید که اعداد چهار رقمی را بیابد که مضربی از مقلوب خود باشند.

۱۹. برنامه‌ای بنویسید که عدد n را بگیرد و با توجه به شکل زیر موقعیت نقطهٔ n–اُم را به‌صورت یک دوتایی مرتب شامل x و y چاپ نماید.



۲۰. برنامه‌ای بنویسید که عدد را بخواند و بگوید آن عدد کامل هست یا نه. عددی کامل است که مجموع مقسوم‌علیه‌های آن به جزء خودش برابر با آن عدد باشد.

۲۱. برنامه‌ای بنویسید که رنگ پس زمینه و رنگ متن را از کاربر بپرسد و آن‌گاه به‌واسطهٔ دستور **system** و با استفاده از دستورات **DOS** رنگ پس‌زمینه و متن صفحهٔ کنسول را تغییر دهد.

۲۲. برنامه‌ای بنویسید که یک عدد اعشاری را خوانده و وارون (مقلوب) آن را به خروجی ببرد.

۲۳. برنامه‌ای بنویسید که یک عدد اعشاری را خوانده و قسمت‌های صحیح و اعشاری آن را به‌صورت دو عدد صحیح مجزا به خروجی ببرد.

۲۴. برنامه‌ای بنویسید که **n** عدد اعشاری با ۴ رقم اعشار به‌صورت تصادفی تولید کند، سپس هر عدد را با توجه به آنچه در خصوص گرد کردن اعداد در قسمت ۲-۱۰ فراگرفتید تا سه رقم اعشار گرد کند.

۲۵. برنامه‌ای بنویسید که یک سکهٔ ۱۰۰ ریالی را با سکه‌های ۲، ۵، ۱۰، ۲۰ و ۵۰ ریالی خرد کند. برنامه باید تمامی حالت‌های ممکن را چاپ کند.

۲۶. برنامه‌ای بنویسید که توزیع اعداد اول را به‌صورت یک نمودار گرافیکی نشان دهد. در این نمودار هر مکان در صفحه نمایش ۸۰ ستون و ۲۵ سطری از ۰ تا ۹۹۹ به‌طور فرضی شماره‌گذاری شده است. اگر شمارهٔ یک مکان خاص از صفحه نمایش، اول باشد آن مکان با رنگ سفید، باید رنگ آمیزی شود، و در صورت اول نبودن باید به رنگ خاکستری در آید. جهت رنگ‌آمیزی به رنگ سفید از کد اسکی ۲۱۹ و جهت رنگ‌آمیزی به رنگ خاکستری از کد اسکی ۱۷۶ استفاده کنید.

۲۷. یکی از جالب‌ترین خصوصیات دنبالهٔ فیبوناچی رابطهٔ آنها با عدد طلایی است، به‌طوری که هرچه اعداد فیبوناچی را تولید کنیم و نسبت دو جملهٔ آخر دنباله را به‌دست آوریم (نسبت عدد کوچکتر به عدد بزرگتر) تقریب بهتری از نسبت طلایی به‌دست خواهیم آورد. برنامه‌ای بنویسید که عدد طلایی را با بهترین تقریب ممکن به‌دست آورد.

۲۸. در برنامهٔ مثال ۵- ۸ ممکن است یک اشکال منطقی در حین اجرای برنامه پیش آید. ضمن کشف این اشکال، برنامه را دوباره به‌صورت صحیح بازنویسی کنید.

۲۹. برنامه‌ای بنویسید که عملیات چهارگانهٔ ریاضی را بر روی دو کسر پیاده‌سازی کند.

۳۰. مردی اصفهانی سرمایه‌ای معادل ۱۰ میلیون تومان دارد. او سرمایهٔ خود را در یک بانک خصوصی با بهرهٔ مرکب ۱۵٪ در یک حساب پس‌انداز بلند مدت قرار داده. او می‌خواهد مقدار سرمایهٔ خود را به‌صورت سالانه و تا ۱۰ سال محاسبه کند. فرض کنید وی هیچگونه سودی تا پایان ده سال از بانک دریافت نکند و سود پول او توسط بانک در یک حساب کوتاه مدت دیگر ریخته شود، برای حساب کوتاه مدت سود سپرده یک سوم سود حساب پس انداز بلند مدت محاسبه می‌گردد. سرمایهٔ وی در پایان هر سال برابر مجموع پول‌های موجود در این دو حساب است. برنامه‌ای بنویسید که سود هر سال را در یک خط خروجی برای ۱۰ سال چاپ کند.

۵-۵ : موارد مطالعاتی

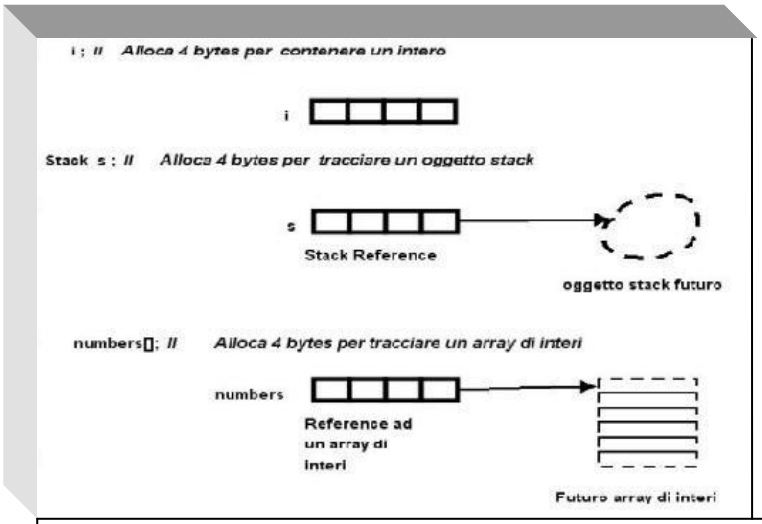
۱. (پروژهٔ برنامه‌نویسی): با توجه به آنچه در مسئلهٔ ۲۶ بخش ۴-۴ دیدید بازی چرخ گردان را با تولید اعداد تصادفی شبیه‌سازی کنید.

۲. (پروژهٔ برنامه‌نویسی): برنامه‌ای بنویسید که بازی دوز را اجرا کند. بازی دوز به این صورت است که از یک جدول ۳×۳ تشکیل می‌شود. و دو بازی‌کن دارد. به قید قرعه یک بازی‌کن شروع‌کننده می‌شود. بازی به این‌صورت است که هر بازی‌کن ۳ مهرهٔ جداگانه دارد و به ترتیب یکی پس ازدیگری مهره‌های خود را وارد جدول می‌کنند. پس از آنکه هر دو بازی‌کن تمامی مهره‌های خود را در خانه‌های جدول قرار دادند و تنها سه خانه آزاد ماند از آن پس تنها مجاز به جابه‌جا کردن مهره‌ها در بین خانه‌های آزاد جدول هستند. هر کس که زودتر مهره‌های خود را در یک خط راست(چه به‌صورت قطری و چه به‌صورت ردیفی یا ستونی) قرار داد، برنده محسوب می‌شود و بازی خاتمه می‌یابد. برنامهٔ شما باید با ریختن دو تاس آغاز شود و هر بازی‌کن که تاس بالاتری بیاورد آغازکننده خواهد بود. برنامه در هر بار باید از بازی‌کنی که مجاز به انجام بازی است بپرسد که کدام مهره‌اش را می‌خواهد جابه‌جا کند. همچنین باید محل جدید مهره را از بین محل‌های ممکن بپرسد. در صورت درست بودن باید حرکت را انجام دهد. هر موقع که یک بازی‌کن بازی را برد باید پیام مناسب را چاپ کند و بازی را خاتمه دهد. برنامه باید موقعیت مهره‌ها را در جدولی ۳×۳ در مانیتور نمایش دهد. بهتر است هر بار که نوبت بازی بین بازی‌کن‌ها عوض می‌شود رنگ پس زمینه نیز تغییر یابد. مثلاً رنگ سبز برای بازی‌کن شماره یک و رنگ آبی برای بازی‌کن شماره دو. همچنین هنگام ورود داده، نباید جزء ارقام مجاز چیز دیگری در صفحه چاپ گردد.

۳. (پروژهٔ برنامه‌نویسی): سعی کنید پروژهٔ قبلی را به‌گونه‌ای گسترش دهید که یکی از بازی‌کن‌ها کامپیوتر باشد.

۴. دقت اعداد اعشاری **float** تا ۷ رقم اعشار است. بنابراین کسر عدد ۹/۹۰ از ۱۰/۰۰ می‌تواند نتیجهٔ متفاوتی با ۰/۱۰ داشته باشد. برای مثال ممکن است عدد ۰/۰۹۹۹۹۹۹ در خروجی چاپ شود. لذا در محاسبات پولی و مالی که دقت اعداد در آنها اهمیت دارد بهتر است از نوع **float** استفاده نکنید. با مراجعه به شبکهٔ جهانی اینترنت یا مستندات **MSDN** تحقیق کنید که در کتابخانهٔ **C++** چه امکاناتی برای محاسبات پولی و مالی فراهم شده است. مثلاً به دنبال نوع **decimal** بگردید.

۵. از مدرس خود بخواهید منابعی را در خصوص نحوهٔ تولید اعداد تصادفی در کامپیوتر به شما معرفی کند. با مراجعه به آن منابع مقاله‌ای در خصوص این موضوع تهیه و به کلاس ارائه دهید. سعی کنید شما نیز روش جدیدی را برای تولید اعداد تصادفی ابداع کنید.



فصل ششم

نقش آرایه‌ها و متغیرهای لیستی در الگوریتم

اهداف فصل و چکیده مطالب :

۱. آرایه چیست و چه کاربردی دارد؟
۲. متغیر لیستی(زیرنویس‌دار)
۳. آشنایی با جستجوی ترتیبی و جستجوی دودویی
۴. آشنایی با مرتب‌سازی حبابی و مرتب‌سازی درجی
۵. عملیات اشتراک و اجتماع و ادغام بر روی مجموعه‌ها
۶. کاربرد متغیرهای وضعیت(سوئیچ)
۷. آرایه‌های دو بعدی و ماتریس‌ها
۸. کاربرد حلقه‌های تودرتو در پردازش ماتریس‌ها
۹. جدول خوانی با روش‌های تطابق و نصف کردن

۶-۱ : آرایه و کاربرد آن

مقدمه

فرض کنید برای هر یک از داوطلبان کنکور امسال پس از تصحیح پاسخ‌نامه‌ها و منظورکردن تمامی ضرایب، نمره‌ای بین ۱ تا ۱۰۰۰ به‌دست آمده است. نمرات ۱.۴۵۳.۲۸۶ نفر داوطلب شرکت‌کننده در آزمون امسال، بر روی یک نوار مغناطیسی به‌صورت یک سری عدد ثبت و ضبط شده است. حال سازمان سنجش کشور می‌خواهد با پردازش این تعداد رکورد، آماری از داوطلبان امسال تهیه کند و در جدولی ۱۰۰۰ ردیفی در پیک سنجش چاپ نماید، که شامل تعداد دانشجویانی باشد که هر یک از این نمرات را کسب کرده‌اند. به‌عنوان مثال در ردیف صدم تعداد افرادی که نمرهٔ ۱۰۰ گرفته‌اند مشخص باشد و به همین ترتیب ادامه یابد. شما چه روشی را برای حل این مسئله پیشنهاد می‌کنید؟

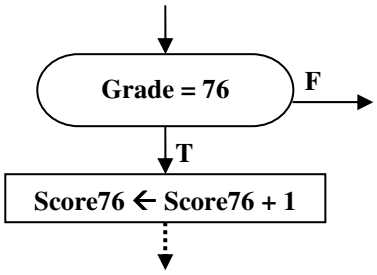
اگر به‌خاطر بیاورید پیشتر در بخش ۲-۶ کارنمایی را در شکل ۲-۲۲ برای شمارش نمرات موجود در چند محدودهٔ خاص پیشنهاد کردیم. شاید در نخستین تلاش‌ها اینگونه به نظر آید که، از مکانیسمی مشابه کارنمای شکل ۲-۲۲ بتوان برای حل این مسئله نیز استفاده کرد. اما آیا واقعاً چنین است؟ اگر بخواهیم از روش مذکور برای حل این مسئله استفاده کنیم با مهمترین مشکلی که برخورد می‌کنیم، چگونگی تعریف هزار متغیر برای ذخیرهٔ تعداد هر یک از این هزار نمره است!؟ مسلماً هزار حرف الفبا نداریم که چنین کاری را بتوانیم به‌راحتی انجام دهیم، پس باید به فکر ترکیب حروف و ارقام جهت ساخت هزار نام متفاوت باشیم. مثلاً یک نام را بگیریم **ljdas** و دیگری را بگیریم **hGas** و همین‌طور ادامه دهیم تا به هزار نام متفاوت برسیم. اما آیا واقعاً کار کردن با این اسامی جورواجور و متفاوت آسان است!؟ از کجا به خاطر داشته باشیم که کدام متغیر مربوط به کدام نمره است!؟ پس باید اصلاحاتی در شیوهٔ نامگذاری متغیرها ایجاد کنیم تا بر مشکلات مطرح شده فائق آییم. پیشنهادی که می‌توان به واسطهٔ آن، این مشکل را حل کرد، قرار دادن یک دنباله عددی به دنبال نام هر متغیر است تا ارتباط هر شمارنده را با نمرهٔ مربوطه‌اش نشان دهد. مثلاً اگر قرار است متغیر **hGas** مربوط به نمرهٔ ۴۶۲ باشد نام این متغیر را به‌صورت زیر تغییر دهیم و بنویسیم **hGas426** . اما اگر توجه کرده باشید، با تغییر کردن دنبالهٔ نام متغیرها، دیگر احتیاجی نیست به دنبال ترکیب حروف مختلف برای ساخت نام‌های متفاوت باشیم. چرا که تغییرات دنباله خود تضمین‌کنندهٔ متفاوت بودن نام متغیرها است. بنابراین می‌توانیم شروع نام تمامی متغیرها را با یک نام مشابه مثل **Score** در نظر بگیریم. با توجه به نکات مطرح شده می‌توان هزار نام متفاوت را برای شمارنده‌های موردنیاز این مسئله به‌صورت زیر در نظر بگیریم:

Score1 , Score2 , Score3 , … , Score999 , Score1000

شَکّی نیست که این هزار نام متفاوت مشکل نامگذاری را برای ما هموار می‌سازند اما آیا حالا دیگر می‌توانیم مسئله را به‌راحتی حل کنیم؟ فقط به نخستین جعبهٔ کارنمای شکل ۲-۲۲ نگاه کنید اگر بخواهیم معادل این جعبه را برای کارنمای این مسئله رسم کنیم باید ۱۰۰۰ دستور انتساب قرار دهیم که هر یک از این متغیرها را با صفر مقدار اولیه بدهد. پس از خواندن هر نمره باید هزار جعبهٔ تصمیم تشکیل دهیم تا با خواندن هر نمره، شمارندهٔ مربوط به آن نمره را یک واحد افزایش دهیم. حال به نظر شما چگونه می‌توان این الگوریتم را با این گستردگی پیاده‌سازی کرد؟حتی نوشتن برنامهٔ C++ آن نیز به ۱۰۰۰ دستور **else if** نیاز دارد! بنابراین باید راه دیگری برای این مسئله پیدا کنیم. از طرف دیگر این

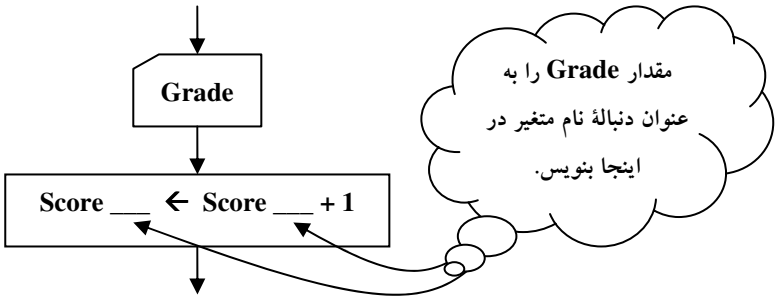
روش الگوریتم‌نویسی یک مشکل اساسی دیگر هم دارد، به‌طوری که اگر بازهٔ نمرات از ۱ تا ۱۰۰۰ به ۱ تا ۵۰۰۰ تغییر کند باید کل الگوریتم را تغییر دهیم تا ۴۰۰۰ نمرهٔ جدید را پوشش دهد و اگر بازهٔ نمرات به ۱ تا ۵۰۰ تقلیل یابد، دوباره نیز باید الگوریتم را تغییر دهیم تا ۵۰۰ جعبهٔ اضافی حذف گردد. پس پاسخ این الگوریتم با کوچکترین تغییری در بازهٔ نمرات ناکارآمد می‌شود. بنابراین بهتر است روشی پیدا کنیم که الگوریتم نه تنها تعداد نمرات را از کاربر دریافت کند، بلکه بازهٔ نمرات را هم از وی دریافت کند و بر اساس آن بازه روند الگوریتم پیش‌رفته، و نتایج حاصل آید.

برای حل این مشکلات بیایید یکبار دیگر ارتباط بین داده‌های ورودی و شمارنده‌ها را مد نظر قرار دهیم. الگوریتم ما بر این مبنا استوار است که اگر بر فرض عدد ۷۶ خوانده شود یعنی متغیر مربوط به دادهٔ ورودی(**Grade**) حاوی عدد ۷۶ باشد به متغیر **Score76** به‌عنوان شمارندهٔ تعداد افراد دارای نمرهٔ ۷۶ یک واحد افزوده شود. قسمتی از الگوریتم که این کار را انجام می‌دهد در شکل ۶-۱ آمده است.



شکل ۶-۱ : نمایی از مراحل الگوریتم شمارش نمرات کنکور

اما نکتهٔ بسیار مهمی که می‌خواهیم شما را در شکل ۶-۱ معطوف به آن سازیم، برابر بودن عدد ۷۶ خوانده شده، با دنباله شمارندهٔ این عدد، است. با توجه به نکتهٔ اخیر می‌توان کارنمای شکل ۶-۱ را به زبان ساده‌تری توصیف کرد. شکل ۶-۱ به ما می‌گوید اگر عدد ورودی برابر ۷۶ می‌باشد به متغیری که نام آن **Score** و دنبالهٔ نام آن ۷۶ است یک واحد اضافه کن. احتمالاً اگر بتوانیم این توصیف را به شکلی پیاده‌سازی کنیم مشکل ما نیز در حل این مسئله رفع خواهد شد. یک شیوه برای پیاده‌سازی این توصیف در شکل ۶-۲ آمده است.



شکل ۶-۲

واقعیت آن است که ایدهٔ مطرح شده در شکل ۶-۲ همان چیزی است که می‌توان به کمک آن این مسئله و مسائل مشابه آن را به‌راحتی حل کرد. اما بیایید روش خود را به شیوه‌ای بهتر و سیستماتیک پیاده‌سازی کنیم. پیشتر در ریاضیات

با متغیرهای زیرنویس‌دار آشنا شده‌اید، و چنانکه در ریاضیات دورهٔ دبیرستان نیز دیده‌اید در برخی از فرمول‌های ریاضی مثل $\sum$ از متغیرهایی استفاده می‌کنیم که دارای زیرنویس هستند. به‌عنوان مثال با فرمول زیر که بیانگر میانگین حسابی چند عدد است از قبل آشنایی دارید:

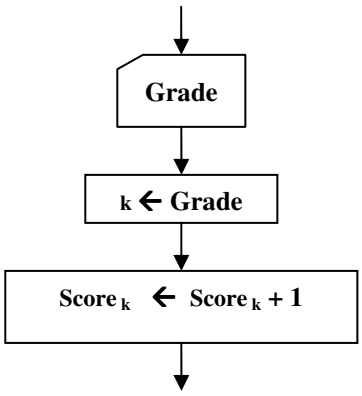
$$\overline{X} = \frac{1}{n} \times \sum_{i=1}^{n} X_i$$

در الگوریتم‌ها هم می‌توانیم چنین متغیرهایی را تعریف کنیم. به‌طوری که یک نام مشترک برای تعدادی متغیر در نظر می‌گیریم و به ترتیب به آنهایک زیرنویس اختصاص می‌دهیم. با تغییر زیرنویس دستیابی ما به متغیر دیگری که دارای آن زیرنویس است، میسر می‌شود. بنابراین می‌توانیم نامگذاری را که در قسمت قبل انجام دادیم به‌صورت زیر تغییر دهیم:

$$Score_1 , Score_2 , Score_3 , \dots , Score_k , \dots , Score_{999} , Score_{1000}$$

در برخی از کتب به مجموعهٔ متغیرهایی که به این روش به‌دست می‌آید متغیرهای لیستی گفته می‌شود، اما در این کتاب به جهت هماهنگی با مباحث زبان C++ بیشتر از لفظ آرایه به‌جای متغیرهای لیستی استفاده شده است.

تا اینجا یک گام دیگر در حل این‌مسئله به جلو رفتیم. چرا که حالا می‌توانیم به‌راحتی متغیر **Grade** که حاوی نمرهٔ ورودی است را به‌عنوان زیرنویس متغیر لیستی خود معرفی کنیم، و با دسترسی مستقیم به شمارندهٔ مربوطه یک واحد بیفزاییم. تکه کارنمایی برای این الگوریتم در شکل ۶-۳ با استفاده از یک متغیر لیستی ارائه شده‌است:

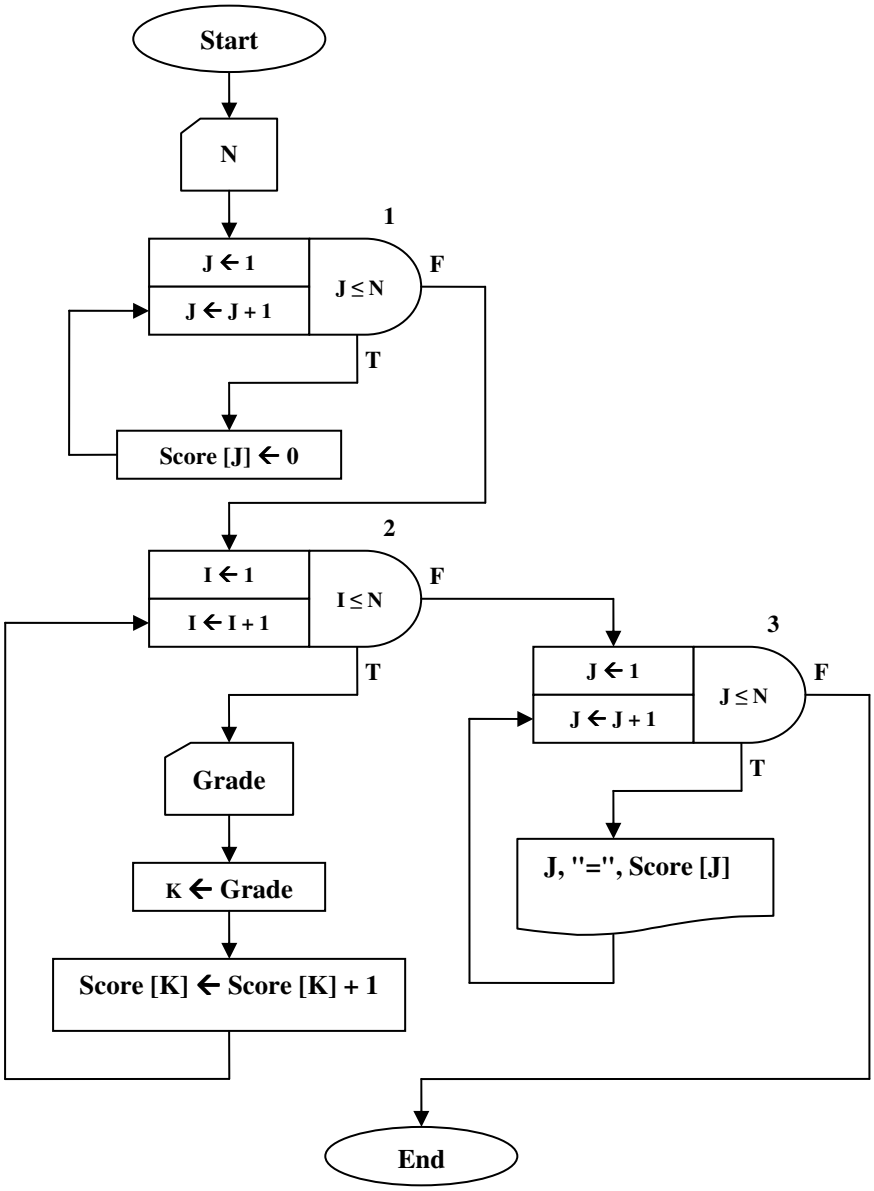


شکل ۶-۳ : چگونگی استفاده از متغیرهای لیستی در الگوریتم

اما به دلایلی که پیشتر در قسمت ۲-۸ مطرح کردیم نمادهایی که از حالت خطی خارج می‌شوند برای کامپیوتر قابل خواندن نیستند، بنابراین مجبوریم زیرنویس‌ها را نیز به‌صورت خطی بنویسیم. لذا به جهت تمایز دادن زیرنویس‌ها از نام متغیر لیستی زیرنویس را بین دو علامت کروشه یعنی [ ] محصور می‌کنیم. بنابراین از این پس نامگذاری متغیرهای لیستی(آرایه‌ها) را مطابق مثال زیر انجام می‌دهیم.

Score[1] , Score[2] , Score[3] , … , Score [k] , … , Score[999] , Score[1000]

در شکل ۶-۴ کارنمای نهایی برای این الگوریتم با استفاده از آرایه‌ها ارائه شده است.

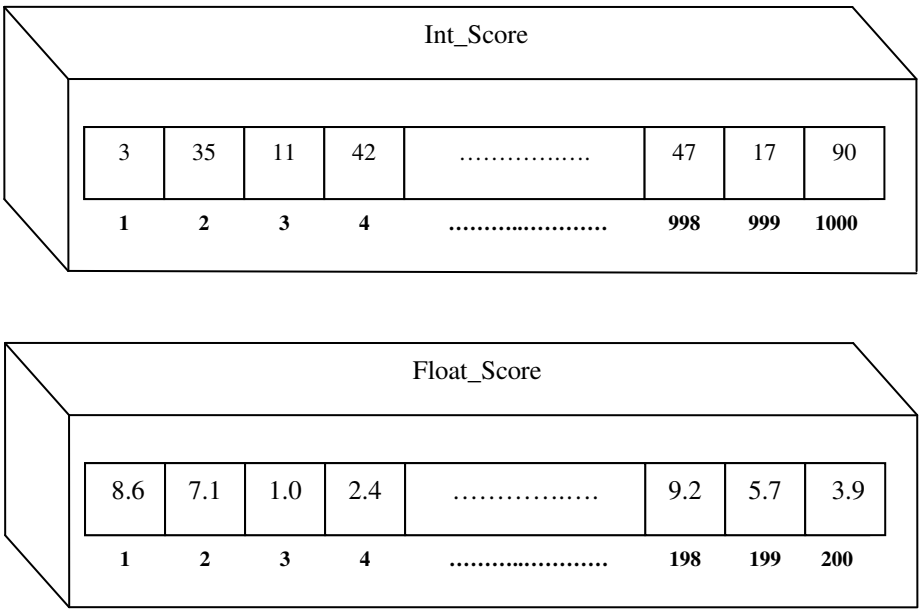


شکل ۶-۴ : کارنمای شمارش تعداد نمرات مختلف کنکور

توضیح کارنما : در اولین حلقهٔ تکرار در کارنمای فوق کلیهٔ متغیرهای آرایه را به صفر مقدار اولیه داده‌ایم سپس در حلقهٔ دوم شروع به خواندن تک تک نمرات کرده‌ایم و چنانکه پیشتر گفته شد با خواندن هر نمره شمارندهٔ معادل آن را یک واحد افزایش داده‌ایم. و در حلقهٔ سوم نیز مقادیر متغیرهای لیست را چاپ کرده‌ایم.

در خصوص آرایه‌ها نکات زیر را مد نظر قرار دهید:

۱. اگر بخواهیم یک نماد برای آرایه‌ها در نظر بگیریم اینگونه مطرح می‌سازیم که؛ آرایه عبارت است از تعدادی خانه از حافظه که به‌صورت متوالی در نظر گرفته می‌شوند. در شکل ۶-۵ نماد آرایه را مشاهده می‌کنید.



شکل ۶-۵ : نمادی برای خانه‌های آرایه به همراه زیرنویس آنها

۲. محتوای آرایه‌ها منحصر به مقادیر نوع صحیح نیست، چنانکه در شکل ۶-۵ می‌بینید، می‌توان مقادیری از نوع اعشاری یا رشته‌ای یا کاراکتری را نیز در آنها ذخیره کرد. ولی همواره همهٔ مقادیر مربوط به عناصر یک آرایه از یک نوع هستند. به‌عنوان مثال نمی‌توان در داخل یک آرایه هم مقادیر صحیح وهم مقادیر اعشاری را ذخیره کرد.

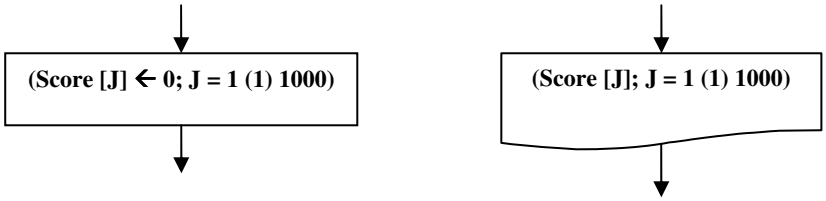
۳. زیرنویس آرایه می‌تواند به‌صورت یک عبارت محاسباتی هم باشد. لذا در کارنما به‌کاربردن زیرنویس‌هایی از قبیل:

A [n+3] , B [2*J*(K−1) +7] , C [FLR (J%k^I)]

کاملاً مجاز است.

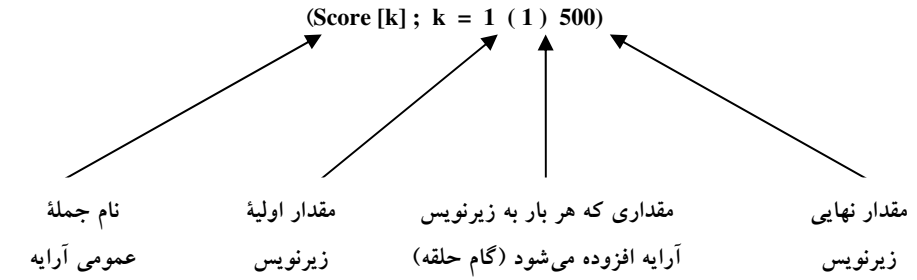
۴. چنانکه در کارنمای شکل ۶-۴ دیدید، یکی از مهمترین مزایای لیست‌ها (آرایه‌ها) پردازش جمعی تمامی عناصر یا بخشی از عناصر لیست در داخل یک حلقهٔ تکرار است. این مزیت ما را قادر می‌سازد که به‌عنوان مثال به‌جای نوشتن تعداد زیادی دستور انتساب، تنها یک دستور انتساب بنویسیم و توسط یک حلقهٔ تکرار آن دستور را برای تمامی عناصر لیست تکرار کنیم. اما اگر قرار باشد برای هر عملیات ترتیبی که می‌خواهیم بر

روی لیست‌ها انجام دهیم مجبور باشیم یک حلقهٔ تکرار در کارنمای خود بگنجانیم، کارنمای ما از صفحات کاغذ بیرون خواهد رفت. بنابراین بهتر است برای عملیات پردازش ترتیبی لیست‌ها نمادهایی را در نظر بگیریم، تا از گنجاندن حلقه‌های تکرار متعدد، معاف گردیم. مقصود ما از پردازش ترتیبی لیست‌ها، هر گونه عملیاتی است که شمارندهٔ حلقهٔ تکرار در حین محاسبات، به‌عنوان زیرنویس آرایه منظور شود. به‌عنوان مثال حلقه‌های تکرار اول و سوم در کارنمای شکل ۶-۴ از نوع پردازش ترتیبی می‌باشند ولی حلقهٔ تکرار دوم از نوع پردازش ترتیبی نیست. برای حلقه‌های اول و سوم کارنمای شکل ۶-۴ می‌توان جعبه‌های زیر را پیشنهاد کرد.



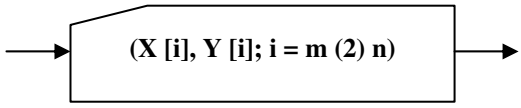
شکل ۶-۶ : نمادی برای پردازش ترتیبی لیست‌ها

چنانکه در این شکل‌ها می‌بینید نوع عملیات توسط همان جعبه‌هایی که پیشتر یاد گرفتید بیان می‌شود. سپس در داخل جعبه، عبارتی مطابق نمونهٔ زیر نوشته می‌شود.

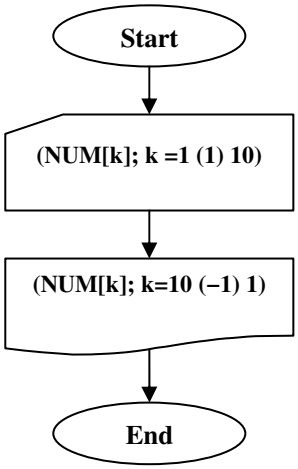


شکل ۶-۷ : جملهٔ داخل جعبه‌ها برای پردازش ترتیبی لیست‌ها

در عبارت فوق، در ابتدا نام لیست نوشته می‌شود، سپس یک علامت سمیکالن قرار می‌گیرد، و معنای قسمت دوم عبارت فوق این است که زیرنویس **k** از ۱ تا ۵۰۰ تغییر می‌کند. میزان تغییر زیرنویس آرایه در هر مرحله برابر مقداری است که در بین دو پرانتز ذکر شده است. بنابراین ممکن است بخواهیم مقدار آرایه را یکی در میان پردازش کنیم، در این صورت مقدار گام را برابر ۲ در نظر می‌گیریم. همچنین امکان پردازش چند لیست در یک جعبه نیز وجود دارد. به‌عنوان مثال جعبهٔ زیر را ببینید:



مثال ۶-۱: الگوریتمی که ۱۰ عدد را از ورودی خوانده و سپس عناصر آرایه را از آخرین عنصر به اولین عنصر به خروجی می‌برد.



شکل ۶-۸ : کارنمای مثال ۶-۱

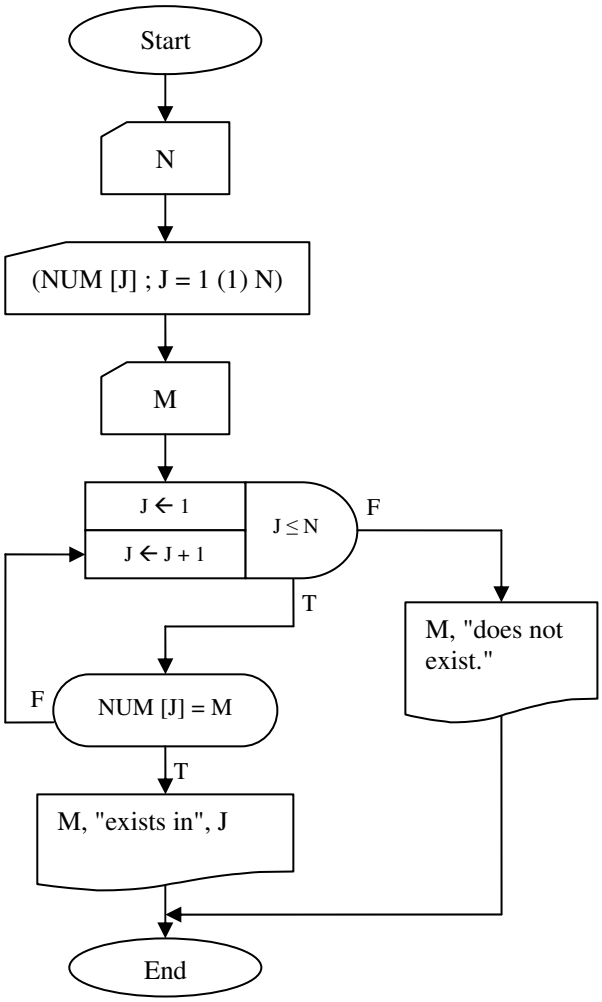
کار در کلاس ۶-۱ :

کارنمای شکل ۶-۴ را یکبار دیگر با حذف حلقه های تکرار اول و سوم بازنویسی کنید. آیا امکان حذف حلقهٔ تکرار دوم نیز وجود دارد؟

جستجوی ترتیبی در یک لیست

پیشتر در فصل دوم با مقدماتی در خصوص الگوریتم‌های جستجو آشنا شدیم، حال در این قسمت با مبحث جستجوی ترتیبی در آرایه‌ها آشنا می‌شویم. مقصود از جستجوی ترتیبی، پردازش عناصر یک لیست از اولین عنصر تا آخرین عنصر آن برای یافتن مکان یک نمونهٔ خاص در آن لیست می‌باشد. در مثال ۶-۲ الگوریتمی در این خصوص ارائه شده است.

مثال ۶-۲ : الگوریتمی که لیستی از اعداد صحیح را دریافت می‌کند و سپس با دریافت یک عدد مشخص می‌کند که این عدد در کجای آرایه قرار دارد. و در صورتی که تا آخر آرایه چک شود و عدد مورد نظر در آرایه یافت نشود، پیغام عدم وجود عدد در لیست چاپ می‌شود.



شکل ۶-۹ : کارنمای مثال ۶-۲

۶-۲: تمرین

۱. الگوریتمی که برای پیدا کردن بزرگترین عدد از میان N عدد، در شکل ۲-۲۱ نشان داده شد، احتیاج ندارد که در هیچ لحظه‌ای بیش از دو مقدار از N مقدار داده را در حافظه نگهداری کند. در اکثر موارد لیستی شامل N مقدار داده در حافظه آماده داریم و مسئله، پیدا کردن بزرگترین آنها است.گاهی هم نه فقط می‌خواهیم بزرگترین عدد را به‌دست آوریم بلکه می‌خواهیم تعیین کنیم که بزرگترین عدد در کجای لیست واقع شده است (زیرنویس آن، چه بوده است).

الف- قطعهٔ کارنمایی بسازید که در صورت اجرا، بزرگترین مقدار را در لیست A شامل N عنصر حقیقی پیدا کرده و آن را چاپ کند. فرض کنید که مقادیر A و N در حافظه آماده هستند.

ب- قطعهٔ کارنمایی را که در قسمت (آ) ساخته شده است طوری تغییر دهید که نه فقط بزرگترین مقدار، بلکه «محل» عنصر انتخاب شده، یعنی زیرنویس آن را نیز چاپ کند. چنانچه بزرگترین مقدار بیش از یک بار در داخل لیست آمده باشد زیرنویس اولین مورد یعنی کوچکترین زیرنویس را چاپ کنید.

۲. کارنمای شکل ۶-۱۰ الگوریتمی است که به سه مرحلهٔ زیر صورت عمل می‌بخشد.

- عدد c را به داخل می‌آورد.
- یک صد عدد b[1], b[2], …, b[100] را به داخل می‌آورد.
- لیست مقادیر b[i] را که در رابطهٔ $b[i] \geq c$ صدق می‌کند تعیین و چاپ می‌کند.

کارنما را به دقت مطالعه کنید و سؤال‌های زیر را پاسخ دهید.

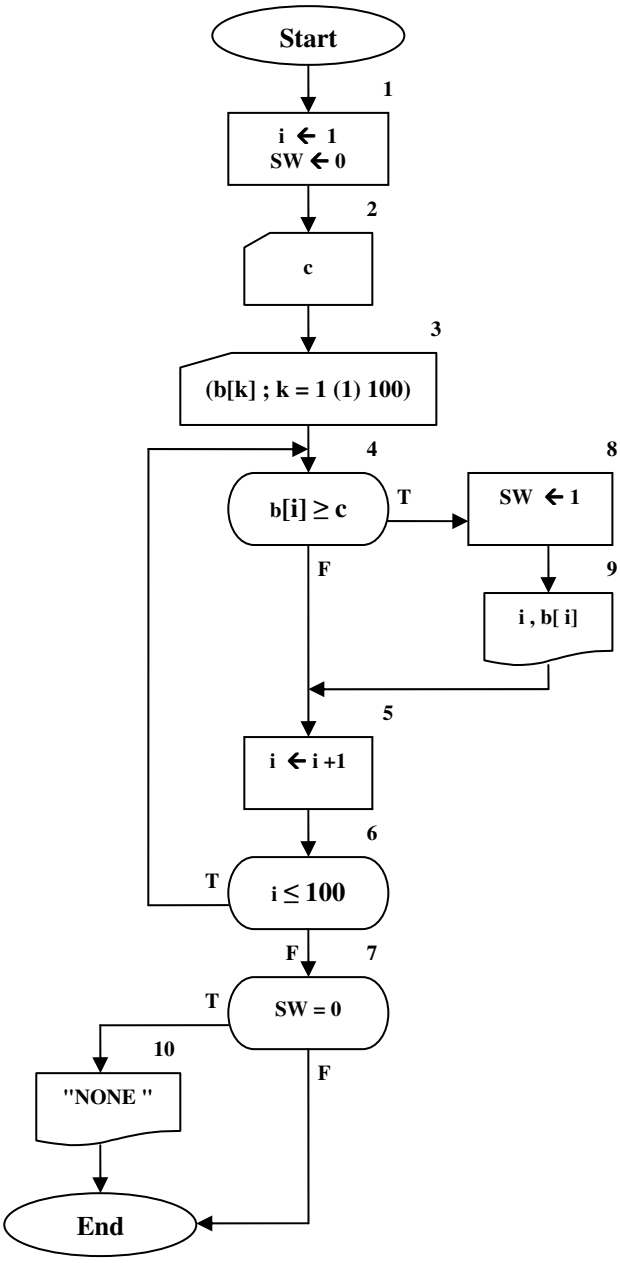
الف- جعبهٔ ۶ چند بار اجرا می‌شود؟

ب- جعبهٔ ۸ چند بار اجرا می‌شود؟

ج- تحت چه شرایطی جعبهٔ ۱۰ اجرا خواهد شد؟ اشاره می‌شود به اینکه SW یک «متغیروضعیت یا سوئیچ» است و مورد استعمال آن شبیه استفادهٔ سوزنبانان از سوزن خطوط قطار، (جهت کنترل حالات مسئله) است. در این خصوص در بخش ۶-۷ بیشتر صحبت خواهیم کرد.

د- آیا واقعاً برای کسب منظور این برنامه نیاز به این هست که در هر لحظه بیش از یکی از مقادیر b در حافظه باشد؟ به عبارت دیگر «آیا وجود آرایه» واقعاً در این الگوریتم لازم است؟ اگر جوابتان منفی است، کارنما را بر آن اساس دوباره رسم کنید و در کنار هریک از جعبه‌هایی که تغییر داده‌اید علامتی برای نشان دادن این امر بگذارید.

و- برای این که به جای خواندن ۱۰۰ عنصر برای b هر تعداد معین n از آنها خوانده شود، کارنمای شکل ۶-۱۰ و یا صورت تغییر یافتهٔ آن را که در قسمت (ث) ترسیم کرده‌اید چگونه تغییر خواهید داد؟



شکل ۶-۱۰ : کارنمای تمرین ۲

۳. سرمایه‌گذاری که مایل است در امر معاملهٔ سهام، یک روش «الگوریتمی» را اتخاذ کند، برای خرید و فروش سهم به قواعد زیر رسیده است.

أ) هرگاه ارزش یک نوع سهم برای دو هفتهٔ متوالی صعود کرد، ۱۰۰ سهم از آن را بخر(حتی اگر در آن حال صاحب سهامی از آن نوع باشی).

ب) اگر ارزش سهامی که داری برای دو هفتهٔ متوالی نزول کرد، همهٔ دارایی خود را از آن سهام بفروش.

او به‌منظور امتحان این خط مشی، دستهٔ داده‌ای شامل قیمت ۵۲ هفتهٔ یک نوع سهام خاص جمع‌آوری کرده است. تکلیف شما این است که کارنمایی بسازید که با فرض این‌که همواره پول لازم برای خریدها موجود باشد، سود یا زیان خالص او را در پایان ۵۲ هفته محاسبه کند. سود یا زیان هنگامی حاصل می‌شود که سهام فروخته می شود و چون سهام همیشه ۱۰۰ تایی خریده می‌شود، سود حاصل (از هر ۱۰۰سهم) برابر ۱۰۰ × (قیمت خرید- قیمت فروش) خواهد بود. (به این ترتیب، سود منفی، زیان است.) اگر سهام در چند نوبت خریداری شود و سپس به فروش برسد آنگاه لازم است سابقهٔ قیمت هر خرید نگهداری شود تا بتوان هنگام فروش، سود یا زیان را به‌طور صحیح حساب کرد. (آیا آرایه مفید خواهد بود؟) اگر در پایان هفتهٔ پنجاه و دوم سهامی داشته باشیم همگی در آن هنگام فروخته و در سود و زیان حاصله محسوب می‌شود. احتمالاً انتخاب متغیری، مثلاً **OWN** ، برای نشان دادن اینکه آیا در حال حاضر دارای سهمی هستیم یا نه، مفید واقع خواهد بود.

۴. کارنمایی رسم‌کنید که مقدار **n** و لیست a_0 ، a_1 ، ... ، a_n را به داخل بیاورد. **a**ها ضرایب چندجمله‌ای

$$a_0 + a_1 X^1 + a_2 X^2 + ... + a_n X^n$$

و **n** درجهٔ ظاهری چندجمله‌ای است. البته برخی و یا تمامی ضرایب ممکن است صفر باشند. کارنمایی بسازید که درجهٔ حقیقی (**m**) چند جمله‌ای را تعیین کند. البته واضح است که $m \leq n$ و **m** را می‌توان با جستجو در مجموعهٔ ضرایب، برای پیدا کردن عنصر غیر صفر دارای بزرگترین زیرنویس، تعیین کرد. مقدار **m** و ضرایب a_0 الی a_m را در خروجی بنویسید. اگر همهٔ ضرایب صفر باشند، هیچ ضریبی چاپ نشود و مقدار **m** برابر ۱- چاپ شود.

۵. برای مسئلهٔ زیر یک راه حل کارنمایی تهیه کنید. فهرست علائم و اختصارات برای متغیرها فراموش نشود. متغیرها را با تأمل انتخاب کنید و نام‌های با معنی برای آنها به‌کار ببرید.

در دانشگاه **x** ، برای درس **Y** دو امتحان میان دوره‌ای و یک امتحان نهایی وجود دارد. حداکثر تعداد امتیازهای هر امتحان ۱۰۰ و حداقل آن صفر است. نمرهٔ نهایی درس برای هر دانشجو میانگین امتیازات امتحان هاست که با منظور کردن ضریب دو برابر برای امتحان نهایی نسبت به امتحان‌های میان دوره‌ای محاسبه می‌شود. لذا نمرهٔ درس نیز عددی در محدودهٔ صفر الی ۱۰۰ (شامل این دو) است. هر دانشجو دارای یک شمارهٔ دانشجویی

اختصاصی از ۰۰۱ تا ۱۰۰ است. تعدادی داده برای پردازش نمرات آماده شده است. این داده‌ها شامل یکسری(یا جریانی از) اعداد دوازده رقمی مانند آنچه در زیر آمده است، هستند.

0 4 5 0 7 5 0 6 2 0 8 9

سه رقم اول هر عدد دوازده رقمی شمارهٔ دانشجویی، سه رقم دوم امتیاز امتحان میان دوره‌ای اول، سه رقم سوم امتیاز امتحان میان دوره ای دوم، و بالاخره سه رقم آخر امتیاز امتحان نهایی آن دانشجو است. بعضی دانشجویان موفق به شرکت در هیچ امتحانی نشده‌اند و لذا شمارهٔ دانشجویی آنها جزء داده‌ها نیست.

اجرای کارنمای شما باید موجب شود که کل داده‌ها خوانده و نمره نهایی، در موارد ممکن، محاسبه و لیست نمرات دانشجویان کلاس چاپ شود. ورودی برحسب شمارهٔ دانشجویی مرتب نشده است. لیست نمرات باید به ترتیب شمارهٔ دانشجویی و به‌صورتی شبیه آنچه که در زیر نشان داده شده است، باشد.

Student Code Number	Grade
001	96
002	70
003	No scores found
004	85
.	.
.	.
.	.
100	75

شکل ۶- ۱۱ : نمونهٔ خروجی تمرین ۵

در بالای هر ستون باید عنوانی برای مشخص کردن آن چاپ شود. اگر برای یک شمارهٔ دانشجویی نمره‌ای یافت نشود باید در محل مربوط به نمرهٔ نهایی پیام «**No scores found**» چاپ شود. از آنجا که تعداد داده‌ها ممکن است دقیقاً ۱۰۰ تا نباشد، دادهٔ نگهبانی شامل صفر در انتهای داده‌ها قرار داده خواهد شد. ابتدا دربارهٔ مسئله تأمل کنید تا داده‌ها و خواسته‌های آن کاملاً بر شما معلوم شود. کارنمای شما باید ورودی ۱۲ رقمی را در چهار گروه سه رقمی از هم جدا کند.

گرچه در مباحث آتی به‌طور مفصل در خصوص مرتب‌سازی صحبت خواهیم کرد، اما مسئلهٔ حاضر را با استفاده از مرتب کردن حل نکنید چون که راه حل بسیار ساده‌تری برای آن وجود دارد.

۶. با استفاده از یک جعبهٔ WHILE، کارنمایی برای بیان پردازش زیر رسم کنید. لیستی از اعداد مثبتی به‌صورت (B[i] ; i = 1 (1) n) داده شده است، لیست را از B[1] تا B[n] به ترتیب جستجو و اولین عنصر B[j] را که حداقل دو برابر عنصر بلافاصله قبل از خود یعنی B[j–1] است، پیدا کرده و مقدار آن را چاپ کن. در صورتی که چنین عنصری یافت نشود کلمهٔ «نیست» را چاپ کن.

۷. با استفاده از نوعی جعبهٔ کنترل حلقه، الگوریتم کاملی بنویسید که همهٔ کارهای زیر را انجام بدهد.

الف– M مقدار عددی برای لیست VEC به داخل بیاورد. (این مقادیر ورودی از قبل به ترتیب عددی غیر صعودی مرتب شده‌اند.)

ب– با شروع از VEC[1]، لیست را برای پیدا کردن یک جفت عدد تکراری مورد جستجو قرار دهد.

ج– اگر یک جفت عدد تکراری پیدا شد، مقدار تکرار شده و زیر نویس اولین عنصر جفت را چاپ کند و متوقف شود.

د– اگر جفت عدد تکراری پیدا نشد، با چاپ پیام مقتضی متوقف شود.

۸. از آنجا که در کامپیوتر هیچ روش سخت‌افزاری برای محاسبهٔ ریشهٔ n–اُم وجود ندارد اصولاً ریشهٔ n– اُم عدد A را با استفاده از روش تکراری زیر محاسبه می‌کنند. الگوریتمی بنویسید که مقدار A و n را از ورودی خوانده و ریشهٔ n–اُم عدد A را با تقریب ۰/۰۰۱ محاسبه و چاپ کند. X_0 را برابر یک انتخاب کنید.

$$X_i + 1 = \frac{1}{n} \times (n - 1) \times X_i + \frac{A}{X_i^n - 1}$$

۹. الگوریتمی بنویسید که دو عدد مبنای ۲ را با هم جمع کرده و سپس به خروجی ببرد. طول دو عدد مبنای دو ۳۲ رقم است.

۱۰.الگوریتمی برای تفریق دو عدد ۱۰ رقمی در مبنای ۱۰ طراحی کنید. این مسئله را یک بار برای تفریق معمولی و یکبار با استفاده از روش مکمل–۹ حل کنید.

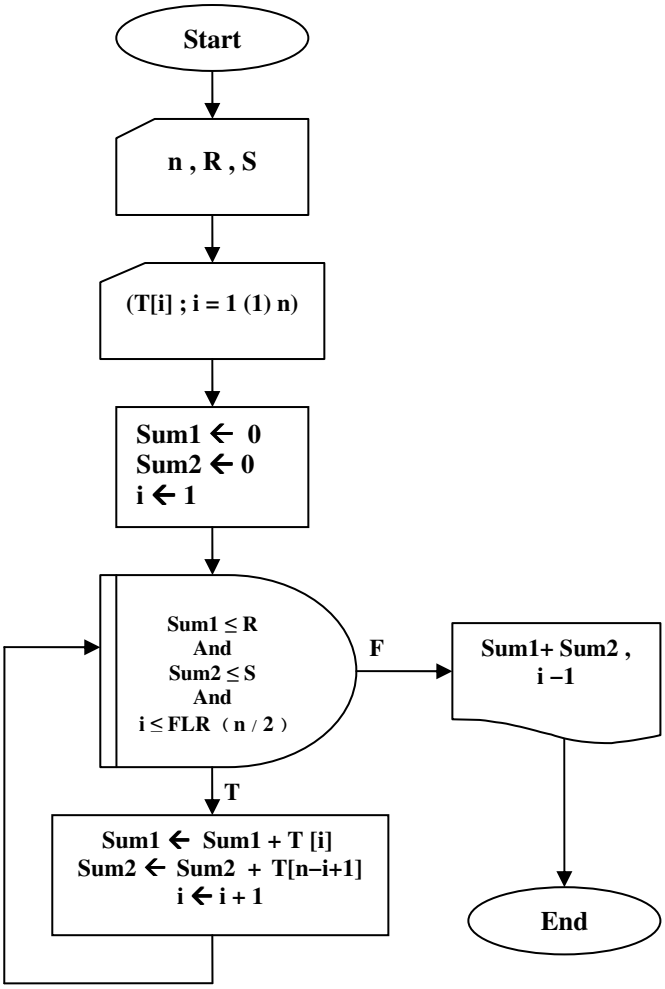
راهنمایی : برای اطلاع از روش مکمل–۹ می‌توانید از استاد خود کمک بگیرید و یا آن که به فصل اول کتاب "طراحی دیجیتال" نوشتهٔ موریس مانو مراجعه کنید.

۱۱.الگوریتمی بنویسید که ضرایب و توان‌های دو چند جمله‌ای را به‌صورت ۴ لیست مجزا خوانده و سپس حاصل‌جمع و حاصل‌ضرب آنها را در لیست‌های جدیدی ذخیره کند و به خروجی بفرستد.

۱۲.الگوریتمی بنویسید که روز اول یک سال (از نظر روز هفته) خوانده و سپس تقویم این سال را تولید کند و در خروجی چاپ کند. آیا استفاده از لیست در این الگوریتم ضروری است؟

الگوریتمی بنویسید که در یک لیست از اعداد، عددی را که بیشتر از همه تکرار شده را مشخص کند و به‌همراه تعداد دفعات تکرار آن به خروجی ببرد.

۱۳.به زبان سادهٔ فارسی توضیح دهید که با اجرای کارنمای زیر چه عملی انجام می‌شود.



شکل ۶- ۱۲ : کارنمای تمرین ۱۳

۱۴.آرایه‌ای مرکب از **n** عدد صحیح داریم، این آرایه را در دو آرایه جدید کوچکتر چنان افراز کنید که اندازهٔ هر یک **n/2** باشد، به‌طوری که اختلاف میان حاصل جمع اعداد صحیح موجود در دو آرایه کوچکتر، حداقل باشد. الگوریتم باید هم برای **n** های زوج و هم برای **n** های فرد درست عمل کند.

۱۵.مسئلهٔ قبل را برای حالتی که اختلاف بین دو آرایهٔ کوچکتر حداکثر باشد نیز حل کنید.

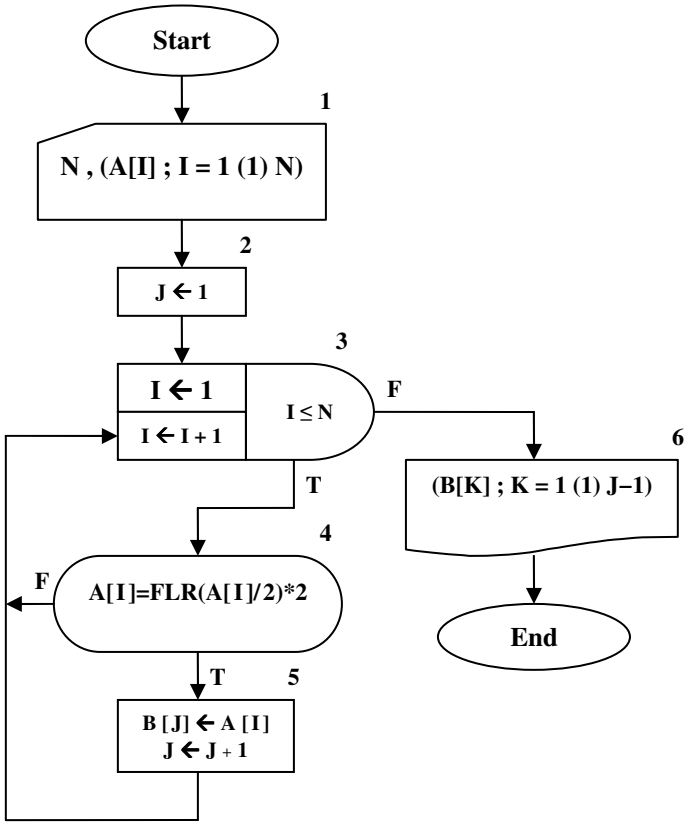
۱۶. در شکل ۶-۱۳ کارنمایی به شما داده شده است.

الف- به زبان ساده بگویید این الگوریتم چه کاری انجام می‌دهد؟ توضیح شما واضح و روشن باشد.

ب- فرض کنید در جعبهٔ ۱ شش مقدار ورودی زیر به لیست A منسوب شده باشد.

۳ ، -۴ ، ۶ ، ۶ ، ۷ ، ۱۲

چه مقادیری (در صورت وجود) هنگام اجرای جعبهٔ ۶ چاپ می‌شود؟



شکل ۶- ۱۳ : کارنمای تمرین ۱۶

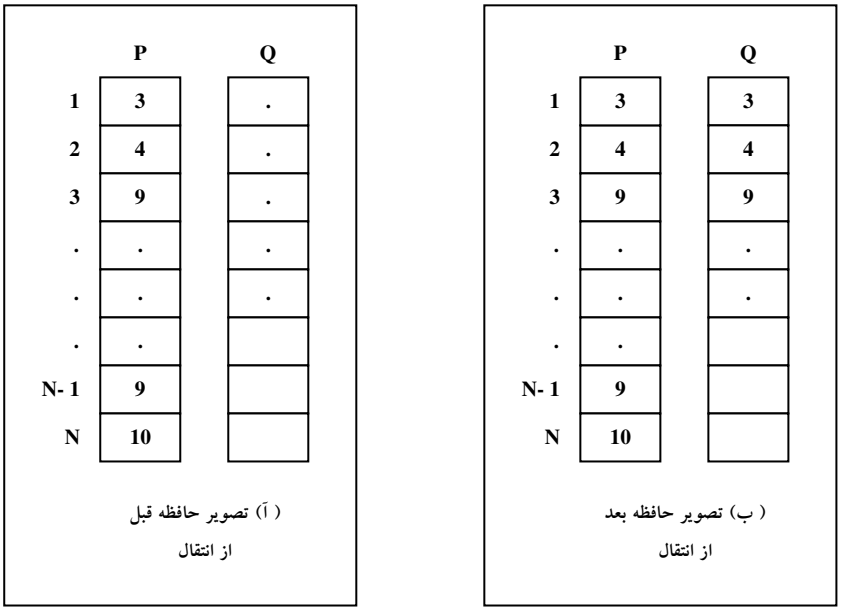
در مسائل ۱۷ تا ۲۳ دو لیست که هر کدام شامل N عدد است در حافظه دارید. یک لیست را P و دیگری را Q نامیده‌ایم. وظیفهٔ شما این است که بیان کلامی هر مسئله را به قطعه کارنمایی که معادل آن باشد تبدیل کنید. جعبهٔ تکرار را در کارنما مفید خواهید یافت. شما ممکن است بخواهید به این شکل عمل کنید که ابتدا قسمت محاسبهٔ حلقه (الگوریتم واحد) را به کارنما تبدیل کنید و سپس آن را به جعبهٔ تکرار مناسبی وصل کنید و بالاخره در صورت لزوم قبل از جعبهٔ تکرار، جعبهٔ تعیین مقادیر اولیه را قرار دهید.

۱۷.مقادیر عناصر**i**- اُم لیست‌های **P** و **Q** را به‌صورت جفت **P[i]** و **Q[i]** تصور کنید. مقادیر موجود در هریک از این جفت‌ها را با هم عوض کنید.

۱۸.کارنمایی را که برای مسئله قبل رسم کرده‌اید طوری تغییر دهید که عمل تعویض فقط برروی جفت‌های با زیرنویس زوج انجام گیرد. آیا زوج یا فرد بودن **N** تأثیری دارد؟

۱۹.کارنمایی را که برای مسئلهٔ ۱۸ رسم کرده‌اید، با این فرض که می‌خواهید از جفت پنجم شروع و مقادیر جفت‌ها را دو در میان با هم عوض کنید، به نحو مقتضی تغییر دهید.

۲۰.نسخه‌ای از تعداد **FLR(N/2)** عنصر اول لیست **P** را به لیست **Q** منتقل کنید. به شکل ۶-۱۴ نگاه کنید.

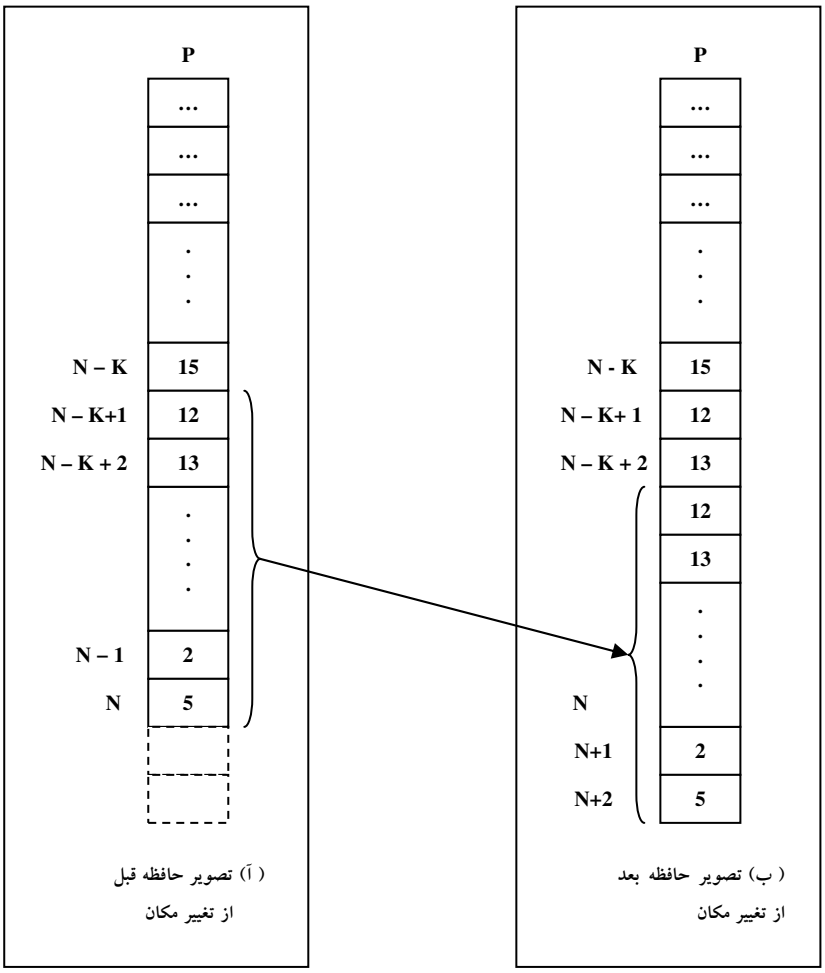


شکل ۶- ۱۴ : تصویر انتقال عناصر لیست

۲۱.تعداد **FLR(N/2)** عنصر آخر لیست **P** را به **FLR(N/2)** محل اول لیست **Q** منتقل کنید. فرض کنید **N** زوج است.
توجه : زیرنویس اولین عنصری از **P** که باید منتقل شود چیست؟

۲۲.مسئلهٔ قبل را بدون فرض زوج بودن **N** یک بار دیگر حل کنید.

۲۳. **K** عنصر آخر لیست **N** عنصری **P** را دو محل به پایین «تغییر مکان» دهید تا جا برای فرار دادن مقادیر جدید در محل‌های **N–K+1** و **N–K+2** باز شود (شکل ۶-۱۵ را ببینید).



شکل ۶-۱۵: تصویر تغییر مکان عناصر لیست

۲۴.الگوریتمی بنویسید که تعدادی شماره حساب، موجودی این شماره حساب‌ها و کد نوع حساب (۱:جاری و ۲:پس انداز) را خوانده و در یک سری لیست قرار دهد. حساب‌های جاری و پس انداز را تفکیک کند، ابتدا حساب‌های جاری و سپس حساب‌های پس‌انداز را به خروجی ببرد.

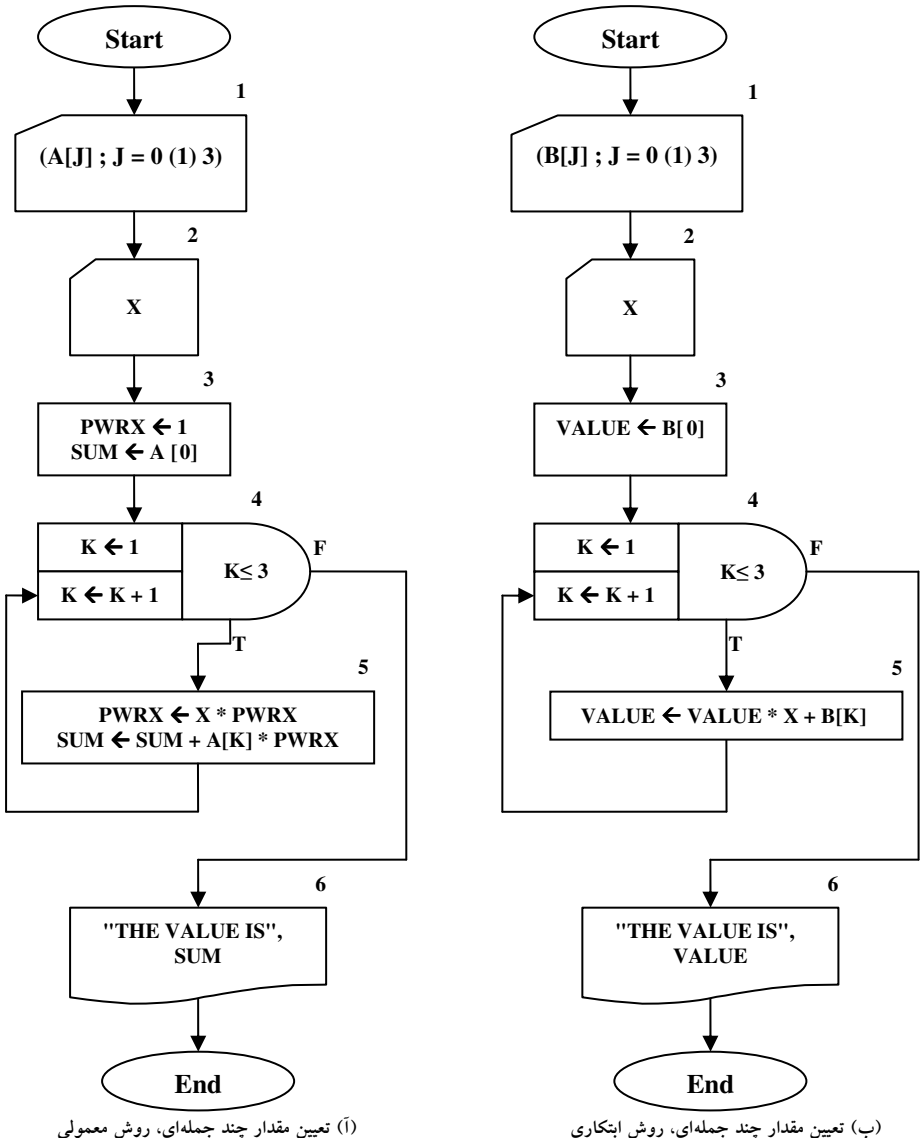
۲۵.الگوریتمی بنویسید که اعمال ریاضی ترکیب و انتخاب را برای دو ورودی a و b انجام دهد. برروی کارآمدترین الگوریتم بحث کنید. مسلماً محاسبهٔ تعدادی فاکتوریل و سپس ساده‌سازی عاقلانه نیست، چرا که ممکن است فاکتوریل‌های به‌دست آمده از محدودهٔ اعداد قابل قبول برای int فراتر رود. لذا باید به روشی فاکتوریل‌ها را ساده کنید. احتمالاً آرایه‌ها کمک فراوانی می‌توانند به حل این مسئله بکنند. مراقب باشید که ترکیب و انتخاب رابطهٔ ریاضی با یکدیگر دارند! حال به نظر شما بهتر است کدام یک را با روش الگوریتمی و کدام یک را با روابط ریاضی از روی دیگری به‌دست آوریم!؟

۲۶.دو کارنمای شکل۶-۱۶ را که برای تعیین مقدار چند جمله‌ای درجهٔ سه داده شده‌اند، مطالعه کنید و سؤالات زیر را پاسخ دهید:

الف- آیا دو کارنمای مزبور از نظر اثر نهایی معادل‌اند؟ شرح دهید.

ب- چه تغییراتی در هریک از این کارنماها لازم است داده شود تا تعیین مقدار چند جمله‌ای‌ها از هر درجهٔ N را میسر شود؟

ج- کدام کارنما کاراتر است؟ (یعنی تعداد کمتری محاسبه لازم دارد؟) شرح دهید.



(آ) تعیین مقدار چند جمله‌ای، روش معمولی

(ب) تعیین مقدار چند جمله‌ای، روش ابتکاری

شکل ۶- ۱۶ : کارنماهای تمرین ۲۶

در هریک از سه تمرین زیر، هر جا که مناسب باشد از جعبهٔ **WHILE** استفاده و فرض کنید که قبلاً **N** مقدار به لیست **P** منسوب شده و در حافظه موجود است.

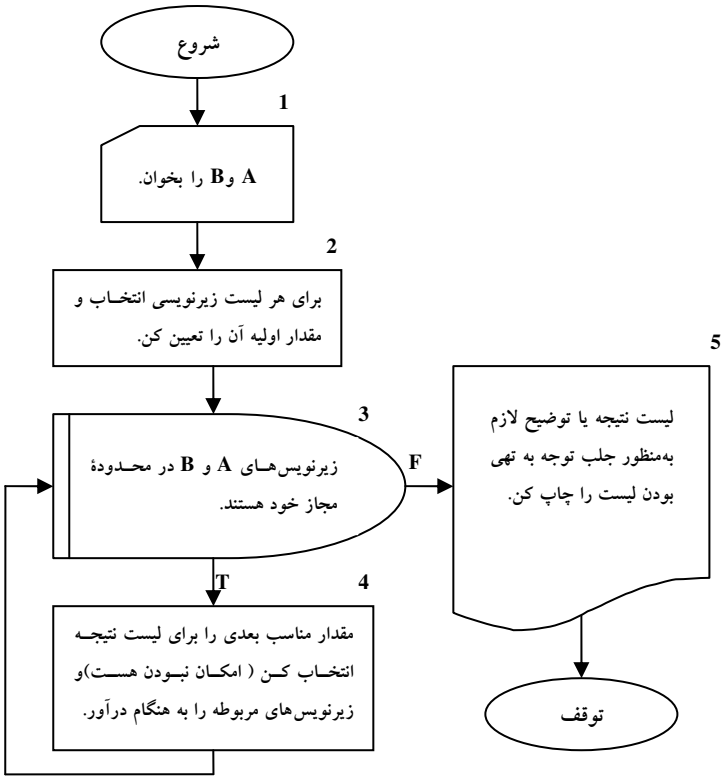
۲۷.لیست را در جهت صعودی زیرنویس‌ها، یعنی **P[1]** ، **P[2]** و ... برای پیداکردن اولین مقداری که قدرمطلق آن بزرگتر از ۵۰ باشد جستجو کرده و در صورت یافتن چنین مقداری آن را به **W** و عدد ۱ را به **ANY** منسوب کن. در صورتی که یک چنین مقداری یافت نشد مقدار صفر را به **ANY** منسوب کن، در هر صورت، پس از این مرحله به نقطهٔ مشترکی از کارنما برو.

۲۸.لیست را به‌طور وارونه، یعنی **P[N]** ، **P[N–1]** و ... برای پیدا کردن اولین عنصری که قدرمطلق آن بزرگتر از ۵۰ باشد مورد جستجو قرار بده و در صورت پیدا شدن آن را به **W** منسوب کن. اگر چنین مقداری یافت نشد، مقدار ۵۰ را به **W** منسوب کن. در هر دو صورت پس از این مرحله، به نقطهٔ مشترکی از کارنما برو.

۲۹.تمام **N** عنصر فهرست **P** را برای پیدا کردن عنصر مخالف صفری که دارای بزرگترین قدرمطلق کوچکتر از **|M|** باشد مورد جستجو قرار بده. فرض کن که مقدار **M** از قبل در حافظه ذخیره شده است. مقدار عنصری را که در این جستجو پیدا می‌شود به **T** منسوب کن. در صورتی که چنین عددی پیدا نشد پیغام «پیدا نمی‌شود» را چاپ کن و متوقف شو.

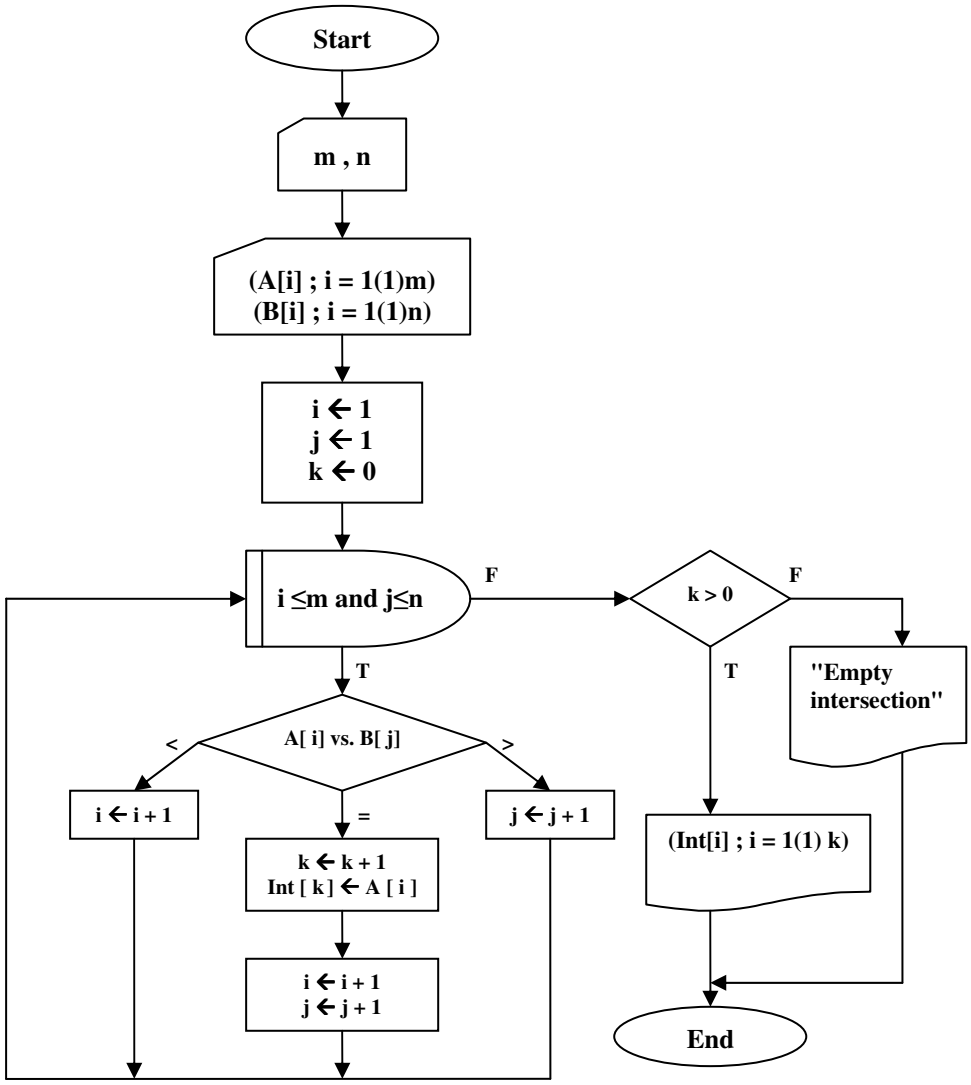
۶-۳ : عملیات برروی مجموعه ها

در اینجا توجه شما را به یک دسته از عملیاتی که در دنیای واقعی پردازش داده‌ها (اعم از این‌که برروی کامپیوتر باشد یا نه) از اهمیت بنیادی برخوردار است، معطوف می‌نیم. مثال‌ها و مسائلی را در این قسمت خواهید دید که مبین عملیات اشتراک، اجتماع و ادغام دو مجموعه هستند. در تمامی مباحث این قسمت فرض کنید، **A** یک لیست **m** عنصری از اعداد و **B** یک لیست **n** عنصری از اعداد است. به علاوه مقادیر موجود در لیست‌ها قبلاً از نظر عددی به ترتیب صعودی مرتب شده‌اند. الگوریتم‌های اشتراک، اجتماع و ادغام دارای شباهت‌های ساختی معینی هستند، لذا کارنمای تشریحی زیر روند کلی حل این سه دسته مسئله را نشان می‌دهد. از طرفی به جهت شباهت مذکور ما در اینجا تنها عملیات اشتراک دو لیست را به‌عنوان نمونه حل می‌کنیم و عملیات اجتماع و ادغام به‌عنوان تمرین بر عهدهٔ خود دانشجویان گذاشته می‌شود.



شکل ۶- ۱۷ : کارنمای تشریحی عملیات مجموعه‌ای

مثال ۶-۳ : اشتراک: دو لیست **A** و**B** داده شده است. کارنمای الگوریتمی را بسازید که لیست جدید **Int** را طوری تشکیل دهد که عناصر آن دربرگیرندهٔ اشتراک مجموعه‌هایی باشد که توسط **A** و **B** تعریف شده‌اند. در اینجا فرض می‌کنیم که در هیچیک از دو لیست مقادیر تکراری وجود ندارد. منظور ما از اشتراک این است که **Int** شامل نسخه‌ای از مقادیری است که در هر دو لیست **A** و **B** مشترک هستند. مقادیری که در **Int** قرار داده می‌شود باید دارای ترتیب عددی صعودی باشد. توجه داشته باشید که حداکثر تعداد اجزایی که می‌تواند در لیست اشتراک **Int** بیاید محدود است به حداقل **m** و **n** که به ترتیب ابعاد **A** و **B** هستند.



شکل ۶- ۱۸ : کارنمای مثال ۶-۳

۶-۴ : تمرین

۱. اجتماع: دو لیست مرتب شدهٔ **A** و **B** داده شده است، کارنمای الگوریتمی را بسازید که لیست جدید **U** را طوری تشکیل دهد که عناصر آن دربرگیرندهٔ اجتماع مجموعه‌هایی باشد که توسط **A** و **B** تعریف شده‌اند. فرض بر این است که در هیچیک از دو لیست مقادیر تکراری وجود ندارد. بدیهی است منظور ما از اجتماع این است که **U** شامل یک نسخه از هر مقدار متمایزی است که در ضمن پویش همهٔ عناصر **A** و همهٔ عناصر **B** پیدا شود. مقادیری که در **U** قرار می‌گیرند باید به ترتیب صعودی باشند. توجه کنید که اجتماع **U** حداکثر شامل **n + m** عنصر خواهد بود.

۲. ادغام: دو لیست مرتب شدهٔ **A** و **B** داده شده است. کارنمای الگوریتمی را بسازید که لیست جدید **W** را مرکب از همهٔ مقادیر **A** و **B** به‌گونه‌ای تشکیل دهد که نتیجه نیز به‌صورت مرتب شده باشد. به‌عنوان مثال، اگر

A = { ۱۰ و ۵ و ۴ و ۳ و ۲ و ۱ }

و

B = { ۱۲ و ۱۰ و ۶ و ۲ }

آنگاه:

W = { ۱۲ و ۱۰ و ۱۰ و ۶ و ۵ و ۴ و ۳ و ۲ و ۲ و۱ }

الف- در حل مسئله ابتدا فرض کنید که هریک از دو لیست فاقد مقادیر تکراری است.

ب- حال تصمیم بگیرید که در صورت لزوم، چه تغییراتی باید برای کار در مواردی که مقادیر تکراری در A یا B یا در هر دو موجود باشد، داده شود.

۳. اجتماع و اشتراک به‌طور همزمان: در مواردی که برای یک جفت لیست **A** و **B** ، اجتماع **U** و اشتراک **Int** هر دو را بخواهیم، می‌توانیم با ترکیب قسمت‌هایی از الگوریتم‌های قبلی و تشکیل الگوریتم واحدی که طی یک پویش **A** و **B** هر دو لیست **U** و **Int** را ایجاد کند، صرفه‌جویی قابل ملاحظه‌ای به عمل آوریم. ابتدا کارنمای تفصیلی این الگوریتم دو منظوره را بسازید. و سپس کارنمای عادی آن را ترسیم کنید.

۶-۵ : مرتب سازی و جستجو

مرتب‌سازی[1] لیستی از داده‌ها و جستجو[2] برای یافتن یک نمونهٔ خاص در یک لیست دو عمل بسیارمهم در علوم کامپیوتر هستند. شاید سخنی گزاف نباشد که بگوییم ۹۰٪ عملیات بسیاری از مسائل کامپیوتری را مرتب‌سازی و جستجو تشکیل می‌دهد. چیزی که در این زمینه بسیار مهم است یافتن کارآمدترین الگوریتم‌ها برای این دو عمل کلیدی در برنامه‌های کامپیوتری است. به‌عنوان مثال پیشتر با الگوریتم جستجوی ترتیبی که اصولاً در لیست‌های نامرتب استفاده می‌شود آشنا شدید. آیا فکر می‌کنید این الگوریتم برای یافتن یک نمونهٔ خاص در یک لیست، الگوریتمی کارآمد است؟ واقعیت این است که اگر بخواهیم تحلیلی کوتاه برروی الگوریتم جستجوی ترتیبی انجام دهیم، با فرض اینکه عمل مقایسهٔ انجام شده در جعبهٔ تصمیم در کارنمای شکل ۶-۹ تنها عمل وقتگیر در این الگوریتم باشد، چنین مطرح می‌سازیم که؛ بهترین حالتی که ممکن است برای الگوریتم جستجوی ترتیبی رخ دهد، هنگامی است که نمونهٔ مورد نظر در ابتدای لیست قرارداشته باشد. در این‌صورت اگر زمان لازم برای یک عمل‌مقایسه در این الگوریتم برابر α باشد، بهترین زمان ممکن برای انجام این الگوریتم برابر 1α در نظرگرفته‌می‌شود. از طرفی بدترین حالت زمانی رخ می‌دهد که نمونهٔ مورد نظر یا در لیست نباشد و یا آنکه در انتهای لیست قرار داشته‌باشد. در هر دوی این حالات عمل مقایسه در الگوریتم جستجوی ترتیبی باید n بار صورت پذیرد تا الگوریتم به حالت توقف برسد، که n طول لیست مورد پردازش می‌باشد. در این صورت زمان لازم برای اجرای این الگوریتم برابر $n.\alpha$ می‌باشد. لذا نتیجه‌می‌گیریم به‌طور میانگین برای یافتن یک نمونهٔ خاص باید نیمی از لیست را مورد پردازش قراردهیم. حال تصور کنید مقدار α برابر ۰/۰۰۰۱ ثانیه و طول لیست مورد پردازش برابر یک میلیون نمونه باشد. به‌طور متوسط برای یافتن یک نمونهٔ خاص در این لیست باید ۵۰ ثانیه وقت صرف‌کنیم، چیزی در حدود یک دقیقه!؟ اما اگر در موتورهای جستجوی اینترنتی مثل **Google** مطلبی را **Search** کنید، زمان جستجو در بین چند ده میلیارد سایت موجود در اینترنت را چیزی کمتر از حتی یک ثانیه اعلام می‌کند. بنابراین می‌توانید تصور کنید که زمان جستجو تا چه حد در عملیات کامپیوتری می‌تواند مؤثر باشد. چه بسا اگر قرار بود، ما برای جستجوی یک مطلب در اینترنت بیش از ده ثانیه معطل می‌شدیم از جستجوی خود به کلّی منصرف می‌گشتیم.

الگوریتم‌های متنوعی برای عملیات جستجو کشف شده و حتی بسیاری از آنها به دلیل رقابت شرکت‌های تجاری در دسترس عموم قرار نگرفته است، اما بسیاری از آنها فراتر از سطح این کتاب است و در دروسی همچون ساختمان‌داده ها و طراحی‌الگوریتم‌ها مطرح می‌شود. لذا در این کتاب تنها به بیان جستجوی دودویی بسنده می‌کنیم، و مطالعهٔ بیشتر در این زمینه را به خوانندگان عزیز محول می‌سازیم. اما نکتهٔ بسیار مهمتر آن که بسیاری از روش‌های جستجو از جملهٔ جستجوی دودویی با این فرض بنا شده‌اند که لیست مورد پردازش از قبل مرتب شده است. لذا قبل از آنکه به بحث در خصوص جستجوی دودویی بپردازیم به بیان مطالبی در خصوص مرتب‌سازی می‌پردازیم.

---

1. Sort
2. Search

قبل از این که هر گونه الگوریتمی را در خصوص مرتب‌سازی و جستجو بیان کنیم از شما می‌خواهیم که کاردر کلاس زیر را تنها با تکیه بر خلاقیت‌های فردی و آموزه‌هایی که تا اینجا فراگرفتید انجام دهید.

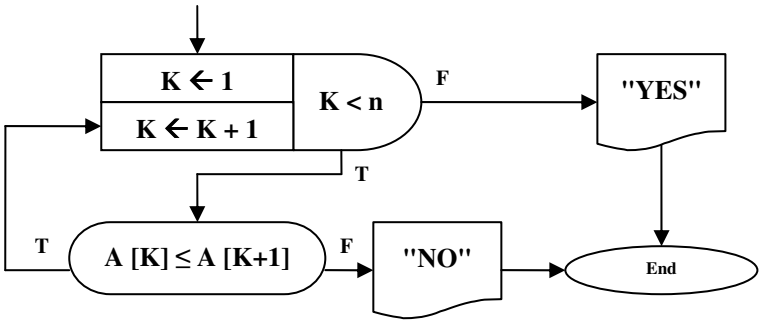
کار در کلاس ۶-۲ :

کارنمایی برای مرتب کردن یک آرایه از اعداد حقیقی به طول **n** *رسم کنید.*

نخستین تلاش برای ساخت یک الگوریتم مرتب‌سازی

شما بارها و بارها در زندگی خود عمل مرتب‌سازی را انجام داده‌اید. از مرتب‌سازی ورق‌هایی بازی که در یک دست به شما می‌افتند گرفته تا مرتب سازی اسکناس‌های داخل کیف پولتان و یا مرتب‌سازی کتاب‌های درسیتان. هر کدام از این مرتب‌سازی‌ها بر اساس یک معیار خاص انجام می‌شود. اما مسلماً هدف و منظور ما از مرتب‌سازی آن مرتب سازی نیست که به منظم کردن اتاق یا کتاب‌های درسی و امثال آن اطلاق می‌شود. مرتب‌سازی که مدنظر ما است مرتب سازی است که در نتیجهٔ وجود «رابطهٔ هم ارزی ترتیب» به‌دست می‌آید. هرگاه برروی دسته‌ای از اشیاء بتوان رابطهٔ هم ارزی $\preceq$ را تعریف کرد به آن یک رابطهٔ ترتیب گفته می‌شود. به‌عنوان مثال از آنجا که عمل «کوچکتر یا مساوی» بر روی اعداد حقیقی تعریف شده است می‌توان گفت اعداد حقیقی دارای رابطهٔ ترتیب هستند. اما اعداد مختلط دارای این رابطه نیستند. مسلماً چنین رابطه‌ای در بین اشیاء یک اتاق نیز وجود ندارد و نظم و ترتیب آنها از یکسری قواعد بشری نشئت می‌گیرد. هر کجا که بتوان رابطهٔ ترتیب را تعریف کرد به دنبال آن دو نوع ترتیب صعودی و نزولی نیز بین اشیاء دارای این رابطه مطرح می‌شود. به‌عنوان مثال شما به دفعات لیستی از اعداد را به‌صورت صعودی یا نزولی مرتب کرده‌اید. اما ترتیب صعودی یا نزولی بودن نمونه‌های مذکور هیچ اهمیتی برای ما ندارد، چرا که اگر به‌عنوان مثال بتوان اشیائی را به‌صورت صعودی مرتب کرد با کمی تغییرات در الگوریتم مرتب‌سازی می‌توان خروجی الگوریتم را به‌گونه‌ای تغییر داد که ترتیب لیست به‌صورت نزولی باشد. لذا در اینجا نخستین تلاش خود را برای ساخت کارنمایی که لیستی از اعداد حقیقی را به‌صورت غیرنزولی مرتب‌سازی کند معطوف می‌سازیم.

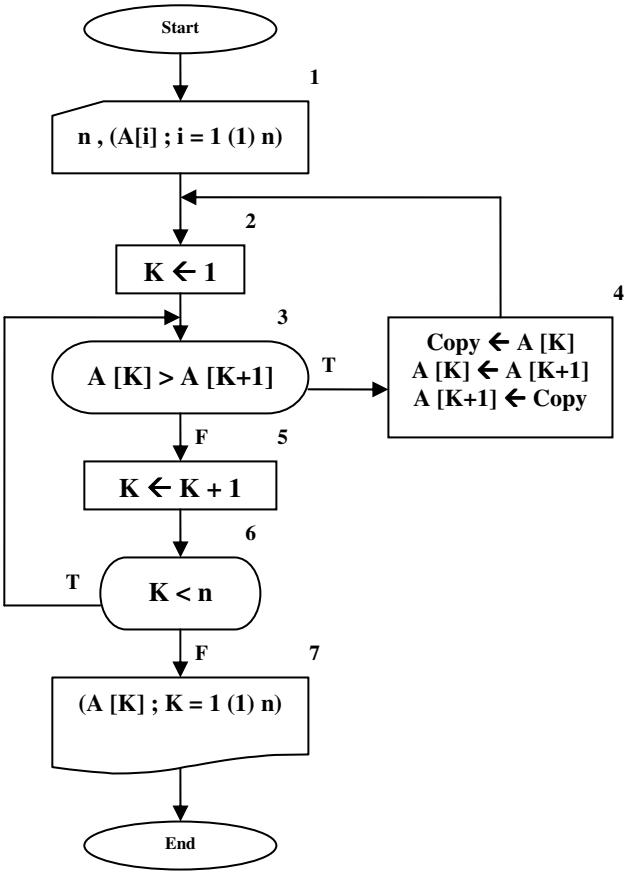
برای حل این مسئله ابتدا این سؤال را مطرح می‌کنیم که شرط مرتب بودن یک لیست به‌صورت صعودی چیست؟ پاسخ این سؤال بسیار روشن است! یک لیست از اعداد حقیقی به شرطی به‌صورت صعودی مرتب است که اگر از ابتدا تا انتهای لیست، هر عدد را با عدد بعد از خود مقایسه کنیم، عددی که در مکان پایین‌تر قرار دارد کوچکتر یا مساوی عددی که در مکان بالاتر است، باشد. در شکل ۶-۱۹ قطعه کارنمایی ارائه شده است که مرتب بودن یک آرایه از اعداد حقیقی به طول **n** را تشخیص می‌دهد.



شکل ۶- ۱۹ : قطعه کارنمای تشخیص مرتب بودن یک لیست

در شکل ۶-۲۰ کارنمایی به‌منظور مرتب‌سازی لیستی از اعداد حقیقی ارائه شده است. در این کارنما از ایدهٔ مطرح شده در تکه کارنمای شکل۶-۱۹ استفاده کرده‌ایم. بدین‌ترتیب که فرض می‌کنیم لیست مذکور به‌صورت غیرصعودی

مرتب است. لذا پیمایش لیست را از ابتدا تا انتها برای تعیین غیر نزولی بودن ترتیب عناصر آن، آغاز می‌کنیم. اگر در حین پیمایش دو عدد مجاور را در لیست پیدا کردیم که عدد با زیرنویس کوچک‌تر نسبت به عدد با زیرنویس بزرگتر دارای ارزش بیشتری بود، ابتدا مکان دو عدد را مذکور را تعویض کرده تا رابطهٔ ترتیب صعودی بین آنها برقرار گردد. سپس دوباره پیمایش لیست را برای تعیین غیرنزولی بودن آن از ابتدای لیست آغاز می‌کنیم.



شکل ۶- ۲۰ : کارنمای مرتب کردن ناخوش‌آیند

این الگوریتم به دو دلیل عمده ناخوش‌آیند نام گرفته است. اول به این دلیل که این الگوریتم یک الگوریتم غیر ساخت‌یافته است!؟ دوم به این دلیل که این الگوریتم برای مرتب کردن یک لیست، در بدترین حالت (که لیست از قبل به ترتیب نزولی باشد) باید $n*(n-1)/2$ بار لیستی به طول **n** را پیمایش کند و این زمانِ چندان جالبی برای یک الگوریتم مرتب‌سازی نیست. در ادامه الگوریتم‌هایی را برای مرتب‌سازی معرفی خواهیم کرد که از سرعت بالاتری نسبت به الگوریتم فوق برخوردار هستند و از همه مهمتر ساخت‌یافته نیز می‌باشند.

کار در کلاس ۶-۳ :

این دسته از مسائل مربوط است به الگوریتم مرتب کردن شکل ۶-۲۰ .

(۱) فرض کنید می خواهید با تعیین اینکه آیا لیست

۴ و ۵- و ۲ و ۷

به شکل صحیح یعنی

۷ و ۴ و ۲ و ۵-

به ترتیب صعودی ردیف می شود یا نه، الگوریتم را امتحان کنید.

a. مقادیر ورودی جعبهٔ یک چه می باشد؟

b. با استفاده از این مقادیر ورودی، پیگیری الگوریتم را از جعبهٔ ۲ شروع کنید و شمارهٔ جعبه هایی را که تا رسیدن به جعبهٔ ۷ عملاً اجرا می شوند، به ترتیب نشان دهید. از جدولی مانند آنچه در اینجا ارائه شده است استفاده کنید.

مقدار منسوب به K	۷	۶	۵	۴	۳	۲	جعبه شماره قدم
۱						√	۱

c. تعداد کل جعبه های کارنما که پس از قدم ورودی و پیش از رسیدن به جعبهٔ پایان اجرا می شوند چقدر است؟

d. چند بار به جعبهٔ ۳ می رسیم؟

(۲) تا اینجا باید کاملاً قانع شده باشید که الگوریتم در همهٔ موارد کار خود به طور صحیح انجام خواهد داد. فرض کنید مقادیری که باید مرتب شوند عبارت باشند از:

۱۲ و ۹ و ۵ و ۹-

یعنی از اول به ترتیب صعودی باشند. قبل از رسیدن به جعبه ۷، جعبه ۳ چند بار اجرا خواهد شد؟

(۳) در صورتی که مقادیر ورودی از ابتدا به ترتیب معکوس مرتب باشند مانند:

۹- و ۵ و ۹ و ۱۲

جعبهٔ ۳ چند بار اجرا می شود؟

(۴) آیا می توانید بگویید رابطهٔ $\frac{n(n-1)}{2}$ چگونه برای بدترین حالت به‌دست آمده است؟

روش مرتب‌سازی حبابی

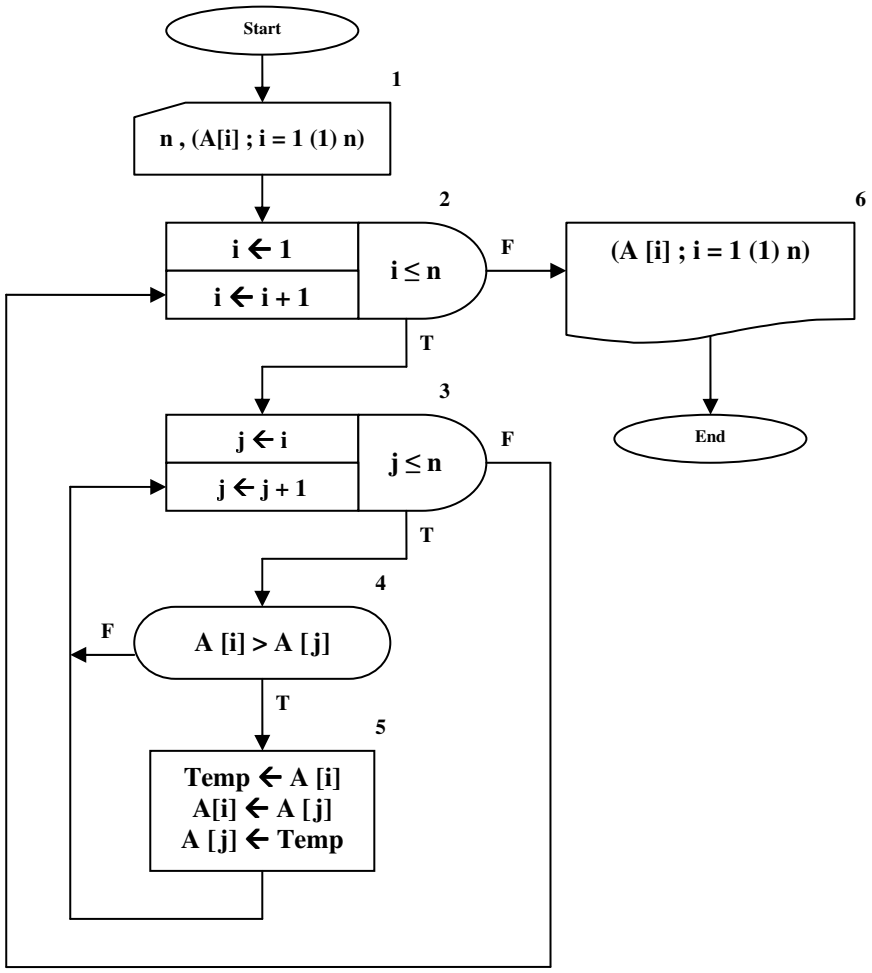
چنانکه پیشتر مطرح شد، الگوریتم شکل ۶-۲۰ الگوریتم چندان کارآمدی نیست. زیرا هر بار که یک جفت عدد را به ترتیب صعودی مرتب می‌سازیم، دوباره به ابتدای لیست باز می‌گردیم و لیست را به‌عنوان یک لیست جدید از نو مورد پردازش قرار می‌دهیم. در حالی که به خوبی می‌دانیم عناصر قبلی تا حدودی مرتب هستند. لذا در اینجا الگوریتمی را تحت‌عنوان الگوریتم مرتب‌سازی حبابی مطرح می‌سازیم که از نظر سرعت و کارآمدی تا حدودی بهتر از الگوریتم غیر ساخت‌یافتهٔ قبلی است.

پیشتر با الگوریتم یافتن بزرگترین عدد از بین N عدد در شکل ۲-۲۱ آشنا شدید. الگوریتم مرتب‌سازی حبابی ارتباط تنگاتنگی با الگوریتم مذکور دارد. اگر یک لیست مرتب را در نظر بگیرید نخستین عنصر لیست کوچکترین عنصر لیست نیز می‌باشد، و یا آخرین عنصر لیست بزرگترین عنصر لیست نیز می‌باشد. حال اگر یک لیست را به جهت یافتن کوچکترین عنصر جستجو کنیم و مکان نخستین عنصر لیست را با کوچک‌ترین عنصر لیست تعویض کنیم یک قدم به لیست مرتب شده نزدیک شده‌ایم. اگر این عمل را برای دومین عنصر لیست انجام دهیم یک قدم دیگر به لیست مرتب شده نزدیک‌ترگشته‌ایم. زیرا از دومین عنصر تا آخر لیست را به جهت یافتن کوچک‌ترین عنصر بعدی مورد جستجو قرار می‌دهیم و مکان آن عنصر را با دومین عنصر لیست تعویض می‌کنیم، بنابراین حالا دوتا از عناصر لیست مرتب در جای قطعی خود قرارگرفته‌اند. اگر این عمل را تا تعیین موقعیت آخرین عنصر لیست انجام دهیم، در نهایت می‌توان ادعا کرد که لیست مذکور مرتب شده است. همین عمل را می‌توان از انتهای لیست و با پیدا کردن بزرگترین عنصر لیست به شکل معکوس انجام داد.

برای پیاده‌سازی الگوریتم مرتب‌سازی‌حبابی به جای یافتن کوچکترین یا بزرگترین عنصر لیست و قرار دادن آن در محل موردنظر از روش دیگری برای پیاده‌سازی این ایده استفاده کرده‌ایم. در کارنمای شکل ۶-۲۱ که مربوط به مرتب سازی‌حبابی است، عنصر اول با عنصر دوم مقایسه شده در صورت بزرگتر بودن عنصر اول از عنصر دوم جای آنها تعویض می‌شود. سپس عنصراول با عنصرسوم مقایسه‌شده و در صورت بزرگتربودن عنصر اول، جای آنها با یکدیگر تعویض‌می‌شود. پس از آنکه مقایسهٔ عنصر اول با تمامی عناصر بعد از خودش پایان یافت کوچکترین عنصر به ابتدای لیست منتقل شده است. همین عمل را برای عنصر دوم با عناصر بعد از خودش انجام می‌دهیم، و این روند برای تمامی عناصر لیست تکرار می‌شود تا آنکه لیست به‌صورت مرتب درآید.

اگر بخواهیم یک تحلیل ریاضی‌وار از زمان لازم برای اجرای کارنمای شکل ۶-۲۱ داشته باشیم باید تعداد دفعات اجرای جعبهٔ شمارهٔ ۴، درون حلقهٔ داخلی (j) را برآورد کنیم. از آنجا که حلقهٔ بیرونی (n–1) بار اجرا می‌شود، حلقهٔ داخلی نیز (n–1) بار از نو اجرا می‌شود. حلقهٔ داخلی نیز، در اولین مرتبه (n–1) بار و در دومین مرتبه (n–2) بار و ... تا در آخرین مرتبهٔ که یک بار اجرا می‌گردد. در مجموع جعبهٔ شرط به تعداد دفعات زیر انجام می‌شود:

$$\sum_{i=1}^{n-1} i = \underbrace{(n-1)+(n-2)+\ldots+1}_{\text{عدد } (n-1)} = \frac{(n-1)(n-2)}{2}$$



شکل ۶- ۲۱ : کارنمای مرتب‌سازی به روش حبابی

کار در کلاس ۶-۴ :

کارنمای شکل ۶- ۲۱ را به ازای مقادیر زیر آزمایش کنید.

۹۸ و ۱۲ و ۸- و ۱۲۷ و ۴۳ و۱۵ و ۷۴ و ۹۶ و ۴۰ و ۱۰

کار در کلاس ۶-۵:

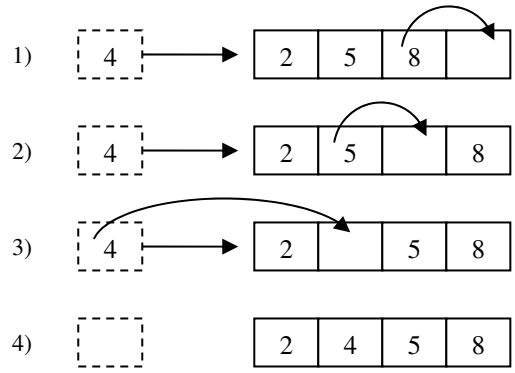
أ- کارنمایی رسم کنید که مرتب‌سازی حبابی را به نخستین شیوهٔ مطرح شده در صفحات قبل، یعنی یافتن کوچکترین عنصر لیست و قرار دادن آن در ابتدای آرایه انجام دهد.

ب- زمان لازم برای اجرای الگوریتم خود را مطابق آنچه که دیدید تخمین بزنید.

ت- سعی کنید با استفاده از یک ابتکار زمان لازم برای اجرای الگوریتم را به نصف کاهش دهید.(راهنمایی: برروی مرتب‌سازی لیست از هر دو سمت فکر کنید).

مرتب سازی درجی

روش مرتب‌سازی درجی بدین‌گونه است که هنگام خواندن داده‌ها به داخل حافظه، آنها را مرتب می‌کنیم. با توجه به آن که اصولاً در عملیات ورودی و خروجی مقداری زمان جهت تبادل داده‌ها بین ترمینال‌های سخت‌افزاری کامپیوتر به هدر می‌رود، می‌توان عملیات مرتب‌سازی درجی را برای لیست‌های کوچک در این زمان مرده انجام داد. مرتب‌سازی‌درجی به اینگونه عمل‌می‌کند که عناصر لیست را تک تک از ورودی می‌خواند و همزمان در مکان مناسبی از لیست قرارمی‌دهد. جهت ایجاد فضای مناسب در بین عناصر لیست برای قرار دادن عنصر مورد نظر در مکان مناسب خود؛ عناصر لیست را از انتهای لیست تا مکان موردنظر، یکی یکی، یک خانه به سمت راست حرکت (شیفت) می‌دهیم. در آن صورت، فضای لازم در بین عناصر آرایه ایجـادمی‌شود. به‌عـنوان مثال فرض‌کنید می‌خواهیم عدد ۴ را در آرایهٔ زیر قرار دهیم. در این صورت ابتدا باید اعداد ۵ و ۸ را یک خانه به سمت راست شیفت دهیم. سپس عنصر ۴ را در فضای ایجاد شده قرار می‌دهیم. به اشکال رسم شده در تصویر ۶-۳۱ توجه کنید.

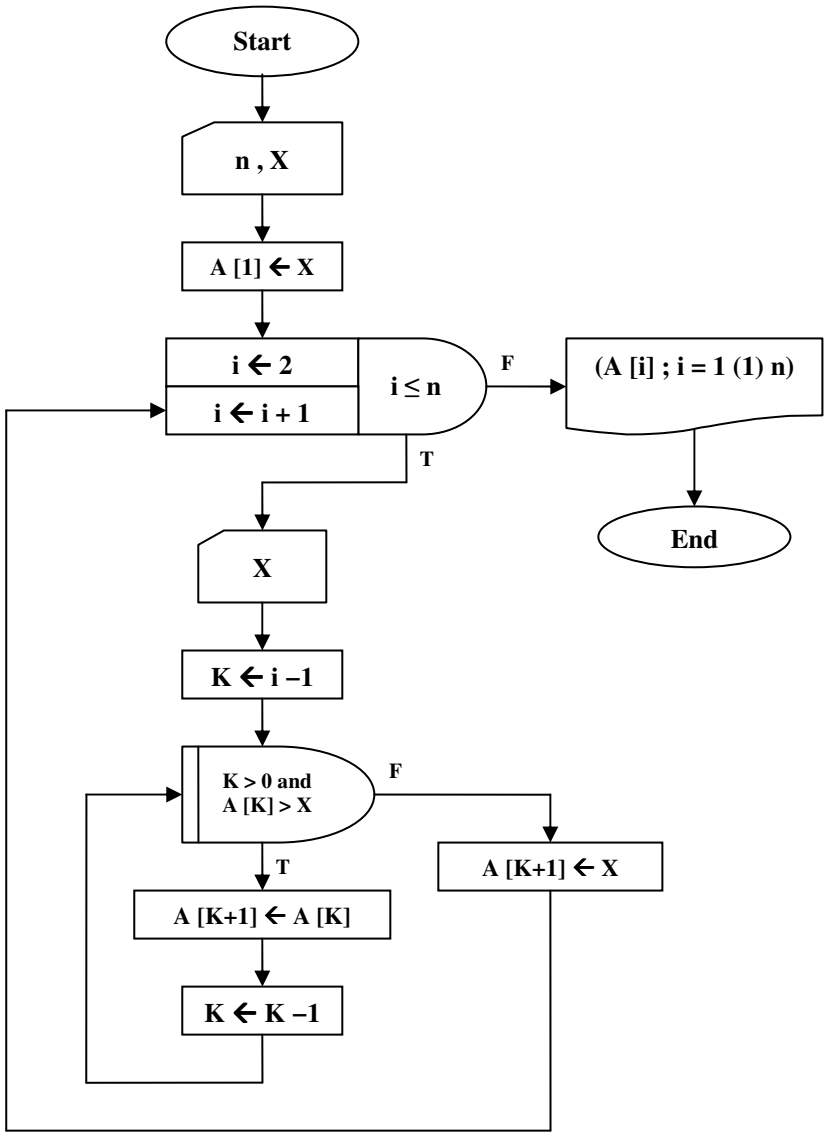


شکل ۶- ۲۲ : روند مرتب‌سازی درجی

به‌کار بردن این روش برای لیست‌های بلند توصیه نمی‌شود، چرا که در آن صورت به دلیل عملیات وقت‌گیر شیفت دادن عناصر، زمان مرتب‌سازی بیش از حد انتظار افزایش می‌یابد. در شکل ۶-۲۲ کارنمای مرتب‌سازی درجی نمایش داده شده است.

کار در کلاس ۶-۶:

کارنمای شکل۶-۲۱ چه تغییری باید بکند تا پس از خواندن کامل یک لیست، آن را در لیستی دیگر به روش درجی مرتب‌سازی کند.



شکل ۶- ۲۳ : کارنمای مرتب‌سازی درجی

از نظر تحلیل زمانی این الگوریتم دقیقاً مشابه الگوریتم مرتب‌سازی حبابی می‌باشد. آیا می‌توانید مراحل تحلیل را به‌طور دقیق در خصوص این کارنما شرح دهید؟

جستجوی دودویی

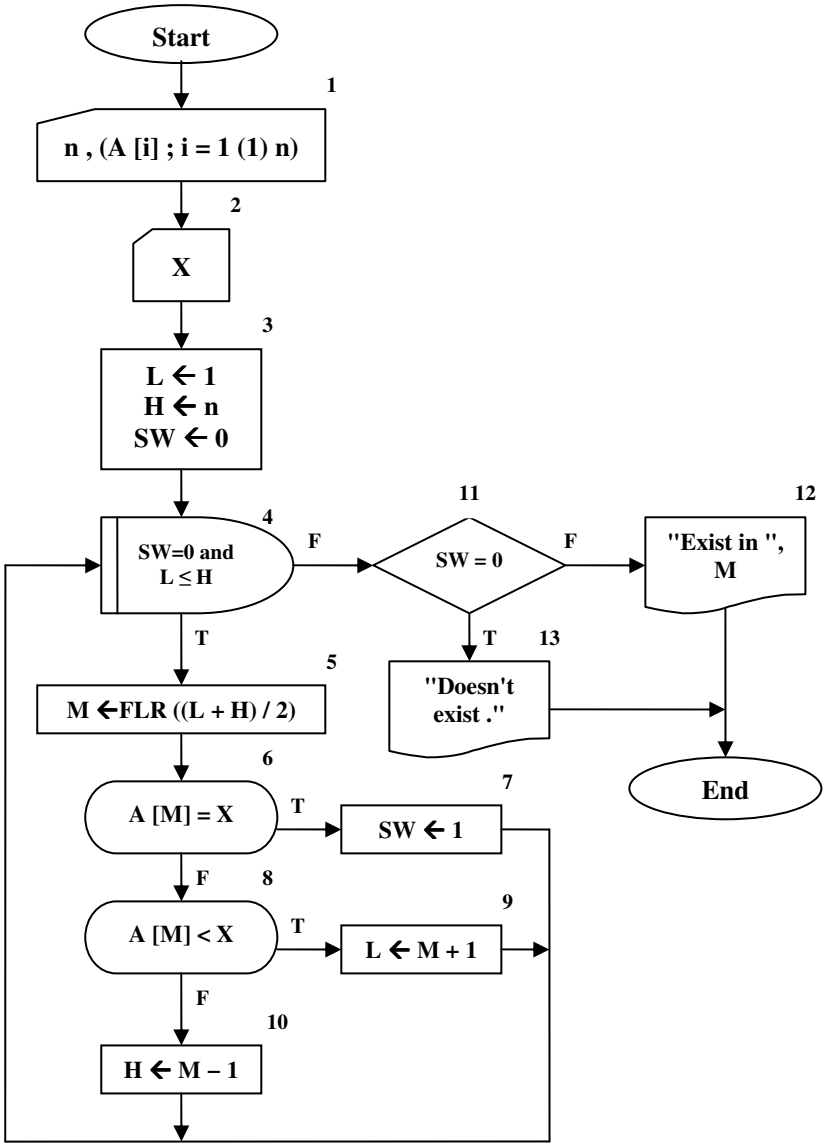
برای آنکه درک خوبی از جستجوی دودویی پیدا کنید این سؤال را مطرح می‌سازیم که، برای یافتن نام یک نفر از دوستان خود در یک دفترچه تلفن چگونه عمل می‌کنید؟

برای پاسخ به این سؤال فرض کنید یک دفترچه تلفن که براساس ۳۲ حرف الفبای فارسی یعنی از «الف» تا «ی» صفحه‌بندی شده است در اختیار داریم. در ابتدا دفترچه تلفن را از وسط آن باز می‌کنیم. یعنی مرز بین دو حرف «ش» و «ص» . حال نگاه می‌کنیم حرف اول نام خانوادگی فرد مورد نظر در کدام نیمهٔ دفترچه تلفن قرار می‌گیرد. مثلاً فرض کنید حرف مذکور در نیمهٔ بالایی قرار گیرد، به راحتی نیمهٔ پایینی را کنار گذاشته و عمل نصف کردن را برای نیمهٔ بالایی دفترچه تلفن ادامه می‌دهیم. اگر این عمل را حداکثر ۵ مرتبه انجام دهیم در نهایت به انتخاب یک حرف از بین دو حرف می‌رسیم، و به این ترتیب در نهایت به نام فرد مورد نظر خود می‌رسیم.

جستجوی دودویی بر مبنای همین روش نصف کردن بنا شده است. برای اجرای عمل جستجوی دودویی حتماً باید آرایهٔ مربوطه از قبل مرتب شده باشد. در این روش یک آرایهٔ مرتب را مدام نصف می‌کنیم و با تشخیص اینکه نمونهٔ مورد نظر ما، در کدام نیمه از آرایه قرار دارد، نیمهٔ دیگر را کنار گذاشته و عمل نصف کردن را برای نیمهٔ باقی‌مانده ادامه می‌دهیم.

اگر می‌خواهید به ارزش جستجوی دودویی پی‌ببرید فرض کنید لیستی داریم که شامل $۲^{۲۰}$ نمونه است. برای انجام یک جستجوی ترتیبی در این لیست حداکثر باید ۱.۰۴۸.۵۷۶ عمل مقایسه(پیگرد) انجام داد. در حالی که برای انجام همان عمل جستجو به شیوهٔ دودویی در این لیست، حداکثر به ۲۰ عمل مقایسه نیازمندیم. به‌طورکلی برای صورت دادن یک عمل جستجوی دودویی در یک لیست به طول **n** ، نیازمند $\mathbf{Log_2^n}$ پیگرد هستیم.

در شکل۶-۳۳ کارنمایی ترسیم شده که چگونگی اجرای یک جستجوی دودویی را بر روی یک لیست **n** عنصری نشان می‌دهد. این کارنما کمی با روش مطرح شده در خصوص دفترچه تلفن متفاوت است. در این کارنما متغیر **X** عدد مورد جستجو را در خود ذخیره می‌کند. متغیرهای **L** و **H** اندیس ابتدا و انتهای محدودهٔ مورد پردازش از لیست را در خود نگه می‌دارند. متغیر **SW** یک متغیر وضعیت(سوئیچ) است. از این متغیر جهت کنترل حلقهٔ تکرار استفاده شده است. از آنجا که عمل جستجو باید تا زمان یافتن نخستین نمونه ادامه یابد متغیر **SW** را به مقدار صفر مقدار اولیه داده‌ایم، در صورت یافته شدن نمونهٔ مورد نظر این متغیر به مقدار یک تغییر وضعیت می دهد، تا از اجرای دوبارهٔ حلقهٔ تکرار جلوگیری کند. شرط **L≤H** تا زمانی برقرار است که عناصر مورد پردازش در لیست بیش از یک مورد باشد در جعبهٔ شمارهٔ ۵ اندیس مربوط به عنصر میانی در محدودهٔ مورد پردازش در داخل متغیر **M** قرار می‌گیرد. در جعبهٔ شمارهٔ ۶ چک می‌کنیم عنصر میانی محدودهٔ مورد پردازش برابر نمونهٔ مورد جستجو هست یا نه؟ اگر عنصر موردنظر یافت شود در جعبهٔ ۷ مقدار **SW** را به مقدار یک تغییر داده و از حلقه خارج می‌شویم. در غیر این صورت در جعبه‌های ۸ و ۹ و ۱۰ با مقایسهٔ نمونهٔ مورد جستجو با عنصر میان محدودهٔ در حال پردازش نیمهٔ پایینی و یا نیمهٔ بالایی محدوده را کنار می‌گذاریم. این کار به واسطهٔ مقداردهی جدید به یکی از متغیرهای **L** یا **H** میسر می‌شود. این روند تا هنگامی که نمونهٔ مورد نظر در آرایه یافت شود و یا تمامی آرایه مورد پردازش قرار گیرد و نمونهٔ موردنظر در آرایه وجود نداشته باشد ادامه پیدا می‌کند. در جعبه‌های ۱۱ و ۱۲ و ۱۳ بر حسب مقدار **SW** پیام مناسب چاپ می‌گردد.



شکل ۶- ۲۴ : کارنمای جستجوی دودویی

۶-۶ : تمرین

۱. فرض کنید مجموعهٔ داده‌هایی که در رابطه با کارنمای پایین همین صفحه مورد استفاده قرار می‌گیرد متشکل از مقادیر زیر باشد.

۸ و ۷ و ۶ و ۵ و ۴ و ۳ و ۲ و ۱

الف- پس از اولین اجرای جعبهٔ ۳ مقادیر لیست‌های A و B عبارت‌اند از (یکی را انتخاب کنید):

(1) A : ۱, ۲,۳,٤ B : ٥,٦,۷,۸

(2) A : ۱, ۲,۳,٥ B : ٤,٦,۷,۸

(3) A : ٥,۲,۳,٤ B : ۱, ٦,۷,۸

(4) A : ۸,۲,۳,٤ B : ٥,٦,۷,۱

(5) A : ۸,۲,۳,٥ B : ۱,٦,۷,٤

ب- مقادیری که به خارج داده می‌شود (به ترتیب) عبارت خواهند بود از(یکی را انتخاب کنید):

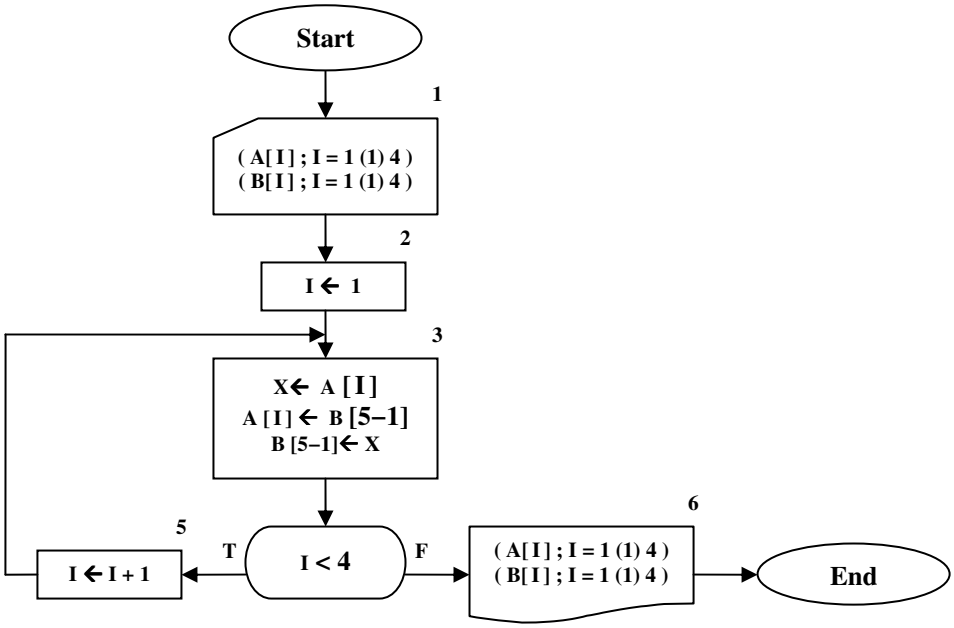
(1) ۸,۷,٦,٥,٤,۳,۲,۱

(2) ۱,۲,۳,٤,٥,٦,۷,۸

(3) ٥,٦,۷,۸,۱,۲,۳,٤

(4) ٦,٥,۳,٤,۱,۲,۷,۸

(5) هیچکدام



شکل ۶- ۲۵ : کارنمای تمرین ۱

۲. فرض کنید داده‌های ورودی کارنمای شکل ۶-۲۶ از مقادیر زیر تشکیل شده باشد.

۸ و ۷ و ۶ و ۵ و ۴ و ۳ و ۲ و ۱

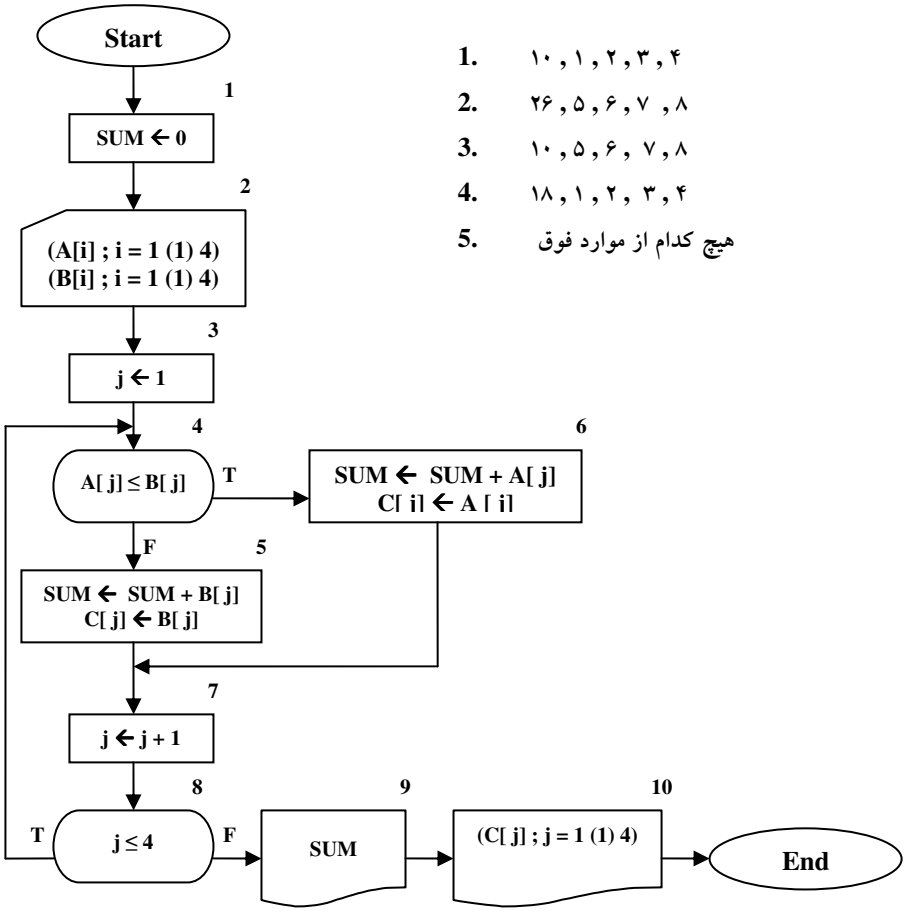
الف- درست یا نادرست: پس از اولین اجرای جعبهٔ ۷ مقادیر فهرست **C** به جز **C[1]** نامعین هستند.

ب- درست یا نادرست : پس از آخرین اجرای جعبهٔ ۷ مقادیر عناصر لیست **B** عبارتند از:

۸ و ۷ و ۶ و ۵

ج- در اجرای جعبه‌های ۹ و ۱۰ مقادیری که به خارج داده می‌شود به ترتیب به قرار زیرند(یکی را انتخاب کنید):

1. ۱۰ , ۱ , ۲ , ۳ , ۴
2. ۲۶ , ۵ , ۶ , ۷ , ۸
3. ۱۰ , ۵ , ۶ , ۷ , ۸
4. ۱۸ , ۱ , ۲ , ۳ , ۴
5. هیچ کدام از موارد فوق



شکل ۶- ۲۶ : کارنمای تمرین ۲

۳. یک ارکستر سمفونی جوانان، که عازم مسافرتی برای اجرای کنسرت است، دارای صد و یک عضو است. خبرنگاری از رهبر ارکستر سؤال می‌کند «میانهٔ سن اعضای شما چه قدر است؟» رهبر ارکستر جواب می‌دهد، «نمی‌دانم، ولی این، فهرست الفبایی اسامی و سنین نوازندگان است.»

الف- تعریفی که در این مسئله برای میانهٔ یک مجموعهٔ مرتب از اعداد می‌کنیم این است که عنصر «میانی» مجموعه باشد. به عبارت دیگر قبل از آن، به همان تعداد عنصر وجود داشته باشد که بعد از آن وجود دارد. برای مثال، در مجموعهٔ زیر میانه ۳۵ است.

۹۷ و ۸۱ و ۶۷ و ۳۵ و ۲۴ و ۷ و۳

کارنمایی رسم کنید که میانهٔ سن نوازندگان را از فهرست مذکور، که بنا به فرض به ترتیب عددی نیست، پیدا کند.

ب- در مواردی که تعداد اعداد مورد نظر زوج باشد، تعریف فوق‌الذکر برای میانه صادق نیست. مثلاً دنبالهٔ زیر دارای عنصر «میانی» نیست.

۶۸ و ۵۷ و ۴۳ و ۳۱ و ۱۷ و ۴

در این صورت می‌توان میانه را به‌صورت میانگین دو عدد میانی تعریف کرد. در دنبالهٔ فوق میانه عبارت است از: ۳۷ = ۲ ÷ (۴۳ + ۳۱).

راهنمایی: برای تعیین میانهٔ مجموعهٔ شامل **n** عدد a_1 ، a_2 ، ... ، a_n عبارتی تهیه کنید. پاسخ شما باید چنان عبارتی باشد که جواب صحیح را برای هر **n** ، اعم از زوج یا فرد، به‌دست دهد.

کارنمایی شبیه کارنمای بند (الف)، ولی طرح شده برای ارکستری شامل تعداد دلخواهی نوازنده، چنان تهیه کنید که نه فقط میانهٔ سن بلکه سن جوانترین و پیرترین نوازنده را نیز پیدا کند.

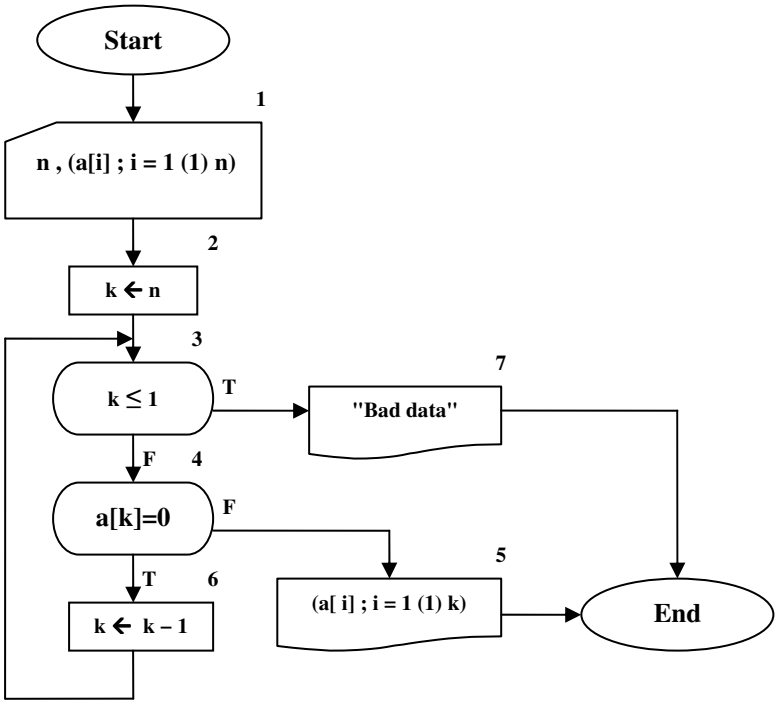
۴. کارنمای شکل ۶-۲۷ را مطالعه کنید و بند (الف) و (ب) را پاسخ دهید.

الف- فرض کنید مجموعهٔ داده‌هایی که در اثر اجرای جعبهٔ ۱ کارنما به داخل آورده می‌شود عبارت باشد از:

۱۶/۷ و ۴/۴ و ۰ و ۴/۱- و ۲/۱ و ۱۹ و ۶

هنگامی‌که اجرای کارنما به جعبهٔ تعریف می‌رسد، مقدار منسوب متغیر **K** کدام یک از مقادیر زیر خواهد بود.

۱. ۰
۲. ۶
۳. ۱-
۴. ۴
۵. ۵



شکل ۶- ۲۷ : کارنمای تمرین ۴

ب- اگر مجموعهٔ داده‌هایی که در نتیجهٔ اجرای جعبهٔ ۱ به داخل آورده می‌شود عبارت باشد از:

۰ و ۰ و۴ و ۰ و ۴ و ۵

خروجی‌ای که به وسیله الگوریتم تولید می‌شود، توسط کدام‌یک از جواب‌های زیر نشان داده شده؟

۱. ۴ ، ۰ ، ۴

۲. «دادهٔ خراب»

۳. ۰ ، ۰ ، ۴ ، ۰ ، ۴ ، ۳

۴. ۴ ، ۰ ، ۳ ، ۳

۵. ۴ ، ۰ ، ۴ ، ۲

۵. فرض کنید کارنمای شکل ۶-۲۸ با استفاده از مجموعهٔ داده‌های زیر اجرا شود.

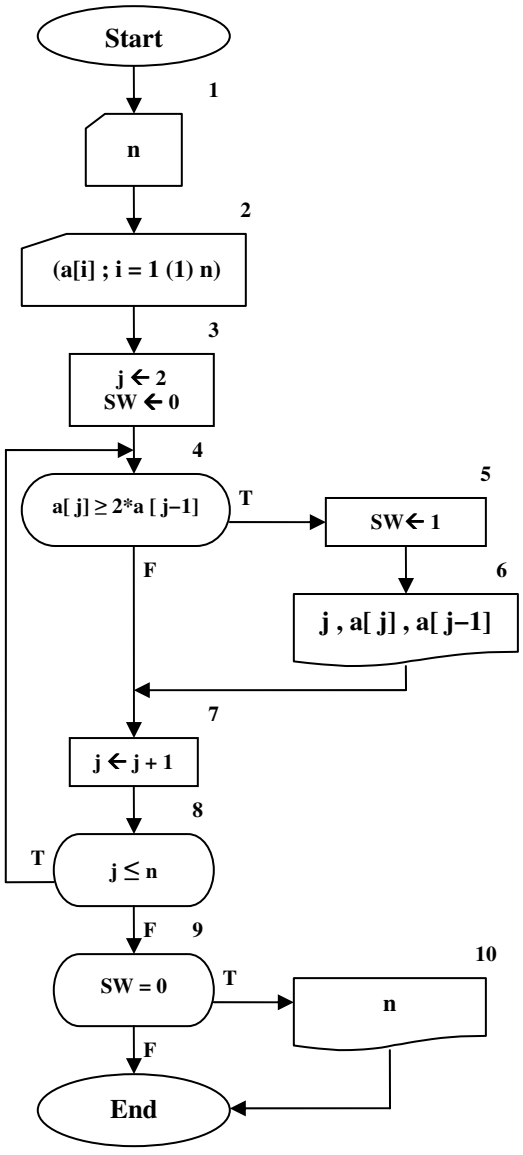
۹ و ۳ و ۷ و ۲۲ و ۱۰ و ۱۲ و ۶ و ۳ و ۸

الف- جعبهٔ ۸ چند بار اجرا خواهد شد؟

ب- جعبهٔ ۵ چند بار اجرا خواهد شد؟

ج- کلیه مقادیری را که چاپ خواهد شد نشان بدهید. فرض کنید هر بار اجرای جعبهٔ ۶ یا جعبهٔ ۱۰ ، موجب عمل چاپ برروی خط جدیدی می‌شود (وقتی جعبهٔ ۶ اجرا شود سه مقدار و وقتی جعبهٔ ۱۰ اجرا شود یک مقدار برروی خط چاپ خواهد شد).

د- آیا استفاده از زیرنویس واقعاً برای مسئله ضروری است؟ اگر جوابتان منفی است، به‌منظور حذف زیرنویس‌ها، کارنما را مجدّداً ترسیم کنید و در کنار هر جعبه‌ای که تغییر می‌دهید علامت تیک (√) قرار دهید.



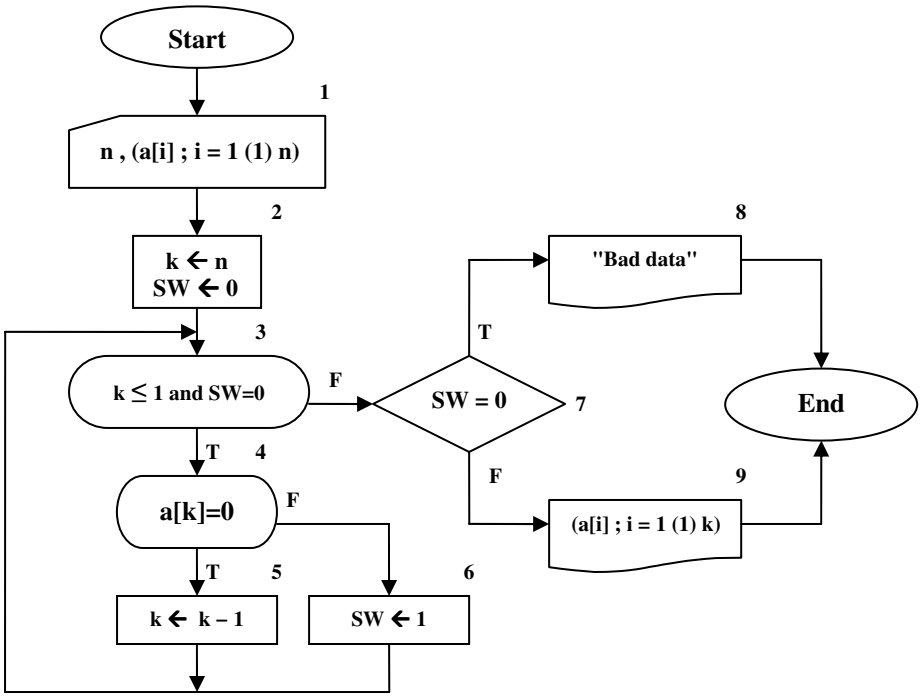
شکل ۶- ۲۸ : کارنمای تمرین ۵

۶. آیا کارنمای شکل ۶-۲۸ ساخت یافته است؟ اگر جواب شما منفی است آن را دوباره به‌صورت ساخت یافته ترسیم کنید و اگر جواب شما مثبت است دلیل خود را شرح دهید.

۶-۷ : کاربرد متغیرهای وضعیت(سوئیچ)

در برخی مسائل مطرح شده تا اینجا با متغیرهای سوئیچ آشنا شدید و برخی از کاربردهای آنها را دیدید. اصولاً در الگوریتم‌ها از متغیرهای وضعیت به نام سوئیچ می‌توان استفاده کرد تا تصمیم‌گیری در دستورات شرطی و حلقه‌های تکرار آسان بشود. به‌عنوان مثال پیشتر در کارنمای شکل ۶-۲۴ با متغیر وضعیت SW برخورد کردید. از این متغیر جهت کنترل حلقهٔ تکرار و تصمیم‌گیری استفاده کردیم. لذا از ارائهٔ مثال تکراری در خصوص این کاربرد متغیرهای سوئیچ خودداری می‌کنیم و به بررسی کاربرد مهمتر این متغیرها می‌پردازیم.

یکبار دیگر به کارنمای شکل ۶-۲۷ توجه کنید. فارق از این که نحوهٔ تعریف آرایه در زبان C++ چگونه است، اگر به شما بگویند برنامهٔ C++ مربوط به این کارنما را با معلومات فعلی خود بنویسید قطعاً به مشکل برخواهید خورد[۱]. علت این امر، رفتن به سوی دکمهٔ پایان از دو محل متفاوت در کارنما یعنی از جعبه‌های ۵ و ۷ است، که هر کدام مربوط به یک جعبهٔ تصمیم جداگانه هستند. برای رفع این مشکل از متغیرهای وضعیت باید استفاده کرد. نحوهٔ استفاده از این متغیرها در کارنمای شکل ۶-۲۹ که بازنویسی کارنمای شکل ۶-۲۷ است نشان داده شده است. البته راه حل زیر را می‌توان به نوعی حرکت به سوی ساخت‌یافتگی بیشتر، در برنامه‌ها دانست.



شکل ۶- ۲۹ : کارنمای بازنویسی شدهٔ با متغیر سوئیچ

۱. البته همانگونه که مطرح شد معلومات فعلی شما کم است، وگرنه می‌توان برنامهٔ معادل کارنمای شکل ٦-٢٤ را با استفاده از تابع کتابخانه‌ای exit نوشت. اما در همهٔ زبان‌های برنامه‌نویسی چنین امکانی وجود ندارد.

۶-۸ : آرایه‌های دو بعدی

در ریاضیات دورهٔ متوسطه با ماتریس‌ها آشنا شده‌اید. با عملیات مختلف جمع، تفریق، ضرب، دترمینان و معکوس‌گیریِ ماتریس‌ها به‌خوبی آشنایی دارید و برخی کاربردهای آنها را دیده‌اید. یکی از مهمترین کاربردهای ماتریس‌ها یعنی حل دستگاه‌های چند معادله و چند مجهول را تجربه کرده‌اید. تمامی این عملیات از نظر علم کامپیوتر بسیار با اهمیت و پرکاربرد هستند. بسیاری از معادلات پیچیده در مدارات الکترونیکی در نهایت منجر به یک دستگاه معادلات می‌شود که حل آنها با دست بسیار وقت‌گیر و شاید نا ممکن است. لذا کارکردن بر روی ماتریس‌ها و عملیات مختلف برروی آنها بسیار حائز اهمیت می‌باشد. برای پیاده‌سازی ماتریس‌ها در حافظهٔ کامپیوتر از آرایه‌های دو بعدی بهره می‌گیریم. فرض کنید ماتریس $A_{3\times4}$ که ماتریس ضرایب دستگاه معادلات زیر است، را بخواهیم در حافظهٔ کامپیوتر نشان دهیم. برای این منظور از یک آرایهٔ دو بعدی که علاوه بر ستون دارای تعدادی سطر نیز هست استفاده می‌کنیم، که در شکل ۶-۳۱ نشان داده شده است.

$$\begin{cases} 2X + 7Y + 63Z = 13 \\ 45X + Z = 91 \\ 6X + 29Y = 4 \end{cases} \Longrightarrow \begin{bmatrix} 2 & 7 & 63 & 13 \\ 45 & 0 & 1 & 91 \\ 6 & 29 & 0 & 4 \end{bmatrix}$$

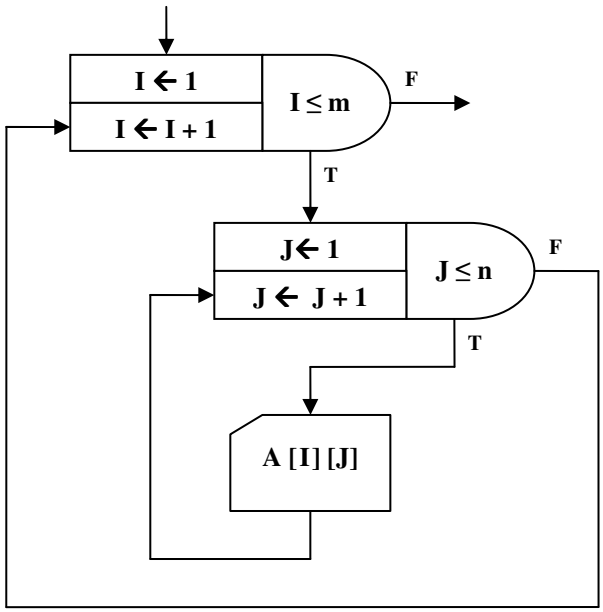
شکل ۶- ۳۰ : یک دستگاه معادلات و ماتریس ضرایب آن

2	7	63	13
45	0	1	91
6	29	0	4

شکل ۶- ۳۱ : آرایهٔ دو بعدی معادل ماتریس فوق

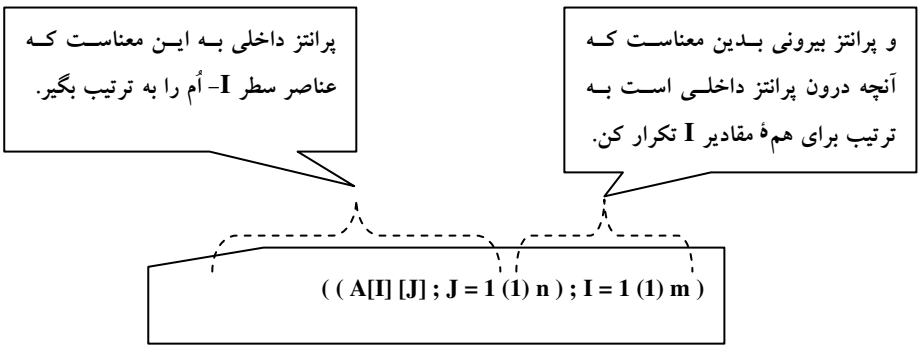
یک آرایهٔ دو بعدی را می‌توان به‌عنوان یک جدول تصور کرد با این توصیف کاربرد آرایه‌های دو بعدی بسیار فراتر از ماتریس‌ها خواهد رفت. به‌عنوان مثال می‌توان از آرایه‌ها به‌عنوان جداول ریز قیمت اجناس یک فروشگاه استفاده کرد. که هر ردیف نام یک کالا و قیمت آن را نگه دارد، و یا آن که می‌توان جدول میهمانان یک هتل را به همراه شمارهٔ اتاق آنها در یک آرایهٔ دو بعدی ذخیره کرد که در ستون اول هر ردیف نام میهمان و در ستون دوم، شمارهٔ اتاق آن را ذخیره کرد. رفته رفته کاربردهای متنوع‌تری از آرایه‌های دو بعدی را خواهید دید.

چنانکه در ریاضیات برای مراجعه به درایهٔ واقع در سطر I- اُم و ستون J- اُم می‌نوشتیم $A_{I,J}$ در اینجا به‌طور معادل می‌نویسیم A[I][J] . نکتهٔ قابل توجه آنکه ابتدا شمارهٔ سطر و سپس شمارهٔ ستون ذکر می‌شود. از طرفی برای خواندن اطلاعات یک ماتریس باید از دو حلقهٔ تکرار به‌صورت زیر استفاده کنیم:



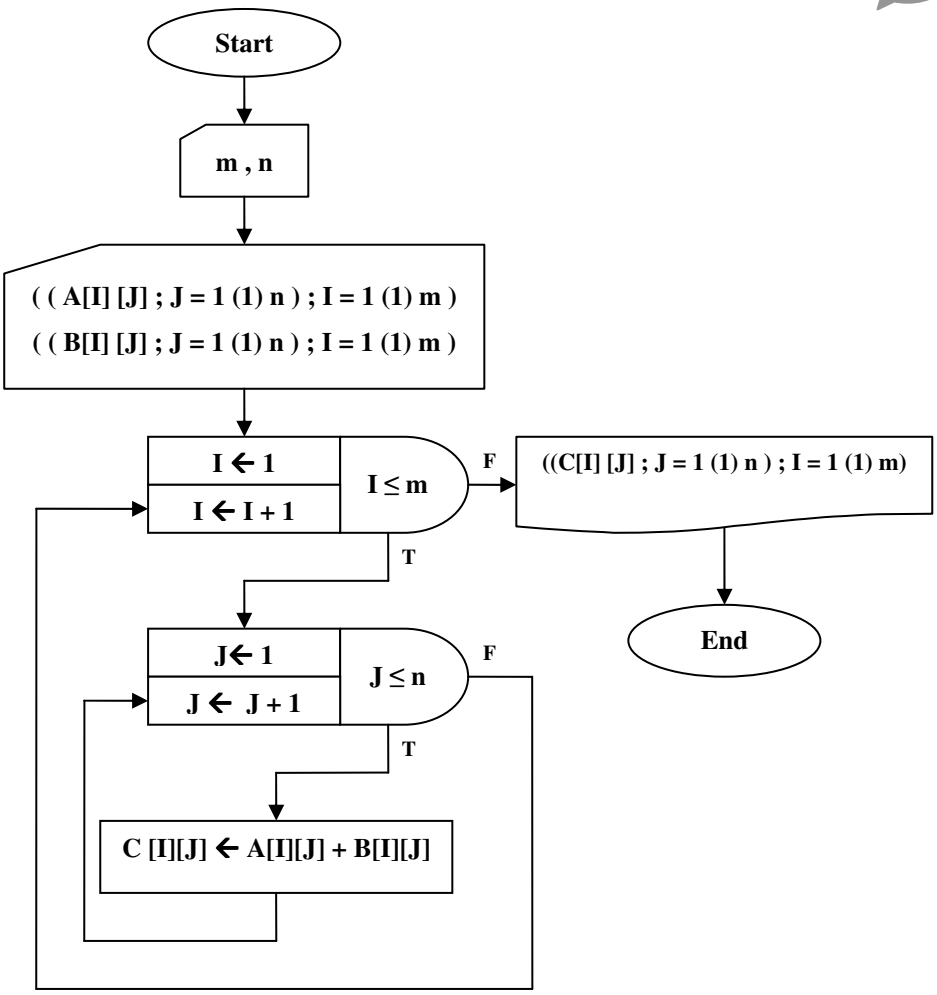
شکل ۶- ۳۲ : نحوهٔ خواندن درایه‌های آرایهٔ دو بعدی

اما چنانکه برای خواندن و نوشتن برروی تمامی درایه‌های آرایهٔ یک بعدی نماد خاصی را قرارداد کردیم برای خواندن و یا نوشتن تمامی درایه‌های یک آرایهٔ دوبعدی به روش زیرعمل می‌کنیم. به نحوهٔ پرانتزگذاری‌ها توجه ویژه داشته باشید.



شکل ۶- ۳۳ : قرارداد جدید خواندن درایه‌های آرایهٔ دو بعدی

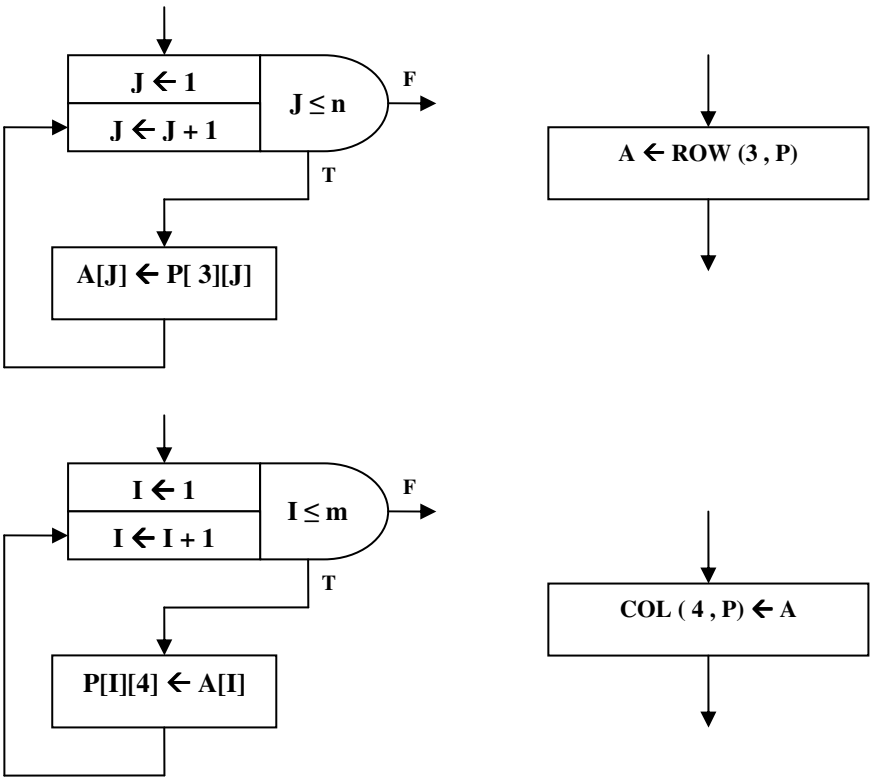
مثال ۶-۴ : کارنمایی که دو ماتریس هم اندازه با ابعاد m×n را با هم جمع می‌کند.



شکل ۶- ۳۴ : کارنمای مثال ۶-۴

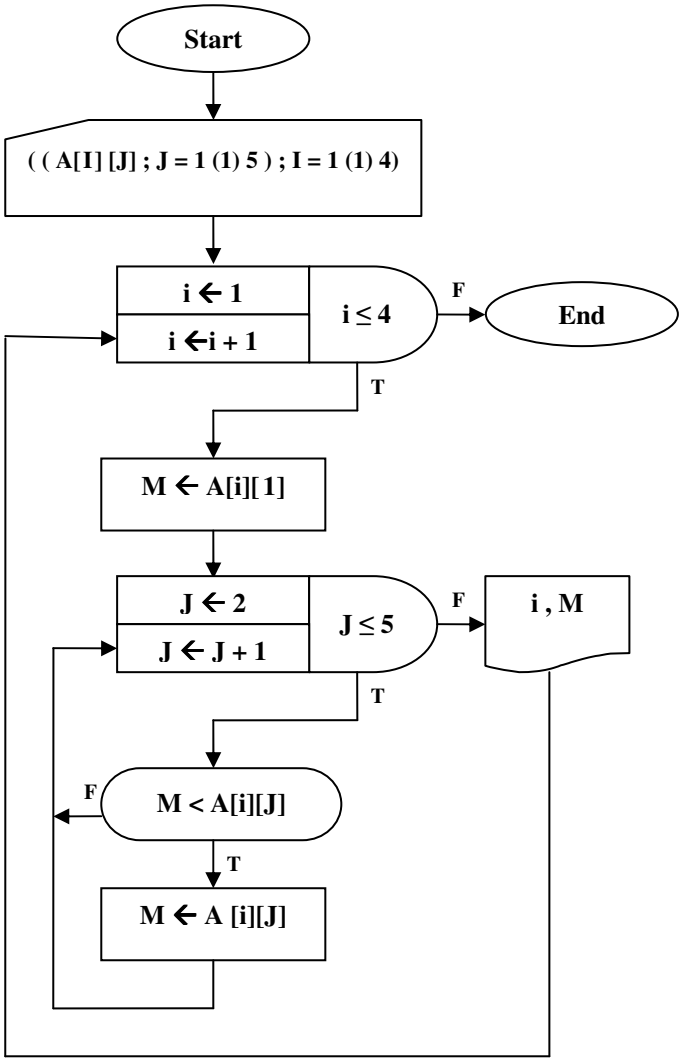
توضیح مثال : جمع دو ماتریس تنها با انجام جمع جبری بر روی درایه‌های نظیر به نظیر ماتریس محقق می‌گردد. لذا پس از خواندن درایه‌های هر دو ماتریس A و B ، در یک حلقهٔ تودرتو شروع به جمع درایه های نظیربه‌نظیر دو ماتریس کرده‌ایم.

اما به‌عنوان آخرین نکته در زمینهٔ کار با آرایه‌ها این که گاهی اوقات می‌خواهیم درایه‌های یک سطر از یک ماتریس دو بعدی را در داخل یک آرایهٔ یک بعدی بریزیم، و یا بر عکس ممکن است بخواهیم درایه‌های یک آرایهٔ یک بعدی را دریک سطر از یک ماتریس دوبعدی قراردهیم. در اینگونه موارد به جای استفاده از حلقه‌های تکرار از نمادهای مندرج در شکل۶-۴۶ استفاده می‌کنیم. به چگونگی کاربرد دو کلمهٔ **ROW** و **COL** به دقت توجه کنید.



شکل ۶- ۳۵ : نماد انتساب تک ردیفی یا تک ستونی

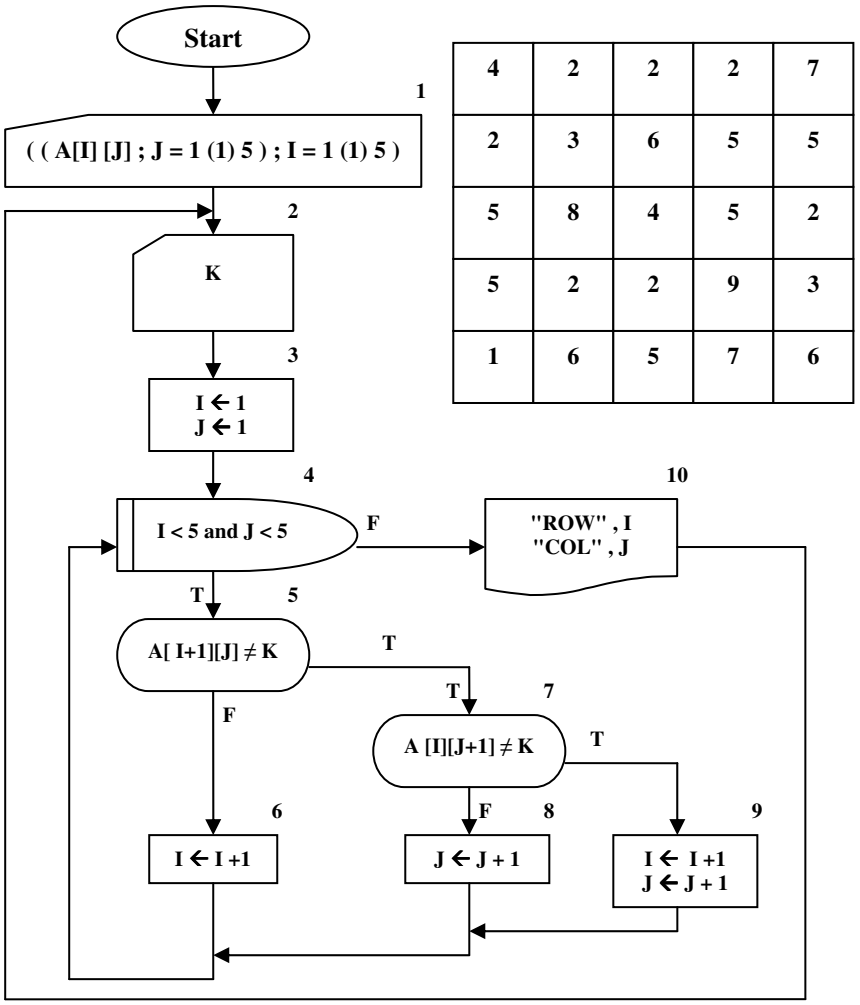
مثال ۶-۵ : الگوریتمی که عناصر یک آرایهٔ ۵×۴ را خوانده و بزرگترین عنصر هر سطر را چاپ می‌کند.



شکل ۶- ۳۶ : کارنمای مثال ۶-۵

۶-۹ : تمرین

۱. فرض کنید با اجرای جعبهٔ ۱ در کارنمای زیر مقادیر ماتریس زیر به داخل **A** وارد شود:



4	2	2	2	7
2	3	6	5	5
5	8	4	5	2
5	2	2	9	3
1	6	5	7	6

شکل ۶- ۳۷ :کارنمای تمرین ۱

الف- فرض کنید اولین اجرای جعبهٔ ۲، مقدار ۲ را به **k** منسوب کند. در این صورت نتیجه‌ای که در جعبهٔ ۱۰ برای این مقدار از **k** چاپ می‌شود مطابق زیر خواهد بود.

۱. **ROW 5 COL 3**
۲. **ROW 2 COL 5**
۳. **ROW 3 COL 5**
۴. **ROW 5 COL 4**
۵. هیچکدام از موارد فوق.

ب- فرض کنید دومین اجرای جعبهٔ ۲ مقدار ۵ را به K منسوب کند. در این صورت نتیجه‌ای که در جعبهٔ ۱۰ برای این مقدار از K چاپ می‌شود مطابق زیر خواهد بود.

۱. ROW 5 COL 3
۲. ROW 5 COL 2
۳. ROW 4 COL 5
۴. ROW 2 COL 5
۵. هیچکدام از موارد فوق.

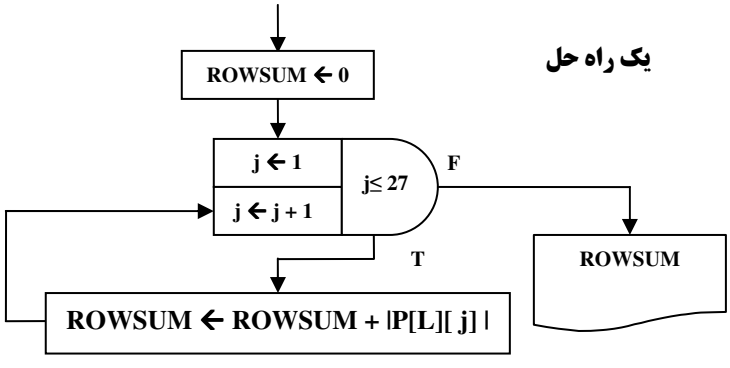
۲. با استفاده از آرایهٔ ۵ سطری و ۲۵ ستونی A از رشته های تک نویسه‌ای، کارنمایی جهت چاپ حروف درشت MJZ، مطابق آنچه در زیر نشان داده‌شده، ترسیم‌کنید. این حروف درشت با فاصله‌ای با عرض ۵ نویسهٔ خالی از یکدیگر جدا شده‌اند و مقادیر ورودی منحصراً شامل نویسه‌های «M » ، «J » ، «Z» هستند.

```
M       M     J J J J J     Z Z Z Z Z
M M   M M         J             Z
M   M   M         J           Z
M       M     J   J         Z
M       M       J       Z Z Z Z Z
```

در مسائل ۳ الی ۸ که در ادامه آمده، فرض کنید که به همهٔ متغیرها یا درایه‌های ماتریس‌هایی که ذکر شده‌اند قبلاً مقادیر اولیه‌ای منسوب شده است. وظیفهٔ شما ترسیم کارنمای عملی است که توضیح داده شده است. (اینها عملیات ابتدایی هستند که غالباً برروی ماتریس‌ها انجام می‌شوند و معمولاً قسمت‌هایی از مسائل بزرگتراند.)

مثال: ماتریسP دارای ۲۲ سطر و ۲۷ ستون است. مجموع قدرمطلق‌های همهٔ درایه‌های سطر L این ماتریس را پیدا کنید. قبلاً مقدار اولیه‌ای به L منسوب شده است.

یک راه حل



شکل ۶- ۳۸ : کارنمایی برای مثال فوق

۳. برای همان ماتریس**P** ، مجموع همهٔ درایه‌های ستون **K**- اُم را به استثنای یک درایه پیدا کنید. درایه‌ای که مستثنی می‌شود درایهٔ سطر ۱۲ است. مجموعی را که بدین وسیله به‌دست می‌آید **COLSUM** بنامید.

۴. برای همان ماتریس**P** ، به هر درایه از ردیف **L** مقدار درایهٔ نظیر آن (در همان ستون) از ردیف **M** را بیفزایید. به‌عنوان یک مثال واقعی و با یک ماتریس به مراتب کوچک‌تر **Q** خواهیم داشت:

ماتریس فرضی Q قبل از عملیات

$$\begin{bmatrix} 3 & 4 & 2 & 5 & -4 \\ 3 & 9 & 1 & 2 & -4 \\ 1 & 1 & 2 & 3 & 2 \end{bmatrix}$$

سطر L-اُم

سطر M-اُم

ماتریس فرضی Q بعد از عملیات

$$\begin{bmatrix} 3 & 4 & 2 & 5 & -4 \\ 4 & 10 & 3 & 5 & -2 \\ 1 & 1 & 2 & 3 & 2 \end{bmatrix}$$

سطر L-اُم

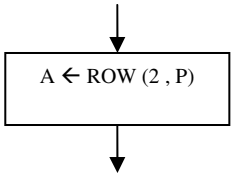
سطر M-اُم

۵. برای همان ماتریس**P** ، به هر درایه از سطر **L** ، به استثنای درایه **K**-اُم آن ، دو برابر درایهٔ نظیر آن در سطر **M**-اُم را بیفزایید.

۶. برا ی همان ماتریس**P** ، جای دو سطر **L** و **M** را با هم عوض کنید.

۷. برای همان ماتریس**P** ، آن درایه از سطر **L** را که دارای بزرگترین قدرمطلق است پیدا کنید. به فرض صفر نبودن آن، هر درایه از سطر **L** را به آن تقسیم کنید.

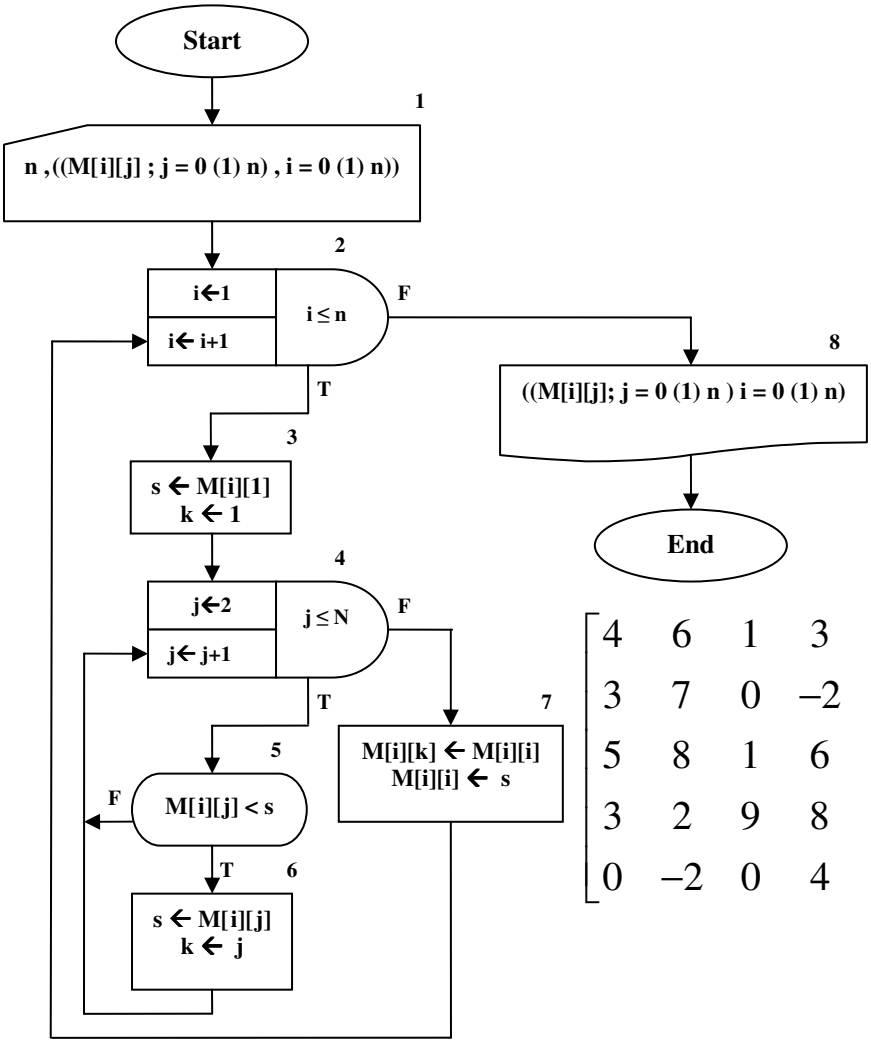
۸. در مباحثی که بلافاصله قبل از این مجموعهٔ تمرین‌ها آمد، ما قراردادهای مربوط به بیان سطح بالاتر را مورد بحث قراردادیم. برای مثال، در جایی که **A** یک لیست و**P** ماتریسی با ابعاد مناسب باشد، توافق شد که نماد انتساب



معنایش این باشد که: مقادیر سطر دوم **P** را به **A** منسوب کن.

اگر تاکنون این کار را نکرده‌اید، با به کار گرفتن یک بیان سطح بالاتر راه‌حل خود را برای تمرین ۷ ساده کنید. سپس ببینید آیا می‌توانید با ابتکار خود حلقه‌های بقیهٔ تمرینات ۴ تا ۸ را نیز حذف کنید.

۹. این سؤال مربوط به کارنمای زیر می‌شود. فرض کنید با اجرای جعبهٔ ۱ مقادیر ماتریس زیر خوانده شود. هنگامی که اجرا به جعبهٔ ۸ برسد مقدار M چیست؟



$$\begin{bmatrix} 4 & 6 & 1 & 3 & 9 \\ 3 & 7 & 0 & -2 & -4 \\ 5 & 8 & 1 & 6 & 2 \\ 3 & 2 & 9 & 8 & 6 \\ 0 & -2 & 0 & 4 & 1 \end{bmatrix}$$

شکل ۶- ۳۹ : کارنمای تمرین ۹

۱۰.الگوریتمی بنویسید که لیست اسامی n دانشجو را به همراه معدل آنها بخواند و سپس بدون این که ترتیب دانشجویان دو لیست را بهم بزند، به آنها رتبه بدهد. مثلاً به دانشجویانی که معدل ۲۰ دارند رتبه ۱ بدهد و به همین ترتیب ادامه دهد. در نهایت لیست دانشجویان را به ترتیب رتبهٔ آنها چاپ کند.

۱۱.فرض کنید ماتریس Q با M سطر و N ستون در حافظه موجود است. سطر L-اُم Q را برای پیدا کردن کوچکترین مقدار مورد جستجو قرار دهید. این مقدار را به SMALL منسوب کنید.

۱۲.فرض کنید ماتریس **Q** با **M** سطر و **N** ستون در حافظه موجود است. ستون **R**-اُم را به‌طور وارونه(یعنی از پایین به بالا) به‌منظور پیدا کردن اولیه درایه (در صورت وجود) که حداقل به بزرگی مقدار جاری **T** باشد مورد جستجو قرار دهید. شمارهٔ سطر این درایه را به **ROW** و مقدار آن را به **BIG** منسوب کنید. در صورتی که چنین مقداری یافت نشد ، مقدار صفر را به **ROW** منسوب کنید.

۱۳.فرض کنید که مقادیر مربوط به یک ماتریس**N×N** به نام **p** قبلاً در حافظه منسوب شده باشند. قطعه‌کارنمایی ترسیم کنید که (بدون نگرانی از ورودی و خروجی) قدم‌هایی را که در زیر توضیح داده شده است (به ترتیب) انجام بدهد.

قدم ۱ : به جای هر درایه از قطر اصلی **p** (از گوشهٔ بالای سمت چپ تا گوشهٔ پایین سمت راست) مربع مقدار آن درایه را قرار دهد.

قدم ۲ : بزرگترین درایهٔ سطر اول را به هریک از درایه‌های سطر آخر بیفزاید و نتایج حاصل را به‌عنوان مقادیر جدید در سطر آخر قرار دهد.

۱۴.فرض کنید یک جدول کلمات متقاطع در ماتریسی به نام **CWP** ذخیره شده باشد به‌طوری که هر عنصر از جدول یک نویسهٔ تنها یا یک جای خالی(که یک مربع سیاه را نمایش می‌دهد) باشد. شکل ۶-۴۰ نمایش ماتریسی یک جدول قابل قبول است. در این مورد یک ماتریس ۹ × ۹ مورد نیاز است.

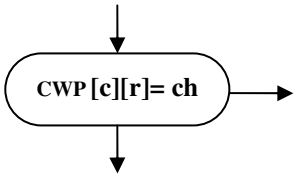
جعبه‌های ۳ الی ۹ کارنمای شکل ۶-۴۰ قدم‌های لازم برای جستجو در جدول و تعیین این‌که آیا اولین حرف آن کلمهٔ «افقی» که در سطر **r** از ستون **C** شروع می‌شود، مطابق با مقدار (تک) حرفی متغیر ورودی **ch** هست یا نه را نمایش می‌دهد. بعضی از جعبه‌های کارنما ناتمام هستند. کارنمای مزبور را مطالعه و تعیین کنید که آیا گزاره‌های (أ) تا (ت) درست هستند یا نادرست.

الف- جعبه‌های ۱و ۲ کارنما برای این منظورند که بترتیب جدول کلمات متقاطع تکمیل شده و داده‌های لازم برای جستجو را به داخل بیاورد.

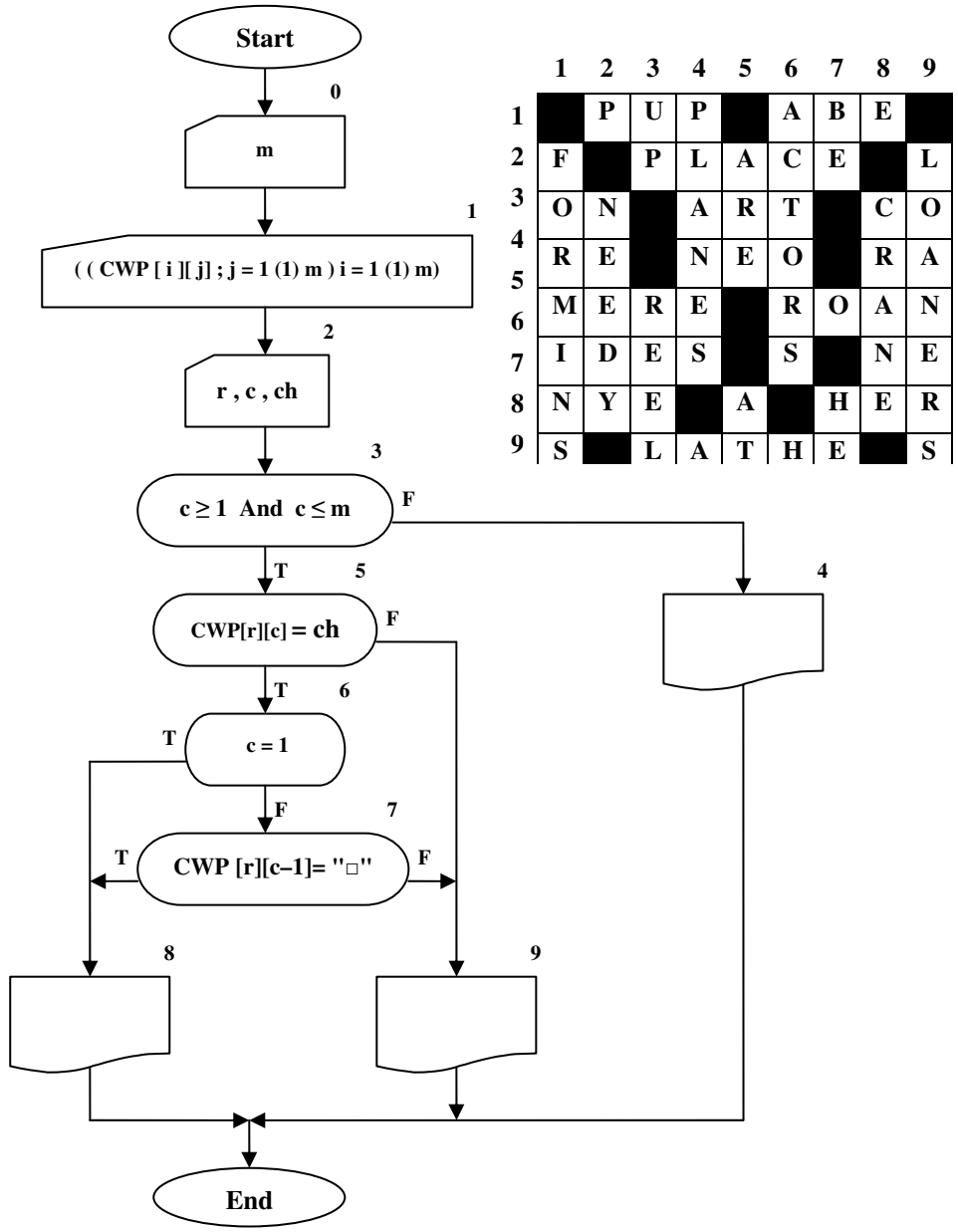
ب- جعبهٔ ۳ صحت شماره ستون، **c** ، را مورد بررسی قرارمی‌دهد، لکن بررسی مشابهی در مورد صحت شماره سطر، **r** ، به عمل نمی‌آید.

ج- فرض کنید که جدولی که در جعبهٔ ۱ داده شده است به داخل آورده شده است. به علاوه، فرض کنید که مقادیر داده‌های ۳ ، ۵ ، **'R'** در جعبهٔ ۲ به داخل آورده شده‌اند. جریان کنترل در نهایت به جعبهٔ ۸ خواهد رسید.

د- اگر قرار باشد الگوریتم به نحوی تغییر پیدا کند که بتواند برای جستجو کلمات عمودی که با مقدار حرفی متغیر **ch** شروع می‌شوند به کار رود، تغییر زیر کفایت می‌کند.



را جانشین جعبه ۵ کن.



	1	2	3	4	5	6	7	8	9
1		P	U	P		A	B	E	
2	F		P	L	A	C	E		L
3	O	N		A	R	T		C	O
4	R	E		N	E	O		R	A
5	M	E	R	E		R	O	A	N
6	I	D	E	S		S		N	E
7	N	Y	E		A		H	E	R
8	S		L	A	T	H	E		S

شکل ۶-۴۰ : کارنمای تمرین ۱۴

۱۵.کارنمایی رسم کنید که در مورد ماتریس **p** با **m** سطر و **n** ستون، در سطرهای شماره فرد و ستون‌های شماره زوج جستجو و عنصری را که دارای کوچکترین مقدار جبری و مخالف صفر است پیدا و آن را به **LEAST** منسوب کند. در حین انجام این جستجو، حساب تعداد عنصرهای مساوی صفر را که به آن برمی‌خورید در متغیر

ZTALLY نگه دارید و سپس مقادیر LEAST و ZTALLY را چاپ کند. اگر تمام عناصر صفر باشند، مقداری که برای LEAST چاپ می‌شود باید صفر باشد.

۱۶.ماتریس مربعی p با M سطر و M ستون موجود است. با جمع سطر به سطر مقادیر، مجموع تمام درایه‌هایی را که در سمت چپ قطر اصلی ماتریس مربعیP واقع شده‌اند را حساب کنید و آن را به SUM1 نسبت دهید. راهنمایی: برای خود تصویری از گروه مثلثی شکل درایه‌هایی که قرار است با هم جمع شوند بسازید. اولین سطری که در محاسبات وارد می‌شود کدام است؟ آخرین ستونی که در محاسبات وارد می‌شود کدام است؟ برای آن دسته از عناصر سطر i که در سمت چپ قطر اصلی واقع شده‌اند زیرنویس ستون آخرین درایهٔ سمت راست کدام است؟

۱۷.ماتریس مربعی P را در نظر بگیرید. با جمع سطر به سطر مقادیر، مجموع تمام عناصری را که در سمت راست قطر اصلی ماتریس مربع P واقع شده‌اند را حساب کنید.

۱۸.به «مثلث» سمت چپ قطر اصلی که در مسئلهٔ ۱۶ با آن کار کردید غالباً «مثلث پایینی» و به مثلث سمت راست قطر اصلی غالباً «مثلث بالایی» گفته می‌شود. در این تمرین می‌خواهیم از ستون آخر شروع نموده و مثلث بالایی را ستون به ستون جستجو کنیم. هر ستون را از بالا به پایین جستجو می‌کنیم تا اولین درایه‌ای را که از نظر قدرمطلق حداقل دو برابر درایه ماقبل خود در همان ستون باشد پیدا کنیم. درایه‌ای که چنین افزایش قدرمطلقی را از خود نشان دهد PIG نامیده خواهد شد.

هر ستون از مثلث بالایی، به استثنای اولین ستون سمت چپ، ممکن است شامل یک PIG باشد. تمام مقادیر PIG را به همان ترتیبی که پیدا می‌شوند، به همراه شمارهٔ سطر و ستون آن‌ها، I و J ، چاپ کنید. اگر هیچ PIG پیدا نشد پیام «هیچ» را چاپ کنید. کوچک‌ترین ماتریسی که می‌تواند PIG داشته باشد چیست؟ (جواب: ۳ × ۳).

راهنمایی: آیا بالاترین عنصر یک ستون می‌تواند PIG باشد؟ زیرنویس سطر پایین‌ترین عنصرستون j- اُم چیست؟

۱۹.الگوریتمی بنویسید که ماتریسی مربعی را خوانده و سپس تشخیص دهد آن ماتریس متقارن است یا نه؟ ماتریسی متقارن نامیده می‌شود که عناصر سطر I-اُم با عناصر ستون I-اُم برابر باشد.

۲۰.کارنمای الگوریتمی را رسم کنید که شکل یک ساعت شنی محصور شده در یک قاب را، به‌طوری که در شکل زیر با چاپ علامت «* » و اندازهٔ ۷ نشان داده شده است چاپ کند. قاب را طوری بسازید که درست مناسب ساعت شنی باشد و فقط دو قلم اطلاعاتی یعنی علامتی که برای چاپ ساعت مورد استفاده قرار می‌گیرد، C، و اندازهٔ آن، I ، را از ورودی بخوانید.

```
- - - - - - - - -
| * * * * * * * |
|   * * * * * *   |
|     * * * *     |
|       *         |
|     * * * *     |
|   * * * * * *   |
| * * * * * * * |
- - - - - - - - -
```

۲۱.صفحهٔ شطرنج را می‌توان به‌صورت یک الگوی نویسه‌ای ۸ × ۸ تجسم کرد. کارنمایی بسازید که دو عدد صحیح (N و M) را به‌عنوان محل قرار گرفتن وزیر (Q) بخواند و نقش صفحهٔ شطرنج را طوری چاپ‌کند که در محل‌های تحت پوشش وزیر (محل‌هایی که در همان سطر یا ستون یا قطر قرار گرفته‌اند) علامت ستاره (*) و در محـل‌های دیگر علامـت منها چاپ شود. شکل زیر حل مسئله را برای حالتی که وزیـر در محل (۴ و۳) قرار گرفته است نشان می‌دهد.

الف- حل کارنمایی مسئله را بدون استفاده از متغیر لیستی یا آرایه بسازید.

ب- حل کارنمایی مسئله را بدون استفاده از آرایه ولی با استفاده از متغیر لیستی بسازید.

ج- حل کارنمایی مسئله را با مجاز بودن استفاده از آرایه بسازید.

```
- * - * - * - -
- - * * * - - -
* * * Q * * * *
- - * * * - - -
- * - * - * - -
* - - * - - * -
- - - * - - - *
- - - * - - - -
```

۲۲.این سؤال مربوط به الگوریتمی برای تولید صورت رمزی یک پیام است. این الگوریتم از دو لیست IN و OUT، مانند آنچه در زیر نشان داده شده است، استفاده می‌کند. هر لیست شامل تمام ۲۶ حرف الفبای انگلیسی است ولی ترتیب حروف در آنها متفاوت است.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
I	C	T	Q	Y	J	W	E	V	A	O	K	B	H	P	R	G	L	S	X	D	N	Z	M	U	F

الگوریتم به طریق زیر کار می‌کند.

الف- مقادیر اولیه برای لیست‌های IN و OUT به داخل آورده می‌شوند.

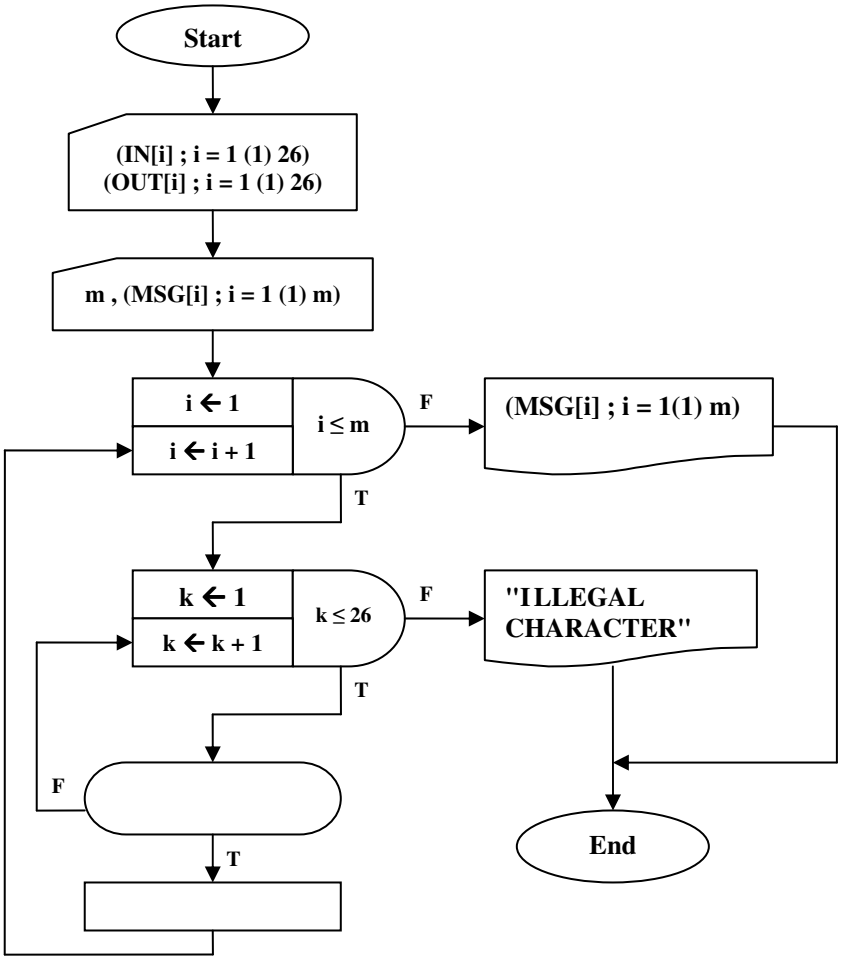
ب- m ، یعنی تعداد نویسه‌های پیام به داخل آورده می‌شود.

ج- پیام را خـوانده و هر کاراکـتر(نویـسه) آن به ترتیب در یک عنصر لیست MSG قرارداده می‌شود. (فرض‌کنید پیام فقط از حروف تشکیل شده است و جای خالی، عدد و غیره در آن وجودندارد)

د- هر حرف پیام، به رمز در می‌آید بدین ترتیب که محل آن در لیست IN مشخص می‌شود و سپس حرف متناظر آن در لیست OUT جایگزین آن حرف می‌شود. در اینجا «عنصر متناظر» یعنی عنصری که دارای همان زیرنویس باشد. (مثلاً در مورد IN و OUT که در زیر نشان داده شده است، هر بار که D در MSG باشد به جای آن Q قرار خواهد گرفت زیرا D ، IN[4] است و Q، OUT[4] است.)

و- شکل رمزی پیام چاپ می‌شود.

در زیر، کارنمای ناتمامی برای این الگوریتم نشان داده شده است. وظیفهٔ شما آن است که این کارنما را تکمیل کنید.



شکل ۶- ۴۱ : کارنمای تمرین ۲۲

۲۳.پس از حل مسئلهٔ قبل کارنمای، شکل ۶-۴۱ را مطابق روش مطرح شده در بخش ۶-۷ به‌گونه‌ای بازنویسی کنید که تنها از یک نقطه به سمت دکمهٔ پایان برود.

۲۴.یک صفحهٔ شطرنج را در نظر بگیرید. یک مهره در خانهٔ گوشه سمت چپ و پایین این صفحه قرار دارد. این خانه را «خانهٔ شروع» می‌نامیم و با مختصات ۱×۱ نمایش می‌دهیم. این مهره در هر بار حرکت فقط می‌تواند یک خانه به سمت راست یا یک خانه به سمت بالا برود. الگوریتمی بنویسید که کلیهٔ مسیرهای ممکن برای رسیدن این مهره از خانهٔ شروع به خانه‌ای با مختصات **i×j** را تولید کند. هر مسیر با دنباله‌ای از **R** و **U** مشخص می‌شود که در آن هر حرکت به سمت راست با **R** و هر حرکت به سمت بالا با **U** نشان داده می‌شود.

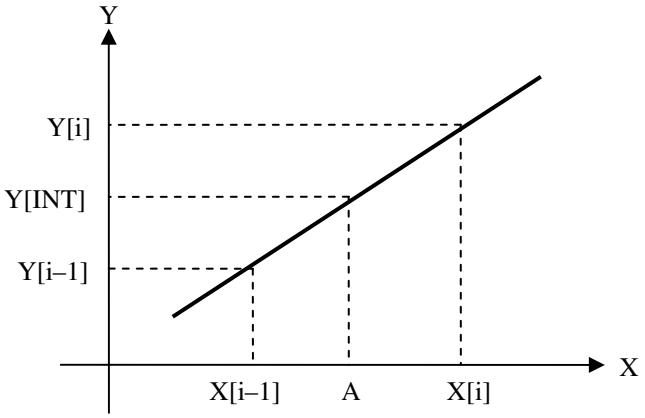
۲۵.جدول مرتبی از مقادیر **X** و **sinX** به شما داده شده است. کار شما ساختن کارنمایی است که:

الف- این مقادیر را به داخل کامپیوتر بخواند،

ب- آنگاه مقداری از **A** را که ممکن است مطابق یکی از مقادیر **X** در جدول باشد یا نباشد، را بخواند،

ج- در صورت امکان مقدار نظیر **sinX** را چاپ کند. در صورتی که **A** با هیچ یک از مقادیر **X** در جدول مطابقت نکند، اگر مقدار **A** خارج از قلمرو تحت پوشش جدول باشد پیام مناسبی مبنی بر این امر چاپ کند و اگر **A** بین دو مقدار **X** قرار می‌گیرد، دو مقدار مذکور و مقادیر **sin** نظیر آنها را چاپ کند.

۲۶.شکل ۶-۴۲ را مطالعه کنید و سپس کارنمای مسئلهٔ قبل را توسعه دهید به‌طوری که با استفاده از درون‌یابی خطی از دو مقدار دربرگیرندهٔ **sinX** در جدول، مقادیر **Y[INT]** یعنی درون‌یابی خطی **sinA** را محاسبه کند.



شکل ۶- ۴۲ : نمایش درون‌یابی مستقیم

۲۷.جدول مرتب نشده‌ای از مقادیر **(X[k],Y[k] ; k = 1(1) N)** به ما داده شده است. حداقل و حداکثر مقادیر **X** معلوم است. می‌خواهیم به دفعات زیاد به این جدول مراجعه کنیم. دو راه در جلوی پای ما باز است:

۱. ابتدا جدول را مرتب کنیم و آنگاه جدول خوانی با روش جستجوی دودویی را انجام دهیم. و یا

۲. کارنمای جدیدی برای جستجودرجدول مرتب‌نشده بسازیم. شکل ۶-۴۳ راه حلی است برای مورد دوم. در این شکل، مقادیر معلوم حداقل و حداکثر **X** در جعبهٔ ۴ به داخل خوانده می‌شوند.

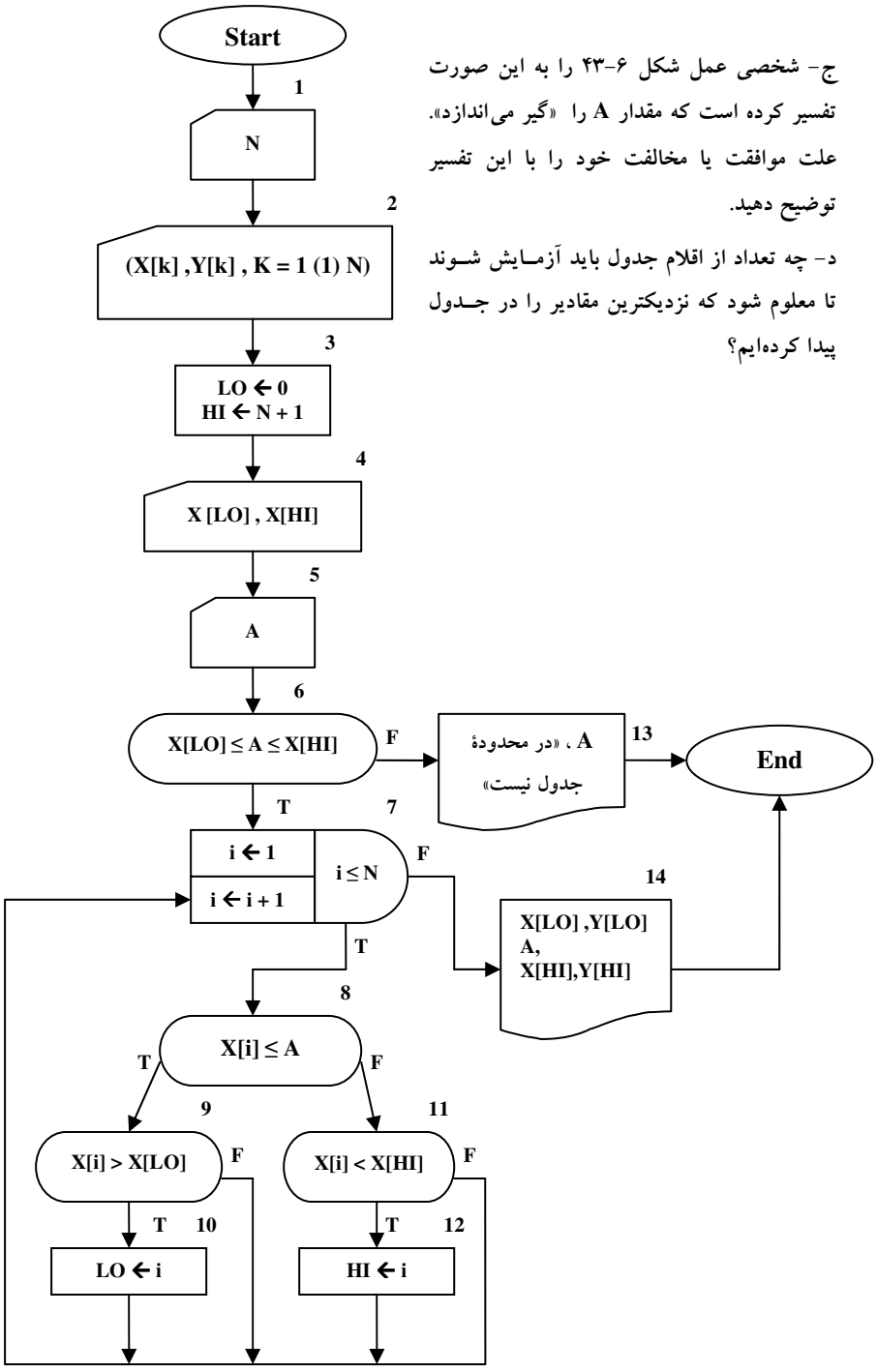
شکل ۶-۴۳ را مطالعه کنید و به سؤالات زیر پاسخ دهید.

الف- برای جدول‌های با تعداد اقلام کم، روش (۱) بهتر است یا روش (۲)؟ کدام یک برای جدول‌های بزرگ بهتر است؟

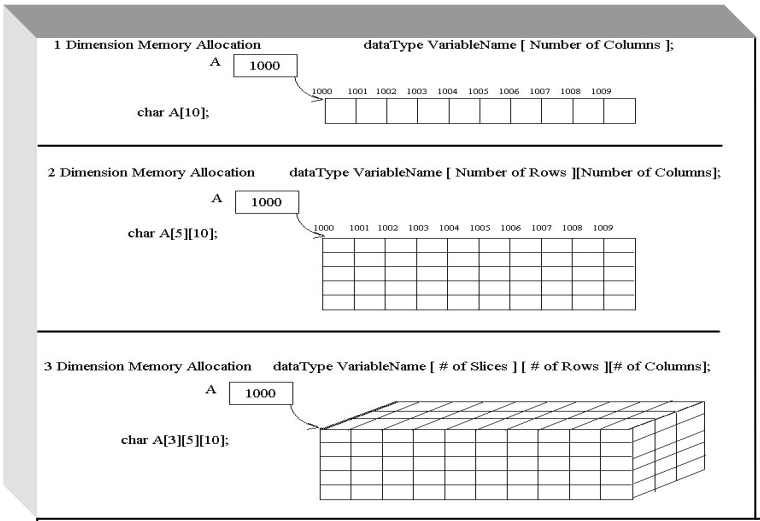
ب- فرض کنید **A** مساوی **X[LO]** ، یعنی مقدار حداقلی که برای **X** داده شده است، باشد. در اولین اجرای جعبهٔ ۸ ، تحت چه شرایطی مسیر خروجی درست اختیار می‌شود؟

ج- شخصی عمل شکل ۶-۴۳ را به این صورت تفسیر کرده است که مقدار A را «گیر می‌اندازد». علت موافقت یا مخالفت خود را با این تفسیر توضیح دهید.

د- چه تعداد از اقلام جدول باید آزمایش شوند تا معلوم شود که نزدیکترین مقادیر را در جدول پیدا کرده‌ایم؟



شکل ۶- ۴۳ : کارنمای تمرین ۲۷



فصل هفتم

آرایه‌ها و رشته‌ها در C++

اهداف فصل و چکیده مطالب :

۱. نحوهٔ تعریف آرایه‌های یک بعدی و چند بعدی
۲. چگونگی سازمان‌دهی آرایه‌ها در حافظه
۳. کانتینر vector (بردارها)
۴. توابع عمل کننده بر روی وکتورها
۵. وکتورهای چند بعدی
۶. رشته‌های زبان C
۷. سرفایل string و رشته‌های نوع string
۸. امکانات فراهم شده برای عملیات برروی توابع زبان C
۹. توابع عمل کننده بر روی رشته های نوع string

۷-۱ : آرایه های یک بعدی

چنانکه در فصل قبل دیدید گاهی نیازمند دسته‌بندی برخی داده‌های هم نوع هستیم. ساده‌ترین را برای دسته‌بندی عناصر هم نوع در C++ استفاده از آرایه است. در حقیقت در هر زبان برنامه‌نویسی مبحثی تحت‌عنوان «ساختمان‌داده‌ها» مطرح می‌گردد که پیرامون نحوهٔ سازماندهی داده‌ها درحافظهٔ اصلی (RAM) صحبت می‌کند. و آرایه اولین ساختمان داده‌ای است که با آن آشنا می‌گردید.

تعریف آرایه

جهت تعریف هرگونه آرایه، مانند تعریف دیگر متغیرها در C++ باید نوع آرایه از نظر نوع داده‌ای که می‌تواند ذخیره‌کند را مشخص سازیم. همچنین هر آرایه دارای نامی است که مطابق قواعد نامگذاری متغیرها باید تعیین شود، و در نهایت اینکه باید طول آرایه را در داخل دو علامت کروشه باز و بسته پس از نام آرایه بیاوریم. با توجه به توضیحات فوق تعریف آرایه‌های یک بعدی(لیست‌ها) در C++ به‌صورت کلی زیر است:

نوع آرایه نام آرایه [طول آرایه] ;

به‌عنوان مثال در زیر تعدادی آرایه تعریف شده است.

```
int   A[5];
```

در این نمونه آرایه‌ای با نام A تعریف شده که قادر است 5 عدد int را در خود ذخیره کند.

```
char  B[100];
```

در این نمونه آرایه‌ای با نام B تعریف شده که قادر است 100 کاراکتر از نوع char را در خود ذخیره کند.

```
const int length=20;
float  C[length];
```

و در این نمونهٔ آخر آرایه‌ای با نام C که قادر است 20 عدد از نوع float را در خود ذخیره کند تعریف شده است. نکتهٔ قابل توجه در خصوص تعریف آرایه‌ها اینکه، طول آرایه باید به‌طور صریح با قرار دادن یک عدد صحیح یا یک متغیر صحیح ثابت (const) مشخص گردد. لذا به هیچ‌عنوان نمی‌توان از متغیرهای غیر ثابت به جهت تعیین طول آرایه در تعریف آنها استفاده کرد. به‌عنوان مثال تکه کد زیر با این هدف نوشته شده که طول آرایه را از کاربر دریافت کند و آرایه‌ای به همان اندازه بسازد.

```
int n;
cin>>n;
int array[n];
```

اما اگر این کد را با هر کامپایلر C++ کامپایل کنید با مجموعه خطای زیر بر خورد خواهید کرد:

error C2057: expected constant expression
error C2466: cannot allocate an array of constant size 0
error C2133: 'array' : unknown size

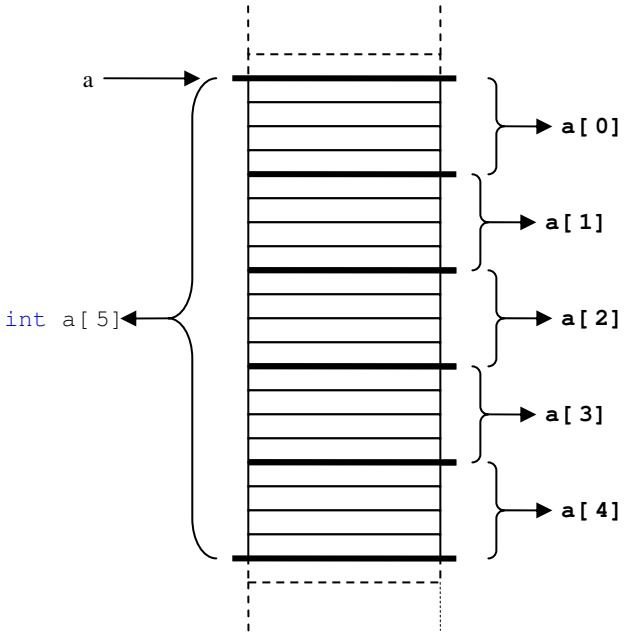
البته این به‌این‌معنا نیست که در C++ نمی‌توان آرایه‌هایی با طول دلخواه کاربر تعریف کرد، بلکه تعریف چنین آرایه‌هایی نیازمند استفاده از اشاره‌گرها است که در فصل ۱۱ چگونگی تعریف آرایه با طول متغیر توسط اشاره‌گر را خواهید دید. همچنین روش دیگر استفاده از بردارها است که در همین فصل با آن آشنا خواهید شد.

دسترسی به عناصر یک آرایه

فرض کنید آرایه‌ای به‌صورت زیر داریم:

```
int a[5];
```

از آنجایی که هر متغیر از نوع **int** در چهار بایت ذخیره می‌شود، برای ذخیرهٔ پنج متغیر از این نوع به **20** بایت از حافظه نیازمندیم. لذا برای چنین آرایه‌ای در حافظه **20** بایت فضا اختصاص داده می‌شود. بنابراین محدوده‌ای همانند شکل ۷-۱ در حافظه به آرایهٔ **a** نسبت داده می‌شود.



شکل ۷- ۱ : نمایی از فضای اختصاص یافته به یک آرایهٔ یک بعدی

مراجعه به خانه‌های حافظه توسط آدرس آنها صورت می‌پذیرد. هر بایت از حافظه دارای یک آدرس به‌صورت یک عدد هگزا دسیمال است. به‌عنوان مثال ممکن است آدرس یک خانه از حافظه به‌صورت **0F12H** باشد. بنابراین آدرس بایت بعد از آن به‌صورت **0F13H** خواهد بود.

هنگامی که کامپایلر به دستور:

```
int a[5];
```

می‌رسد ابتدا **20** بایت از حافظهٔ اصلی را جهت ذخیرهٔ **5** متغیر **int** اختصاص می‌دهد. اما نام آرایه همانند متغیری می‌ماند که حاوی آدرس اولین خانه از فضای اختصاص یافته به آرایه می‌باشد. به عبارت دیگر نام آرایه یعنی **a** را چنین تصور کنید که به اولین خانه از **20** بایت اختصاص یافته به آرایه اشاره می‌کند. حال با در دست داشتن آدرس اولین خانه از آرایه می‌توانیم به تک تک عناصر آرایه مراجعه کنیم. هنگامی که می‌نویسیم:

```
a[0]=5;
```

منظور این است که به آدرسی که **a** اشاره می‌کند برو، سپس **(4*0)** خانه به جلو برو و محتویات چهار بایت بعدی از این آدرس را به‌عنوان یک عدد **int** از عناصر آرایه را برابر **5** قرار بده. (کل این چهار بایت یک متغیر **int** را تشکیل می‌دهند و عدد **5** به‌صورت یک عدد **32** بیتی ذخیره می‌شود.)

حال اگر بنویسیم:

```
a[1]=6;
```

کنترل اجرای برنامه با رسیدن به این دستور ابتدا به مکانی که **a** به آن اشاره می‌کند رفته و سپس **(4*1)** خانه به جلو می‌رود (یعنی عنصر **a[0]** را پشت سر گذشته و به ابتدای **a[1]** می‌رود) و محتویات ۴ بایت بعد از آن را به‌عنوان متغیر **int** بعدی آرایه برابر عدد **6** قرار می‌دهد.

تمامی این مباحث را به این جهت مطرح ساختیم که متذکر شویم در **C++** اندیس آرایه از ۱ شروع نمی‌شود، بلکه از صفر شروع می‌گردد. بنابراین به‌طور کلی برای آرایه‌ای به طول **n** اندیس آرایه از **0** تا **(n–1)** تغییرمی‌کند.

متأسفانه در **C++** اندیس آرایه چک نمی‌شود. به‌عنوان مثال اگر دستوری به‌صورت:

```
a[5]=10;
```

در برنامه بنویسید با توجه به توضیحات فوق کنترل اجرای برنامه به مکانی که آدرس آن در **a** قرار گرفته رفته است و سپس **(4*5)** بایت به جلو می‌رود. یعنی از حدود آرایهٔ ۵ عنصری ما خارج شده و به بعد از آخرین عنصر آرایه رفته و محتویات ۴ بایت بعدی را برابر ۱۰ قرار می‌دهد. لذا این دستور بدون هیچگونه خطا یا هشداری اجرا شده و روند برنامه ادامه پیدا می‌کند. اما نکته بسیار مهم این است که اگر فضایی که توسط این دستور مورد دستکاری قرار می‌گیرد مربوط به متغیر دیگر مثل **x** باشد، ممکن است نتایج غیر منتظرهٔ بسیاری پیش آید. لذا توجه به حدود آرایه توسط برنامه نویسان بسیار حائز اهمیت است.

فضای اشغال شده توسط آرایه

میزان فضای اشغال شده توسط هر آرایه به دو پارامتر *نوع آرایه* و *طول آرایه* بستگی دارد. میزان حافظه‌ای که توسط یک آرایه اشغال می‌شود برحسب بایت از رابطهٔ زیر به دست می‌آید:

طول آرایه × طول نوع آرایه بر حسب بایت = میزان فضای اشغال شده توسط آرایه (به بایت)

مقداردهی به عناصر آرایه‌های یک بعدی

برای مقدار اولیه دادن به عناصر آرایه چندین روش وجود دارد که به ترتیب در زیر مطرح می‌سازیم:

۱. **در یافت مقادیر از کاربر:** در این روش به‌صورت زیر عمل می‌کنیم:

```
int a[10];
  for(int i=0;i<10;i++)
       cin>>a[i];
```

نکتهٔ قابل توجه اینکه اصولاً در حلقه‌های تکرار که با آرایه سروکار داریم به‌جای آن‌که شرط حلقه را به صورت **i <= n–1** بنویسیم به‌صورت **i < n** می‌نویسیم. چرا که بدین طریق با یک نگاه بر **n** می‌توانیم طول آرایه را تشخیص دهیم.

۲. **مقداردهی اولیه به عناصر آرایه هنگام تعریف:** به‌عنوان مثال در زیر آرایهٔ x به‌گونه‌ای تعریف شده که عناصر آن یعنی x[0]، x[1] و x[2] به ترتیب مقادیر ۱ و ۳ و ۵ را در خود نگه می‌دارند.

```
int  x[3]={1,3,5};
```

نکتهٔ قابل توجه آن‌که اگر به روش فوق آرایه‌ای را تعریف کنید می‌توانید طول آرایه را ذکر نکنید. در آن صورت کامپایلر براساس تعداد اعداد ذکر شده در بین دو علامت آکولاد باز و بسته طول آرایه را تعیین می‌کند. به‌عنوان مثال در زیر طول آرایهٔ f برابر 5 تعیین می‌شود.

```
float  f[ ]={5.5,3.7,6.4,8.2,3.43};
```

از طرفی اگر تعداد داده‌های ذکر شده در بین دو علامت آکولاد باز و بسته کمتر از طول آرایه باشد کامپایلر عناصر باقی ماندهٔ آرایه را برابر صفر قرارمی‌دهد. به‌عنوان مثال در تکه کد زیر x[0] برابر 2 و x[1] برابر 7 و بقیهٔ عناصر آرایه برابر صفر قرار داده می‌شوند.

```
int z[10]={2,7};
```

بنابراین وقتی می‌خواهیم عناصر یک آرایه را به صفر مقدار اولیه بدهیم می‌توان به یکی از دو روش زیر عمل کرد:

```
int x[10]={0};
int x[10]={};
```

مثال ۷-۱: برنامه‌ای که تعداد ۱۰ عدد را از ورودی خوانده و در آرایه‌ای قرار می‌دهد و سپس معدل آنها را حساب می‌کند.

```
1.  //This program calculates average of array's elements.
2.  #include <iostream>
3.  using namespace std;
4.  int main()
5.  {
6.       cout<<"Please enter 10 integer numbers"
7.           <<" to calculate them average :";
8.         int data[10]={0};
9.          for(int i=0;i<10;i++)
10.           {
11.              cout<<"Num["<<(i+1)<<"] = ";
12.               cin>>data[i];
13.           }
14.         int sum=0;
15.           for(i=0;i<10;i++)
16.             sum+=data[i];
17.         float avg;
18.               avg=static_cast <float> (sum)/10;
19.         cout<<"Average of these numbers is :"<<avg<<endl;
20.       return 0;
21.  }
```

مثال ۷-۲ : برنامه‌ای که با دریافت تاریخ یک روز سال جاری تعداد روزهای گذشته از ابتدای سال را مشخص می‌کند. این برنامه برای سال‌های کبیسه جواب اشتباه می‌دهد.

```
1.  //This program shows days from start of year.
2.  #include <iostream>
3.  #include <conio.h>
4.  using namespace std;
5.  int main()
6.  {
7.     short month, day, total_days;
8.     int day_per_month[12]={31,31,31,31,31,31,
9.                            30,30,30,30,30,29};
10.    cout<<"Enter month (1 to 12) : ";
11.     cin>>month;
12.     if(month<1 || month>12)
13.      {
14.          cout<<"Wrong input! Press any key to end.\n";
15.           getch();
16.            return 0;
17.      }
18.     cout<<"Enter day (1 to 31) : ";
19.      cin>>day;
20.       total_days=day;
21.         for(int i=0;i<month-1;i++)
22.            total_days+=day_per_month[i];
23.         cout<<"Total days from start of year is:"
24.             <<total_days<<endl;
25.   return 0;
26. }
```

مثال ۷-۳ : از تعداد ۴۰ دانشجو خواسته شده که نظر خود را در مورد غذای سلف دانشگاه اعلام کنند. هر دانشجو عددی بین یک تا ده را به‌عنوان نمرهٔ کیفیت غذا اعلام کرده است. در برنامهٔ زیر تعداد هر یک از نمرات ۱ تا ۱۰ شمرده شده است. چهل جواب دریافت شده را در یک آرایه قرار داده‌ایم.

```
1.  //This is a student poll program.
2.  #include <iostream>
3.  #include <iomanip>
4.  using namespace std;
5.  #define answerSize 40
6.  int main()
7.  {
8.     const int frequencySize=11;
9.      char answer[answerSize]=
10.      {1,2,6,4,8,5,9,7,8,10,1,6,3,8,6,10,3,8,2,7,
11.       6,5,7,6,8,6,7,5,6,6,5,6,7,5,6,4,8,6,8,10};
12.     int frequency[frequencySize]={0};
13.      for(int count=0;count<answerSize;count++)
14.           ++frequency[answer[count]];
15.      cout<<"Rating"<<setw(17)<<"Frequency"<<endl;
```

```
16.         for(int rating=1;rating<frequencySize;rating++)
17.            cout<<setw(6)<<rating<<setw(17)
18.                <<frequency[rating]<<endl;
19.      return 0;
20.   }
```

توضیح مثال: در این مثال دو نکتهٔ قابل توجه وجود دارد. اولین نکته نحوهٔ شمارش پاسخ‌ها در خطوط ۱۲ و ۱۳ می‌باشد، به‌طوری که هر نمونه‌ای که از مجموعهٔ جواب‌ها در آرایهٔ **answer** خوانده می‌شود به شمارندهٔ آن نمره یک واحد افزوده می‌شود. به‌عنوان مثال هنگامی که **answer[3]** برابر **4** می‌باشد، یعنی نمرهٔ خوانده شده برابر **4** است، به عنصری با زیرنویس ۴ در آرایهٔ **frequency** یک واحد افزوده می‌شود. به عبارت دیگر در این صورت خط ۱۳ این کد به‌صورت زیر تعبیر می‌شود:

```
++frequency[4];
```

و دومین نکته استفاده از دستکاری کنندهٔ **setw** است. یادآور می‌شویم که دستکاری کننده‌ها، عملگرهایی هستند که برای تغییر یا پردازش داده‌ها برای چاپ در خروجی با عملگر درج >> به کار می‌روند. قبلاً با دستکاری کنندهٔ **endl** آشنا شدید اکنون توجه شما را به دستکاری کنندهٔ **setw** جلب می‌کنیم که طول میدان خروجی را تغییر می‌دهد. برای استفاده از این دستکاری کننده باید هدرفایل **iomanip** را به برنامه اضافه کرد.

هر مقداری که توسط **cout** چاپ می‌شود میدانی از فضا را اشغال می‌کند. این میدان، محدوده‌ای است که خروجی می‌تواند در آن چاپ گردد. میدان پیش‌فرض دارای طولی به قدر کافی بزرگ است تا بتواند مقدار موردنظر را در خود چاپ (ذخیره) کند. مثلاً عدد صحیح **576** دارای میدانی به طول ۳ کاراکتر و رشتهٔ **"Computer"** میدانی به طول ۸ کاراکتر را اشغال می‌کند. با وجود این در بعضی از موارد، طول میدان پیش‌فرض منتهی به نتایج مطلوبی نمی‌شود. لذا می‌توان به‌واسطهٔ دستکاری کنندهٔ **setw** میدان پیش‌فرض را تغییر داد. دستکاری کنندهٔ **setw** باعث می‌شود عدد یا رشته‌ای که بعد از آن در جریان داده‌ها می‌آید در میدانی به‌طول **n** کاراکتر چاپ شود که در آن **n**، ورودی تابع **setw(n)** است. مقدار عدد یا رشتهٔ موردنظر در داخلِ میدان، به‌صورت تراز شده از راست چیده می‌شود، لذا خروجی برنامهٔ مثال ۳-۷ به‌صورت زیر خواهد بود:

```
Rating          frequency
     1                  2
     2                  2
     3                  2
     4                  2
     5                  5
     6                 11
     7                  5
     8                  7
     9                  1
    10                  3
```

۷-۲: آرایه های چند بعدی

با آرایه‌های دو بعدی و کاربرد در فصل قبل آشنا شدید. برای تعریف آرایه‌های دو بعدی تنها کافی است پس از ذکر نوع و نام آرایه ابعاد آرایه را در دو علامت کروشه باز و بستهٔ جداگانه ذکر کنیم. به‌عنوان مثال در زیر آرایه‌ای با نام **Score** از نوع **int** با ۳۰ سطر و ۵ ستون تعریف کرده‌ایم از این آرایه تصمیم داریم برای ذخیرهٔ نمرات ۳۰ دانشجو که هر کدام ۴ نمره دارند استفاده کنیم. لذا ستون نخست آرایه شمارهٔ دانشجویی و ۴ ستون بعدی هر کدام یک نمرهٔ دانشجو را در خود ذخیره می‌کنند.

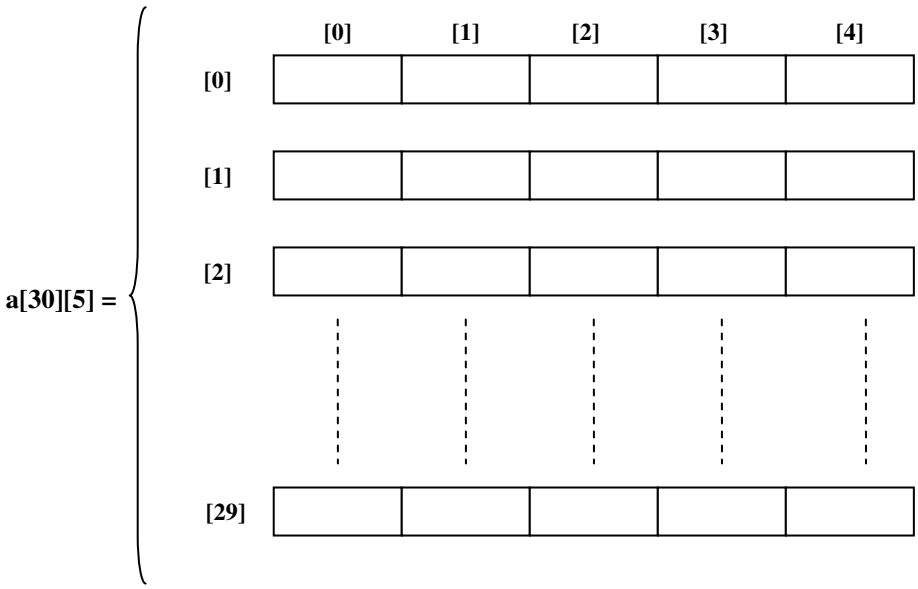
```
int Score[30][5];
```

Score[0][0]	Score[0][1]	Score[0][2]	Score[0][3]	Score[0][4]
Score[1][0]	Score[1][1]	Score[1][2]	Score[1][3]	Score[1][4]
⋮	⋮	⋮	⋮	⋮
Score[29][0]	Score[29][1]	Score[29][2]	Score[29][3]	Score[29][4]

شکل ۷- ۲ : نمایی از آرایهٔ دو بعدی

آرایه‌ای از آرایه‌ها

آرایه‌های دو بعدی را می‌توان به‌عنوان آرایه‌ای از آرایه‌های یک بعدی در نظر گرفت،به‌طوری که می‌توان هر سطر آرایهٔ دو بعدی را یک آرایهٔ خطی(یک بعدی) همانند شکل ۳-۷ تصور کرد.

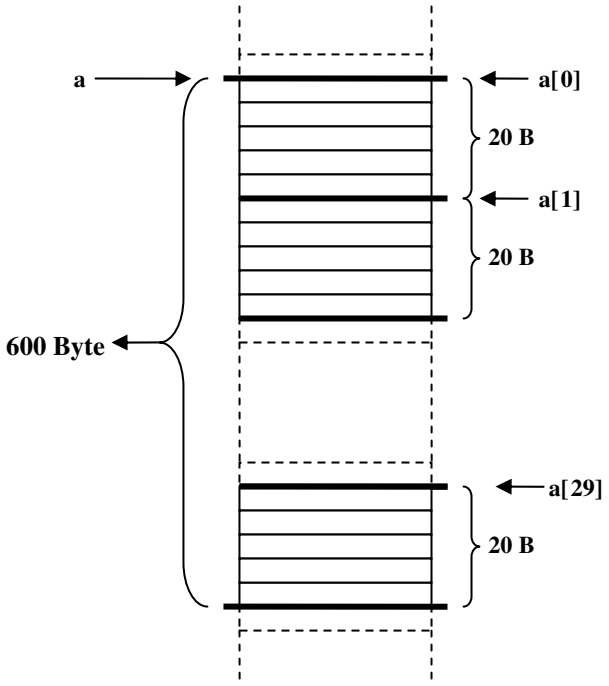


شکل ۷- ۳ : آرایهٔ دو بعدی به‌عنوان آرایه‌ای از آرایه‌های یک بعدی

اما واقعیت هم همین است که هیچگاه در حافظه آرایشی از بایت‌های حافظه همانند شکل ۷-۲ به‌عنوان آرایه دو بعدی تشکیل نمی‌گردد. بلکه آرایهٔ دو بعدی همانند یک آرایه یک بعدی و به‌صورت خطی ذخیره می‌شود. هنگامی که کامپایلر به دستوری مانند:

```
int a[30][5];
```

می‌رسد تعداد ۴ × ۵ × ۳۰ بایت از حافظه را به آرایهٔ **a** اختصاص می‌دهد.(به شکل ۷-۴ توجه کنید.) پس از اختصاص فضای لازم آدرس اولین بایت از آرایه در متغیرِ اشاره‌گری به نام **a** قرارمی‌گیرد. یعنی **a** به اولین خانه از آرایه اشاره می‌کند. اما چون می‌خواهیم آرایه‌ای دو بعدی داشته باشیم، چنانکه در شکل ۷-۳ دیدید آرایه های دو بعدی را می‌توان به‌صورت آرایه‌ای از آرایه‌های یک بعدی تصور کرد. لذا این ۶۰۰ بایت در حافظه به‌صورت ۳۰ لیست ۲۰تایی دسته‌بندی می‌گردد. از آنجا که هر ۴ خانه معرف یک عدد **int** است در نتیجه در هر لیست ۵ عدد **int** قابل ذخیره‌سازی می‌باشد. حال به عناصر آرایهٔ یک بعدی **a[30]** به ترتیب در **a[0]** , **a[1]** , ... , **a[29]** آدرس ابتدای هر یک از این لیست‌ها جای می‌گیرد.



شکل ۷- ۴ : نمایی از فضای اختصاص یافته به یک آرایهٔ دو بعدی

بنابراین هنگامی که می‌نویسیم:

```
a[1][2]=5;
```

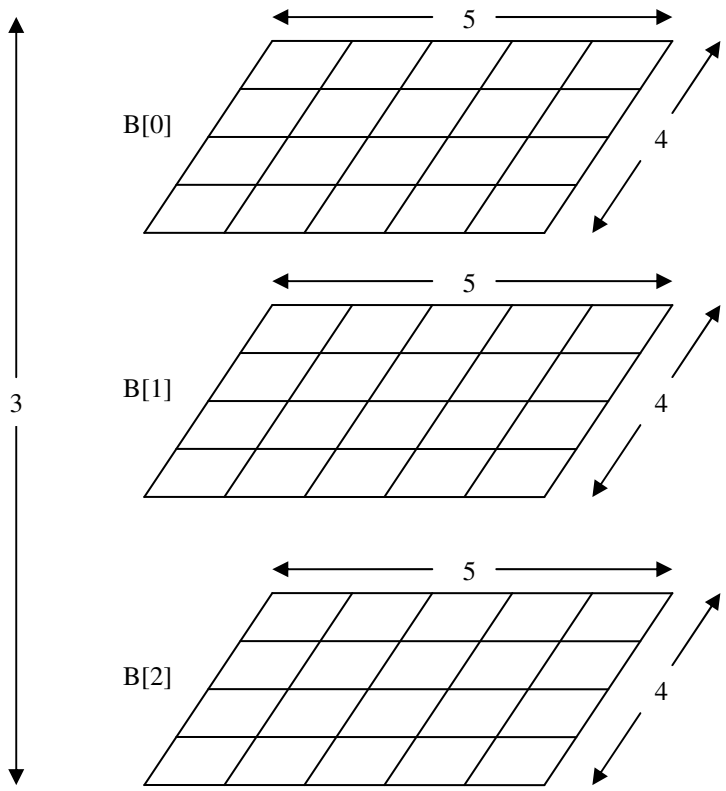
کنترل اجرای برنامه ابتدا به مکانی که **a[1]** به آن اشاره می‌کند می‌رود و سپس **(2*4)** بایت جلو رفته و در **4** بایت بعدی عدد **5** را قرار می‌دهد. اما هیچ مکانی با نام **a[1]** وجود ندارد که بخواهد آدرسی را در خود ذخیره سازد. بلکه

تنها یک متغیر(اشاره‌گر) **a** داریم که به ابتدای آرایه اشاره می‌کند. هنگامی که کامپایلر به **a[1]** برخورد می‌کند، ابتدا به خانه‌ای از حافظه که **a** به آن اشاره می‌کند می‌رود (یعنی به ابتدای آرایه می‌رود) سپس **(1*20)** خانه جلو می‌رود (یعنی ردیف اول آرایه را رد می‌کند). تا اینجا به مکانی رفت که **a[1]** به آن اشاره می‌کند و سپس **(2*4)** بایت دیگر نیز جلو می‌رود. در این صورت به مکان **A[1][2]** رسیده است. این روند آدرس‌یابی برای تمامی دیگر انواع آرایه نیز صادق است.

همچنین دیگر انواع آرایه در **C++** را نیز می‌توان به‌عنوان آرایه‌ای از آرایه‌ها در نظر گرفت. به‌عنوان مثال می‌توان آرایهٔ سه بعدی **B** را که در زیر تعریف شده است به‌عنوان آرایه‌ای متشکل از ۳ آرایهٔ دو بعدی در نظر گرفت.

```
double  B[3][4][5];
```

به شکل ۷-۵ توجه کنید.



شکل ۷- ۵ : آرایهٔ سه‌بعدی به‌عنوان آرایه‌ای از آرایه‌های دو بعدی

مقدار اولیه‌دهی به آرایه‌های چندبعدی

همانند آرایه‌های یک بعدی دو روش برای مقدار اولیه دادن به آرایه‌های چند بعدی وجود دارد.

۱. مقدار دهی از طریق دستور ورودی **cin**:

```
int a[30][5];
for(int i=0;i<30;i++)
  for(int j=0;j<5;j++)
     cin>>a[i][j];
```

۲. مقداردهی اولیه به آرایهٔ چندبعدی در هنگام تعریف:

برای این منظور مطابق روش زیر عمل می‌کنیم:

```
int m[3][2][4]=
{1,2,3,4,5,6,7,8,7,9,3,2,4,6,8,3,7,2,6,3,0,1,9,4};
```

در روش اخیر چنانکه در آرایه‌های یک بعدی نیز دیدید اگر تعداد عناصر ذکر شده در بین دو آکولاد باز و بسته کمتر از تعداد عناصر آرایه باشد. عناصر باقی‌مانده به صفر مقدار اولیه داده می‌شوند.

مثال ۷-۴: برنامه‌ای که۴ نمرهٔ امتحانی ۱۰ دانشجو را خوانده و معدل هر دانشجو و معدل کلاس را در هر درس در نهایت محاسبه و چاپ می‌کند.

```
1. //This program shows usage of multi-dimensional array.
2. #include <iostream>
3. #include <iomanip>
4. using namespace std;
5. int main()
6. {
7.    int score[10][5]={0};
8.    int lesson_sum[4]={0};
9.     for(int i=0;i<3;i++)
10.      {
11.        for(int j=0;j<5;j++)
12.          {
13.             if(j==0)
14.               {
15.                 cout<<"Please enter student number = ";
16.                 cin>>score[i][0];
17.               }
18.             else
19.               {
20.                 cout<<"Enter score number "<<j<<" = ";
21.                 cin>>score[i][j];
22.                 lesson_sum[j-1]+=score[i][j];
23.               }
24.          }
25.        system("cls");
26.      }
27.    system("cls");
```

```
28.         for(i=0;i<10;i++)
29.            {
30.               float sum=0;
31.                 for(int j=1;j<5;j++)
32.                     sum+=score[i][j];
33.               cout<<setw(2)<<(i+1)<<"-Averag of student "
34.                   <<setw(10)<<score[i][0]
35.                   <<" = "<<(sum/4)<<endl;
36.            }
37.         for(i=0;i<4;i++)
38.            cout<<"\nAverage of lesson "<<(i+1)
39.                <<" = "<<((float)lesson_sum[i]/10);
40.       cout<<endl;
41.     return 0;
42. }
```

کار در کلاس ۷-۱ :

برنامه‌ای به زبان **C++** *بنویسید که عناصر دو ماتریس به ترتیب با ابعاد ۵×۴ و ۴×۳ را از کاربر دریافت کرده و ضرب آنها را در ماتریس دیگری با ابعاد ۵×۳ ذخیره و چاپ کند.*

۷-۳: کانتینر vector (بردار)

در C++ امکان استفاده از vector به‌عنوان آرایه‌ای که قابلیت تغییر اندازهٔ آن وجود دارد فراهم می‌باشد. برای استفاده از بردارها سرفایل vector را باید به برنامه اضافه کنید.

برای تعریف یک بردار روش‌های متعددی وجود دارد که عبارت‌اند از:

- vector <نوع مقادیر بردار> نام بردار ;
- vector <نوع مقادیر بردار> نام بردار (ظرفیت اولیه) ;
- vector <نوع مقادیر بردار> نام بردار (مقدار اولیه , ظرفیت اولیه) ;
- vector <نوع مقادیر بردار> نام بردار (اشاره گر انتهای حافظه , اشاره گر ابتدای حافظه) ;

در زیر نمونه‌هایی از تعریف بردارها به هریک از روش‌های فوق را خواهید دید.

```
vector <double>    dub_vec;
```

در نمونه فوق برداری با نام dub_vec تعریف کرده‌ایم که قادر است مقادیری از نوع double را در خود ذخیره کند. اما طول این بردار (یعنی تعداد عناصر vector) برابر صفر می‌باشد. که بعداً قادر هستیم با افزایش ظرفیت این بردار مقادیری از نوع double را در آن ذخیره سازیم.

در زیر، نمونه‌ای مطابق حالت دوم تعریف وکتور را خواهید دید:

```
int n;
cin>>n;
vector <float>  flt_vec(n);
```

در این مثال وکتوری به طول دلخواه n را تعریف کرده‌ایم که طول این بردار توسط کاربر تعیین می‌شود. شاید مهمترین مزیت بردارها نسبت به آرایه‌ها همین قابلیت تعیین ظرفیت آنها باشد. در این حالت تمامی عناصر بردار به صفر مقدار اولیه داده می‌شود.

در زیر نمونه‌ای مطابق حالت سوم تعریف وکتور را خواهید دید:

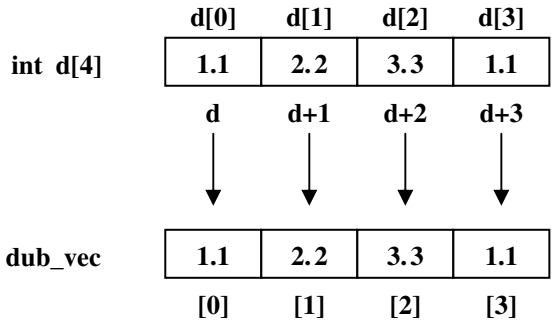
```
int n;
double d;
cin>>n>>d;
vector <double>  dub_vec(n,d);
```

در این مثال نه تنها طول بردار بلکه یک مقدار اولیه نیز برای تمامی عناصر بردار را از کاربر دریافت کنیم. بنابراین در این مثال برداری به نام dub_vec به ظرفیت n تعریف‌شده و مقدار اولیهٔ d به تمامی آنها نسبت داده‌می‌شود.

در زیر نمونه‌ای مطابق حالت چهارم تعریف وکتور را خواهید دید:

```
double d[ ]={1.1,2.2,3.3,4.4};
vector <double> dub_vec(d,d+4);
```

در مثال اخیر ناحیه‌ای از حافظه توسط آرایهٔ d اشغال شده است. از آنجا که d به اولین خانه از حافظهٔ مربوط به آرایه اشاره می‌کند. بنابراین می‌توانیم از آن در تعریف vector استفاده کنیم. در تعریف فوق dub_vec حاوی مقادیر double از بازهٔ [d, d+4) حافظه خواهد بود. بنابراین عناصر dub_vec مطابق شکل ۷-۵ خواهند بود.



شکل ۷- ۶ : انتقال مقادیر از آرایه به وکتور

عملیات برروی وکتورها

عملیات بسیار زیادی را برروی اشیاء **vector** می‌توان انجام داد که در زیر لیست شده‌اند. در زیر توابع عضو اشیاء وکتور و نحوهٔ استفاده از آنها لیست شده است. در این خصوص توجه به دو واژهٔ اساسی اهمیت بسیار دارد. مقصود از ظرفیت یک وکتور حداکثر عناصری است که می‌تواند در خود ذخیره سازد و مقصود از اندازهٔ یک وکتور تعداد عناصری است که در داخل وکتور هم‌اکنون موجود است. به‌عنوان مثال ممکن است یک بردار دارای ظرفیت ۳۰ عنصر باشد ولی در حال حاضر تنها به ۵ عنصر آن مقدار داده باشیم در آن صورت اندازهٔ این وکتور برابر ۵ و ظرفیت آن برابر ۳۰ می‌باشد. برای ادامه مباحث فرض کنید **V2, V 1,V** بردارهایی هستند که از قبل تعریف شده‌اند.

۱. **تابع capacity :** این تابع عدد صحیحی را به‌عنوان ظرفیت بردار برمی‌گرداند، و به شکل کلی زیر به‌کار می‌رود:

```
int cap = V.capacity();
```

در مثال فوق متغیر **cap** ظرفیت وکتور **V** را در خود نگه می‌دارد.

۲. **تابع size :** این تابع اندازهٔ وکتور را به‌صورت یک عدد صحیح برمی‌گرداند، و به شکل کلی زیر به‌کار می‌رود:

```
int sz = V.size();
```

۳. **تابع max_size :** این تابع حداکثر ظرفیتی را که یک وکتور می‌تواند داشته باشد را برمی‌گرداند. این ظرفیت بسته به نوع بردار متفاوت است؛ به‌عنوان مثال اگر وکتور **V** از نوع **int** باشد حداکثر ظرفیت قابل تعریف برای آن برابر **1073741823** عدد **int** می باشد و اگر از نوع **double** باشد حداکثر ظرفیت قابل تعریف برای آن برابر **536870911** خواهد بود. لذا این حداکثر ظرفیت با افزایش طول نوع داده کاهش می یابد درحالی‌که برای نوع **char** که تنها یک بایت را نیاز دارد، حداکثر ظرفیت برابر **4294967295** می‌باشد. نحوهٔ استفاده از این تابع نیز به کلی زیر است:

```
unsigned int  mx_sz = V.max_size();
```

۴. **تابع empty :** اگر وکتور موردنظر خالی باشد این تابع مقدار یک **(true)** را برمی‌گرداند، و در غیر این صورت مقدار صفر را باز می‌گرداند، و به‌صورت کلی زیر به‌کار می‌رود:

```
bool emp = V.empty();
```

به عبارت دیگر این تابع شکل دیگری از عبارت منطقی **(V.size () == 0)** می‌باشد.

۵. **تابع reserve :** به واسطهٔ این تابع می‌توان ظرفیت یک وکتور را افزایش داد. این تابع به شکل کلی زیر به‌کار می‌رود:

```
V.reserve(n);
```

دستور فوق بردار **V** را رشد می‌دهد تا ظرفیت آن به میزان **n** برسد. بهتر است قبل از اجرای این دستور مقدار **n** با حداکثر ظرفیت قابل افزایش برای بردار مذکور مقایسه شود تا از بروز خطای زمان اجرا جلوگیری شود. به‌عنوان مثال می‌توان نوشت:

```
if(n<V.max_size())
   V.reserve(n);
```

البته گاهی اوقات بسته به شرایط حافظهٔ سیستم حتی امکان افزایش ظرفیت یک بردار به میزان **max_size** هم وجود ندارد. پس در رشد دادن بردارها از اعداد غیر معقول استفاده نکنید و یا آنکه باید از پردازش استثناها استفاده کرد که از مباحث فعلی ما خارج است و در آینده به آنها خواهیم پرداخت.

۶. **عملگر [ ] :** جهت دسترسی به هریک از عناصر وکتور می‌توان همانند آرایه‌ها از عملگر کروشه به‌صورت زیر استفاده کرد:

```
V[i]=5;
```

اما عملگر [] همانند آرایه‌ها مرز بردار را چک نمی‌کند. همچنین اندیس **vector** همانند آرایه‌ها از صفر شروع شده و به **V. size ( ) – 1** ختم می‌شود.

۷. **تابع at :** این تابع نیز مانند عملگر [ ] جهت دسترسی به عناصر وکتور طراحی شده است و به شکل کلی زیر استفاده می‌شود:

```
V.at(i)=5;
```

اما این تابع یک تفاوت اساسی با عملگر [ ] دارد و آن، اینکه این تابع مرز بردار را چک می‌کند اگر اندیس موردنظر خارج از حدود وکتور باشد استثنایی را صادر می‌کند. به واسطهٔ استثناها می‌توان بر خطاهای احتمالی زمان اجرا مدیریت کرد و از بروز اشکال در روند اجرای برنامه جلوگیری نمود. در جلد دوم به تفصیل در خصوص مدیریت استثناها[1] صحبت خواهیم کرد اما فعلاً تنها به جهت آشنایی با کار کرد این تابع در مدیریت استثناها نحوهٔ به‌کارگیری آن را در قطعه کد زیر نشان می‌دهیم. لذا پس از آنکه مبحث مدیریت استثناها را خواندید به نکات مبهم این کد واقف خواهید شد.

---

1. exception handling

چنانکه پیشتر دیدید می‌توان برای دریافت عناصر بردار چنین عمل کرد:

```
int size;
cin>>size;
vector <int> int_vec(size);
for(int i=0; i<=size; i++);  //mistake
   cin>>int_vec[i];
```

چنانکه می‌بینید در کد فوق، شرط حلقه موجب شده به اشتباه اندیس حلقه از حدود بردار تجاوز کند. لذا در هنگام اجرای حلقه این امکان وجود دارد که اطلاعات موجود در متغیرهای دیگری از برنامه دستکاری شود. برای جلوگیری از این خطای منطقی با استفاده از مدیریت استثناها به شیوهٔ زیر عمل می‌کنیم:

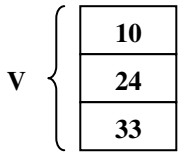
```
try
  {
     for(int i=0; i<V.size(); i++)
        cin>>V.at(i);
  }
catch(out_of_range)
    {
        cerr<<"index out of range";
    }
```

۸. **تابع push_back :** این تابع یک مقدار را در انتهای وکتور اضافه می‌کند و به‌صورت کلی زیر به‌کار می‌رود:

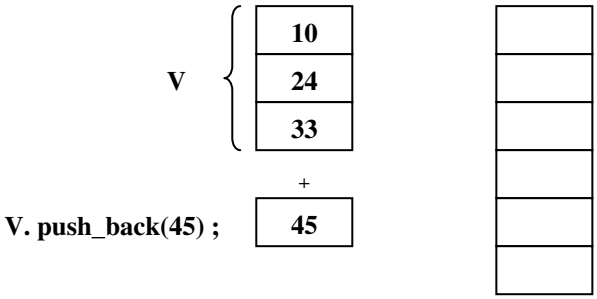
```
V.push_back(value);
```

در دستور فوق مقدار **value** به انتهای بردار **V** اضافه می‌شود و مقدار **size** به میزان یک واحد افزایش می یابد درحالی‌که اگر به واسطهٔ عملگر [ ] عنصر جدیدی به آرایه اضافه کنیم مقدار **size** اصلاح نمی‌شود. نکتهٔ قابل توجه در خصوص این تابع آنکه اگر بردار ظرفیت آزادی برای اضافه شدن عنصر نداشته باشد ابتدا ظرفیت بردار را دو برابر می‌کند و سپس عنصر جدید را به انتهای بردار اضافه می‌کند. لذا بهتر است در هنگام خواندن عناصر بردار از ورودی از این تابع استفاده کنید تا در صورت کمبود حافظهٔ آزاد در بردار، برنامه با مشکل بر نخورد. اما از طرفی در هنگام استفاده از این تابع باید بسیار محتاط و مراقب باشید که بر سرعت اجرای برنامه تأثیر منفی نگذارید. چرا که عمل افزایش ظرفیت بردار یک عمل وقت‌گیر می‌باشد. برای درک علت وقت‌گیر بودن عملیات افزایش حافظهٔ وکتور به مثال زیر توجه کنید.

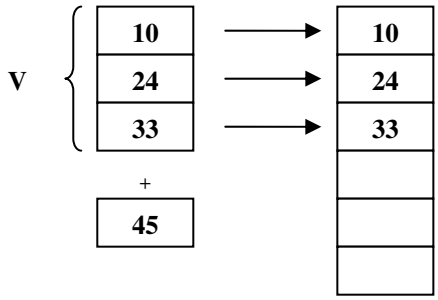
فرض کنید برداری با ظرفیت و اندازهٔ ۳ مطابق شکل زیر داریم:



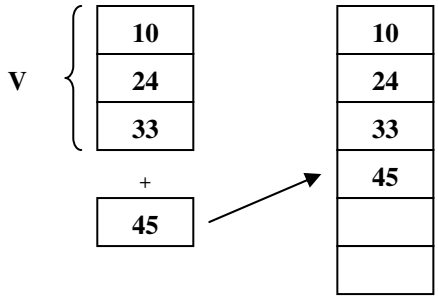
حال می‌خواهیم عنصر دیگری برابر ۴۵ را به انتهای آن توسط تابع **push_back** بیفزاییم. برای این منظور ابتدا بلوکی به اندازهٔ دو برابر ظرفیت وکتور قبلی تشکیل داده می‌شود.



سپس عناصر وکتور قبلی مطابق شکل زیر به ابتدای بلوک جدید انتقال می‌یابد.



پس از آن در مرحلهٔ سوم عنصر جدید نیز به انتهای وکتور جدید انتقال می‌یابد.



و در نهایت وکتور قبلی از بین می‌رود و از این پس **V** به ابتدای بلوک جدید اشاره می‌کند. و حافظه‌ای که از بین رفتن وکتور قبلی، به‌دست می‌آید به مدیریت حافظهٔ سیستم عامل برگردانده می‌شود. با توجه به توضیحاتی که ارائه شد باید به‌راحتی متوجه این مطلب شده باشید که افزایش ظرفیت وکتور به این شیوه در زمانی‌که وکتور ابتدایی، دارای ظرفیت بالایی باشد چقدر بر سرعت برنامه امکان دارد تأثیر منفی بگذارد.

۹. **تابع pop_back :** این تابع موجب می‌شود که آخرین عنصر بردار حذف گردد. نکتهٔ قابل توجه آنکه اندازهٔ بردار به میزان یک واحد کاهش می‌یابد اما ظرفیت بردار ثابت می‌ماند. شکل کلی استفاده از این تابع به‌صورت زیر است:

```
V.pop_back();
```

هنگام استفاده از این تابع باید مراقب باشید که این تابع برای برداری با اندازه صفر که حاوی هیچ عنصری نیست صدا زده نشود، چون برنامه با مشکل مواجه خواهد شد. بنابراین بهتر است بنویسیم:

```
if(!V.empty())
  V.pop_back();
```

۱۰. **توابع back , front :** تابع **front** مرجعی را به اولین عنصر وکتور برمی‌گرداند. و تابع **back** نیز مرجعی به آخرین عنصر وکتور که تابع عضو **end** مشخص می‌کند برمی‌گرداند. البته مفهوم مرجع را در فصل نهم مطالعه خواهید دید، اما نمونه‌ای از کاربرد این توابع به‌صورت زیر است:

```
vector <double> V;
V.push_back(1.1);
V.push_back(2.2);
V.push_back(3.3);
cout<<V.front()<<"\t"<<V.back();
```

خروجی این قطعه کد به‌صورت زیر خواهد بود:

```
1.1        3.3
```

اما اگر در ادامهٔ این کد بنویسیم:

```
V[3]=4.4;
cout<<"\nsize = "<<V.size();
cout<<"\n"<<V.front()<<"\t"<<V.back();
```

خروجی زیر حاصل خواهد شد:

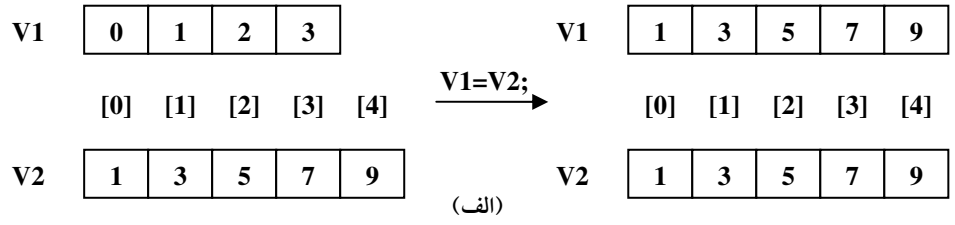
```
Size = 4
1.1        3.3
```

با آنکه آخرین عنصر وارد شده **4.4** می‌باشد ولی **3.3** به‌عنوان آخرین عنصر چاپ می‌شود، زیرا همان‌طور که پیشتر گفتیم عملگر [ ] تکرارگری که توسط **end** برگردانده می‌شود را اصلاح نمی‌کند.

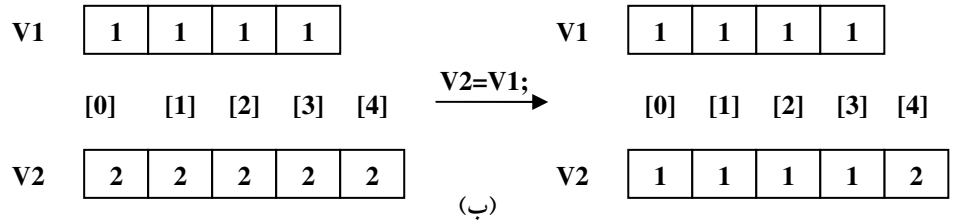
۱۱. **عملگر = :** این عملگر به‌صورت کلی به‌کار می‌رود:

```
V1=V2;
```

این عملگر موجب می‌شود که یک کپی از **V2** در **V1** قرار بگیرد. مثال‌هایی را که در ادامه آمده است مطالعه فرمایید.



(الف)



(ب)

مطابق نمونهٔ (الف) در بالا اگر طول بردار مقصد از بردار مبدأ کوچکتر باشد، ابتدا طول آن به اندازهٔ بردار مبدأ شده و سپس مقادیر آرایهٔ مبدأ در آن کپی می‌گردد.

۱۲.**عملگر = =:** این عملگر به‌صورت کلی زیر به کار می‌رود:

```
V1==V2;
```

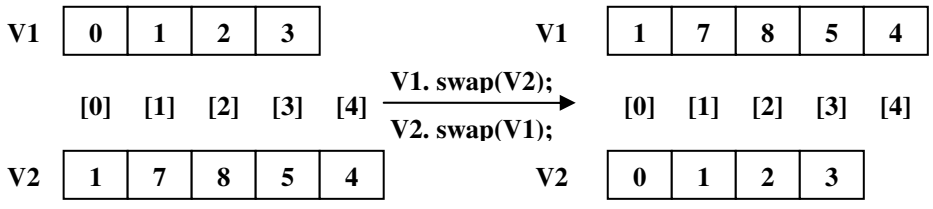
با اجرای این رابطهٔ منطقی اگر **m** مقدار ابتدای **V1** با **m** مقدار ابتدای **V2** برابر باشد آنگاه مقدار **true** و درغیراین صورت مقدار **false** برگردانده می‌شود و مقدار **m** برابر مینیمم اندازهٔ **V1** و **V2** می‌باشد:

```
m = min{V1.size(),V2.size()}
```

۱۳.**تابع swap:** این تابع موجب می‌شود که محتویات دو بردار با هم عوض شود. این تابع به‌صورت کلی زیر به‌کار می‌رود:

```
V1.swap(V2);
```

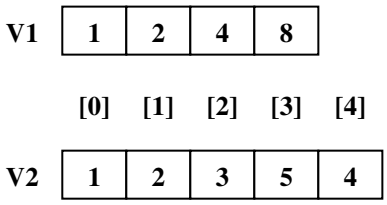
به‌عنوان مثال بردارهای زیر و عملیات **swap** صورت گرفته برروی آنها را در نظر بگیرید:



۱۴.**عملگر < :** این عملگر به‌صورت کلی زیر به‌کار می‌رود:

```
V1<V2
```

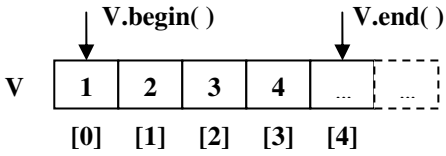
این رابطهٔ منطقی اگر **V1** کوچکتر از **V2** باشد مقدار **true** و در غیر این صورت مقدار **false** برگردانده می‌شود. نحوهٔ عملکرد این عملگر به اینگونه است که عنصر به عنصر عمل مقایسه بین عناصر **V1** و **V2** را انجام می‌دهد تا زمانی که به اولین مورد متفاوت برسد. اگر عنصر مورد اختلاف مربوط به **V1** کوچکتر از عنصر مورد اختلاف مربوط به **V2** باشد مقدار **true** و در غیر این صورت مقدار **false** را برمی‌گرداند. به‌عنوان مثال دو بردار زیر را در نظر بگیرید:



هنگامی که کنترل اجرای برنامه به دستور **V1 < V2** می‌رسد عملیات مقایسه را عنصر به عنصر آغاز می‌کند تا آنکه در عناصر **V1[2] , V2[2]** اولین مورد اختلاف دیده شود و با توجه به آنکه **V1[2] > V2[2]** می‌باشد، مقدار **false** به ازای رابطهٔ **V1 <V2** برگردانده می‌شود.

۱۵.**توابع begin و end :** تابع **begin** یک تکرارگر[۱](اشاره‌گر) به اولین عنصر بردار برمی‌گرداند و تابع **end** نیز یک تکرارگر به بعد از آخرین عنصر بردار را برمی‌گرداند. این توابع به‌صورت کلی زیر به‌کار می‌روند:

```
vector <نوع مقادیر بردار>::iterator  itb, ite;
itb = V.begin();
ite = V.end();
```



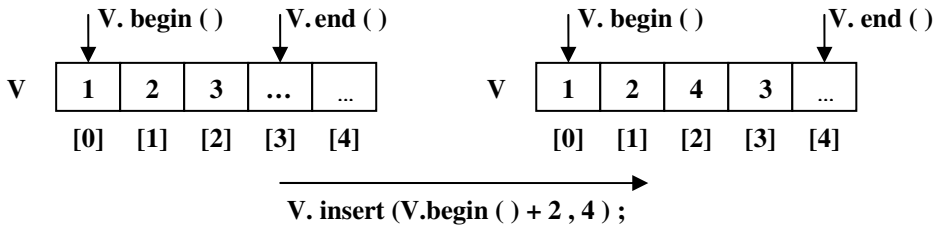
۱۶.**تابع insert :** این تابع به جهت درج مقادیر در بین عناصر یک بردار به‌کار می‌رود. به‌صورت‌های کلی زیر به‌کار می‌رود:

```
V.insert(pos,value);
V.insert(pos,n,value);
```

---

۱. در C++ استاندارد تکرارگرها به‌صورت زیر تعریف شده است: «تکرارگرها، شکل کلی اشاره‌گرها هستند که موجب می‌شوند برنامهٔ C++ بتواند با ساختمان داده‌های مختلف به شکل یکنواخت کار کنند.

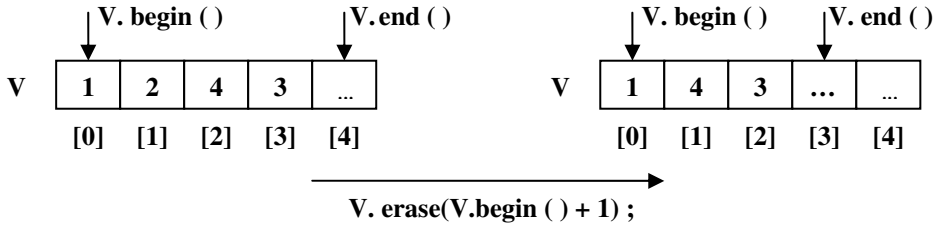
در حالت اول مقدار **value** را در مکانی که تکرارگر **pos** به آن اشاره می‌کند درج می‌نماید و در حالت دوم **n** کپی از مقدار **value** را در موقعیتی که تکرارگر **pos** به آن اشاره می‌کند درج می‌نماید. به‌عنوان مثال شکل زیر را ببینید.



۱۷.**تابع erase :** این تابع به جهت حذف مقادیر از یک بردار کاربرد دارد، و دارای دو صورت کلی زیر است:

```
V.erase(pos);
V.erase(pos1,pos2);
```

در صورت کلی نخست عنصری را که در مکانی که تکرارگر **pos** به آن اشاره می‌کند را حذف می‌کند و در صورت کلی دوم مقادیری از **V** را از موقعیتی که تکرارگر **pos1** تا **pos2** قرار دارند را حذف می‌کند. به‌عنوان مثال شکل زیر را ببینید:



مثال ۷-۵ : برای جمع اعدادی که در محدودهٔ قابل ذخیره در ۳۲ اعداد **int** نباشند باید به‌صورت نرم‌افزاری عمل کرد. در زیر برنامه‌ای ارائه شده که با استفاده از وکتورهای یک بعدی دو عدد بسیار بزرگ **(Huge Integer)** را با یکدیگر جمع می‌کند.

```
1.  //This program gives the sum of two huge integer numbers.
2.  #include <iostream>
3.  #include <conio.h>
4.  #include <vector>
5.  using namespace std;
6.  int main()
7.  {
8.      vector <char> num1,num2,result;
9.      cout<<"Enter tow huge integer numbers to get the sum
10.            of them.\n";
```

```
11.       cout<<"Enter number one and press enter key.\n\n";
12.       char ch;
13.       while((ch=getch())!=13)
14.        {
15.            if(ch>='0'&& ch<='9')
16.              {
17.                   cout.put(ch);
18.                   ch-=48;
19.                   num1.push_back(ch);
20.              }
21.        }
22.       cout<<"\n+"<<endl;
23.       while((ch=getch())!=13)
24.        {
25.            if(ch>='0'&& ch<='9')
26.              {
27.                   cout.put(ch);
28.                   ch-=48;
29.                   num2.push_back(ch);
30.              }
31.        }
32.      cout<<"\n="<<endl;
33.      int min_size;
34.      min_size = num1.size() < num2.size() ?
35.                 num1.size() : num2.size();
36.      bool carry=0;
37.      while(min_size>0)
38.       {
39.         if(num1.back()+num2.back()+carry<10)
40.          {
41.           result.push_back(num1.back()+num2.back()+carry);
42.           num1.pop_back();
43.           num2.pop_back();
44.           carry=0;
45.          }
46.         else
47.          {
48.      result.push_back((num1.back()+num2.back()+carry)%10);
49.           num1.pop_back();
50.           num2.pop_back();
51.           carry=1;
52.          }
53.        min_size--;
54.       }
55.     while(num1.size()!=0)
56.      {
57.        if(num1.back()+carry<10)
58.         {
```

```
59.             result.push_back(num1.back()+carry);
60.             num1.pop_back();
61.             carry=0;
62.           }
63.         else
64.           {
65.             result.push_back((num1.back()+carry)%10);
66.             num1.pop_back();
67.             carry=1;
68.           }
69.       }
70.     while(num2.size()!=0)
71.       {
72.         if(num2.back()+carry<10)
73.           {
74.             result.push_back(num2.back()+carry);
75.             num2.pop_back();
76.             carry=0;
77.           }
78.         else
79.           {
80.             result.push_back((num2.back()+carry)%10);
81.             num2.pop_back();
82.             carry=1;
83.           }
84.        }
85.      if(carry==1)
86.         result.push_back(1);
87.      while(result.size()>0)
88.       {
89.           cout.put(result.back()+48);
90.           result.pop_back();
91.       }
92.      cout<<endl;
93.     return 0;
94.  }
```

توضیح مثال: در این مثال از سه بردار با نام‌های **num1** و **num2** و **result** برای ذخیرهٔ اعداد اول و دوم و سوم استفاده شده است. در دو حلقهٔ اول تک‌تک ارقام دو عدد را دریافت و به انتهای وکتور افزوده‌ایم. درحلقهٔ تکرار خط ۳۷ تا زمانی که دو عدد دارای ارقام معادل هستند عملیات جمع را پیش برده‌ایم. عمل جمع براساس به‌وجود آمدن و یا عدم به‌وجود آمدن رقم نقلی انجام می‌گیرد. به‌طوری که اگر رقم نقلی به وجود نیاید دستورات بلوک **if** و اگر به‌وجود آید دستورات بلوک **else** انجام می‌گیرد. از متغیر **carry** به جهت ذخیرهٔ رقم نقلی استفاده شده است. پس از آن یکی از حلقه‌های خطوط ۵۵ یا ۷۰ اجرا شده و باقی ماندهٔ ارقام عددی را که دارای طول بیشتری است را به حاصل جمع می‌افزاییم. پس از آن در حلقهٔ خط ۸۷ حاصل‌جمع را چاپ نموده‌ایم.

۷-۴ : وکتورهای چند بعدی

در بخش ۷-۲ دیدید که آرایه‌های دوبعدی را به‌صورت آرایه‌ای از آرایه‌های یک بعدی تصویر می‌کردیم. به عبارت دیگر آرایهٔ دوبعدی را به‌عنوان آرایه‌ای یک بعدی که هریک از عناصر آن خود یک آرایهٔ یک‌بعدی است، در نظر گرفتیم. به همین روش می‌توان وکتورهای دوبعدی، سه‌بعدی و ... را در نظر گرفت و تعریف کرد. برای تعریف یک بردار دوبعدی با ۳ سطر و ۴ ستون ابتدا یک بردار یک‌بعدی با ۴ ستون تشکیل می‌دهیم و سپس بردار دیگری با ۳ عنصر که هر یک از عناصر آن یک بردار با ۴ ستون باشد تشکیل می‌دهیم:

```
int Row=3,col=4;
vector <vector <int> > table(Row, vector <int> (col,0));
```

به این ترتیب یک بردار دوبعدی به‌صورت زیر تعریف می‌گردد:

		[0]	[1]	[2]	[3]
table	[0]	0	0	0	0
	[1]	0	0	0	0
	[2]	0	0	0	0

عملیات بر روی بردار دوبعدی

۱. **عملگر [ ] :** به واسطهٔ این عملگر می‌توان به عنصری که در ردیف **r** و ستون **c** قرار دارد به‌صورت زیر دسترس پیدا کرد:

```
table [r][c]=5;
```

۲. **تابع size :** فرض کنید می‌خواهید تعداد سطرهای **vector** دو بعدی را تعیین کنید، برای این منظور اگر بردار دو بعدی **table** را با ابعاد ۳×۴ را که قبلاً تعریف کردیم در نظر بگیرید عبارت:

```
int R=table.size();
```

مقدار ۳ را در متغیر **R** ذخیره می‌کند. همچنین می‌توان تعداد ستون‌های هر سطر را به‌واسطهٔ دستور زیر تعیین کرد که **r** شمارهٔ ردیف موردنظر است:

```
int C=table[r].size();
```

می‌توان به‌واسطهٔ تابع **size** و عملگر [ ] عملیات زیادی را برروی وکتورها انجام داد به‌عنوان مثال قطعه کد زیر موجب نمایش محتویات برداری به نام **table** می‌شود.

```
for(int row=0; row<table.size(); row++)
{
    for(int col=0; col<table[row].size(); col++)
      cout<<table[row][col]<<"\t";
    cout<<endl;
}
```

۳. **تابع push_back :** به واسطهٔ این تابع می‌توان یک ردیف جدید یا یک ستون جدید به بردار اضافه کرد. جهت افزودن یک ردیف به برداری با نام **table** به‌صورت زیر عمل می‌کنیم:

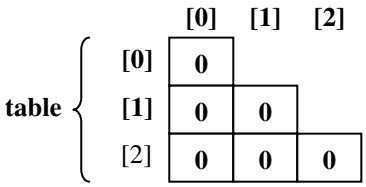
```
table.push_back(vector <int> (col, 0));
```

همچنین برای افزودن یک ستون به بردار دو بعدی **table** می‌توان به تک تک ردیف‌ها یک عنصر اضافه کرد:

```
for(int row=0; row<table.size(); row++)
   table[row].push_back(0) ;
```

بردارهای دندانه‌ای

آرایه ها و بردارهایی که تا اینجا دیدید همگی به‌صورت مربع مستطیل بودند، به عبارت دیگر تعداد ستون‌های همهٔ ردیف‌ها با هم برابر بود. اما می‌توان آرایه‌ها و بردارهایی به‌صورت آنچه در شکل زیر نشان داده شده تشکیل داد. به این آرایه‌ها (یا بردارها)، آرایه‌های دندانه‌ای(یا بردارهای دندانه‌ای) گفته می‌شود.



گرچه در C++ امکان تعریف آرایه‌های دندانه‌ای به‌صورت معمول وجود ندارد و حتماً باید توسط اشاره‌گرها آنها را تعریف‌کرد اما تعریف بردارهای دندانه‌ای به‌راحتی مانند آنچه در کد زیر نشان داده شده است امکانپذیر می‌باشد.

```
vector <vector <double> > aTable;
  for(int col=1; col<=3; col++)
      aTable.push_back(vector <double> (col, 0.0));
```

مثال ۷-۶: برنامه‌ای که ضرب دو ماتریس با ابعاد $m \times n$ و $n \times p$ را پیاده‌سازی می‌کند.

```
1.  //This program calculates the product of 2 matrix.
2.  #include <iostream>
3.  #include <vector>
4.  using namespace std;
5.  int main()
6.  {
7.      int m,n,p;
8.     cout<<"Enter size of tow integer matrix to get
9.           the product of them.\n";
10.     cout<<"Enter size of first matrix A(m*n).\n";
11.      cout<<"Enter m = ";
12.       cin>>m;
13.        cout<<"Enter n = ";
14.         cin>>n;
15.       vector <vector <int> > A(m, vector <int> (n,0));
16.        cout<<"\nEnter size of second matrix B("
17.            <<n<<"*p) .\n";
18.         cout<<"Enter p = ";
19.          cin>>p;
20.       vector <vector <int> > B(n, vector <int> (p,0));
21.         system("cls");
```

```
22.      cout<<"Enter Matrix A members:\n";
23.      for(int i=0;i<m;i++)
24.         for(int j=0;j<n;j++)
25.            {
26.              cout<<"Enter A["<<(i+1)<<"]["<<(j+1)<<"] =";
27.               cin>>A[i][j];
28.            }
29.     cout<<"\nEnter Matrix B members:\n";
30.      for(i=0;i<n;i++)
31.         for(int j=0;j<p;j++)
32.            {
33.              cout<<"Enter B["<<(i+1)<<"]["<<(j+1)<<"] =";
34.               cin>>B[i][j];
35.            }
36.          vector <vector <int> > C(m, vector <int> (p,0));
37.           for(i=0;i<m;i++)
38.             for(int k=0;k<p;k++)
39.               for(int j=0,sum=0;j<n;j++)
40.                 C[i][k]+=A[i][j]*B[j][k];
41.        system("cls");
42.          for(i=0;i<m;i++)
43.             {
44.                 for(int j=0;j<p;j++)
45.                    cout<<C[i][j]<<"\t";
46.                  cout<<endl;
47.              }
48.     return 0;
49. }
```

توضیح مثال: درخطوط ۹ الی ۲۲ این برنامه ابتدا ابعاد ماتریس‌ها را دریافت‌کرده و بردارهای موردنظر راساخته‌ایم. سپس در خطوط ۲۴ الی ۲۹ عناصر ماتریس اول و در خطوط ۳۱ الی ۶۶ عناصر ماتریس دوم را دریافت کرده‌ایم. در خط ۳۷ یک بردار دو بعدی جدید برای ذخیرهٔ عناصر ماتریس حاصل‌ضرب ساخته‌ایم. و در خطوط ۳۸ الی ۴۰ عملیات ضرب دو ماتریس را انجام داده‌ایم. و در نهایت در خطوط ۴۳ الی ۴۸ عناصر ماتریس حاصل ضرب را چاپ کرده‌ایم.

مقایسهٔ آرایه و vector

۱. آرایه در طول اجرای برنامه دارای ظرفیت ثابتی است اما ظرفیت بردارها قابل تغییر است.

۲. درج یا حذف عنصر در انتهای آرایه یا **vector** به شرط وجود ظرفیت خالی دارای زمان ثابتی است، اما درج یا حذف رکورد در میانهٔ آرایه یا **vector** در بدترین حالت نیازمند **n** جابه‌جایی و در حالت متوسط نیازمند **n/2** جابه جایی داده‌ها است، که **n** برابر اندازهٔ وکتور یا آرایه می‌باشد.

نتیجه آنکه بردارها و آرایه‌ها، تنها برای ذخیره کردن دنباله‌های که در آنها اعمال درج و حذف زیاد انجام نمی‌شود و یا فقط در انتهای لیست این اعمال انجام می‌شوند، مفید هستند.

۷-۵: رشته در زبان C

به هرمجموعه از کاراکترهای اسکی رشته گفته‌می‌شود. پیشتردیدید که چگونه رشته‌ای را در خروجی می‌نوشتیم. به‌عنوان مثال دستور زیر:

```
cout<<"Hello, World";
```

موجب می‌شد تا کاراکترهای بین دو علامت کوتیشن چاپ گردد. اما در اینجا می‌خواهیم نحوهٔ ذخیرهٔ اطلاعات رشته‌ای مثل نام، نام‌خانوادگی، محل تولد و ... که توسط کاربر می‌تواند وارد برنامه شود را بیاموزیم.

در C++، رشته نوع جدیدی محسوب نمی‌شود، بلکه یک رشته به‌عنوان آرایه‌ای که مقادیر **char** را در خود ذخیره می‌سازد در نظر گرفته می‌شود. برای ذخیرهٔ یک رشته باید ابتدا آرایه‌ای از کاراکترها را تعریف کنیم:

```
char str[50];
```

پیشتر دیدید که در متغیری از نوع کاراکتر مثل **ch** می‌توانستیم کد اسکی علائم و حروف را ذخیره کنیم. به‌عنوان مثال می‌توان نوشت:

```
ch='A';
```

دستور فوق موجب ذخیرهٔ کداسکی حرف **A** در متغیر **ch** می‌شود. رشته نیز چیزی جز مجموعه‌ای از کاراکترها نیست. بنابراین با تعریف آرایه‌ای از نوع **char** می‌توان کد اسکی تک تک حروف یک جمله (رشته) را در خانه‌های آن آرایه ذخیره‌سازی کرد. اما نکتهٔ مهم در این خصوص این است که، پس از آخرین کاراکتر رشته باید، یک کاراکتر خاص به نام تهی یا **NULL** ذخیره گردد تا انتهای رشته را مشخص سازد. این کاراکتر معادل کد اسکی صفر بوده و به‌صورت **'\0'** نشان داده می‌شود. به‌عنوان مثال برای ذخیرهٔ کلمهٔ **ALI** در آرایهٔ کاراکتری **S** به‌صورت زیر عمل می‌شود:

```
char S[10]="ALI";
```

S[10]	'A'	'L'	'I'	'\0'	?	?	?	?	?	?

هنگامی که می‌نویسیم:

```
cout<<S;
```

کلمهٔ **ALI** در خروجی چاپ می‌شود، در حقیقت چاپ کاراکترهای داخل آرایهٔ **S** از ابتدای آرایه آغاز شده و تا جایی که به علامت **NULL** برسد ادامه پیدا می‌کند. حال اگر این علامت در انتهای رشته قرار نگیرد ممکن است اطلاعات خانه‌های بعدی حافظه (که نمی‌دانیم چه چیزی در آنها قرار دارد) نیز چاپ شود. لذا همواره باید طول آرایهٔ کاراکتری یکی بیشتر از طول رشتهٔ موردنظر باشد. تا پس از آخرین کاراکتر رشته علامت **NULL** قرار داده شود.

مقداردهی اولیه به رشته‌ها

برای مقدار اولیه دادن به رشته‌ها نیز روش‌های متفاوتی وجود دارد، که در زیر نمونه‌های متفاوتی از آن را می‌بینید.

```
char S1[9] = "computer";
```

S1[9]	'c'	'o'	'm'	'p'	'u'	't'	'e'	'r'	'\0'

```
char S2[ ] = "language";
```

S2[9]	'l'	'a'	'n'	'g'	'u'	'a'	'g'	'e'	'\0'

```
char S3[9] = "input" ;
```

S3[9]	'i'	'n'	'p'	'u'	't'	'\0'	?	?	?

```
char S4[ ] = {'p','r','o','g','r','a','m','\0'};
```

S4[8]	'p'	'r'	'o'	'g'	'r'	'a'	'm'	'\0'

خواندن رشته‌ها از ورودی

۱. **خواندن رشته توسط شئ cin :** توسط این شئ می‌توان یک رشته که حاوی کاراکتر فضای خالی (space) نباشد را از کاربر دریافت کرد. به‌عنوان مثال:

```
char str[20] ;
cin>>str;
cout<<"string = "<<str;
```

در مثال فوق یک رشته از کاربر دریافت می‌شود طول رشته حداکثر باید ۱۹ کاراکتر باشد تا در بیستمین خانهٔ آرایهٔ str مقدار NULL قرار بگیرد. اما اگر رشته‌ای با طول بزرگتر از ۲۰ کاراکتر را کاربر وارد کند که شامل هیچ فضای خالی نیز نباشد. شئ cin ذخیرهٔ اطلاعات را در خارج از محدودهٔ آرایهٔ str نیز ادامه می‌دهد که در این صورت امکان از بین رفتن اطلاعاتی که پس از این آرایه قرار دارد، نیز وجود دارد.

۲. **خواندن رشته توسط تابع get عضو شئ cin :** جهت جلوگیری از مشکل مطرح شده در خصوص دریافت رشته توسط شئ cin بهتر است از تابع get که از توابع عضو شئ cin می‌باشد استفاده کنید. شکل کلی به‌کارگیری این تابع به‌صورت‌های زیر است:

```
cin.get (نام رشته , حداکثر طول رشته) ;
cin.get (نام رشته , حداکثر طول رشته , 'جدا کننده') ;
```

در حالت اول نام آرایهٔ کاراکتری (یعنی محل ذخیرهٔ رشته) و سپس حداکثر طول رشته را ذکر می‌کنیم. حداکثر طول رشته باید یکی کمتر از طول آرایهٔ کاراکتری باشد. به‌عنوان مثال کد زیر را در نظر بگیرید:

```
char S[20];
cin.get(S,15);
```

در این مثال برنامه شروع به دریافت رشته از کاربر می‌کند. تا زمانی که کاربر کلید **enter** را بزند. سپس کاراکترهایی از ابتدای رشتهٔ وارد شده، که حداکثر به طول **15** کاراکتر باشد را در آرایهٔ کاراکتری **S** قرار می‌دهد.

در کاربرد دوم نه تنها حداکثر طول رشته، بلکه یک کاراکتر به‌عنوان جدا کننده (**delimiter**) نیز به‌عنوان ورودی تابع **get** ذکر می‌شود تا اگر کاربر چنین کاراکتری را وارد کرد دریافت داده‌ها متوقف گردد. به‌عنوان مثال در کد زیر:

```
char S[20];
cin.get(S,19,'.');
```

در این کاربرد تابع **get** رشته‌ای به طول حداکثر ۱۹ کاراکتر یا تا کاراکتری که به کاراکتر جدا کننده یعنی '.' برسد، را تا زمانی که کاربر کلید **enter** را بزند از ورودی دریافت می‌کند و در رشتهٔ **S** قرار می‌دهد.

حسن بسیار مهمی که استفاده از تابع **get** نسبت به استفاده از شئ **cin** به همراه عملگر << دارد این است که تابع **get** کاراکتر فضای خالی را نیز قادر است بخواند.

مثال ۷-۷: برنامه‌ای که یک رشته را در رشته‌ای دیگر کپی می‌کند.

```
1.  //This program copies one string to another.
2.  #include <iostream>
3.  using namespace std;
4.  int main()
5.  {
6.     char s1[2000],s2[2000];
7.      cout<<"Please enter a sentence:\n";
8.       cin.get(s1,100,'.');
9.        for(int i=0; i<2000 && s1[i]!='\0'; i++)
10.              s2[i]=s1[i];
11.       s2[i]='\0';
12.        system("cls");
13.         cout<<"You entered :\n";
14.           cout<<s2<<endl;
15.    return 0;
16. }
```

توضیح مثال : توجه داشته باشید که به واسطهٔ عملگر انتساب نمی‌توان یک رشته را به رشته‌ای دیگر نسبت داد، زیرا رشته‌هایی که تا کنون دیدید به‌صورت آرایه‌ای از کاراکترها هستند. به این دسته از رشته‌ها رشته‌های زبان **C** نیز گفته می‌شود. بنابراین اگر دستوری مانند دستور **s2=s1;** به‌منظور نسبت دادن رشتهٔ **s1** در رشتهٔ **s2** بنویسید با خطای زیر برخورد خواهید شد.

error C2106: '=' : left operand must be l-value

چنانکه در این مثال می‌بینید در خطوط ۹ و ۱۰ در یک حلقهٔ تکرار عناصر رشتهٔ اول را به عناصر رشتهٔ دوم نسبت داده‌ایم.

کار در کلاس ۷-۲ :

برنامه‌ای به زبان **C++** *بنویسید که ضمن دریافت دو رشته، آنها را بر حسب حروف الفبا با یکدیگر مقایسه کند و برحسب این که رشتهٔ اول نسبت به رشتهٔ دوم کوچکتر یا مساوی و یا بزرگتر باشد به ترتیب یکی از علائم* **>** *و یا* **=** *و یا* **<** *را چاپ کند. مراقب باشید که برای عملیات مقایسهٔ دو رشته نمی توان نوشت* **S1<S2** *، بلکه باید عملیات مقایسه را کاراکتر به کاراکتر انجام داد.*

مثال ۷-۸: برنامه‌ای که ضمن دریافت یک متن ۲۰۰۰ کاراکتری و یک زیر رشته از ورودی، زیر رشته را در متن جستجو کرده، در صورت یافتن زیر رشته در متن، مکان زیر رشته در متن چاپ می‌شود و در غیر این صورت پیغام "Not found!" چاپ خواهد شد.

```
1.  //This Program searchs a pattern in a text.
2.  #include <iostream>
3.  using namespace std;
4.  int main()
5.  {
6.    const int Max_Length=2001;
7.     char text[Max_Length];
8.      cout<<"Enter a text with max 2000 characters.\n";
9.       char ch;
10.       cin.get(ch);
11.        int i=0;
12.       while(ch!='\n'&&i<2000)
13.           {                    //read the text
14.              text[i++]=ch;
15.              cin.get(ch);
16.           }
17.       text[i]='\0';
18.  int text_length=i;
19.    cout<<"\n\nEnter your pattern for search : ";
20.     char pattern[101];
21.      int j=0;
22.       cin.get(ch);
23.        while(ch!='\n'&&j<100)
24.            {                  //read the pattern
25.               pattern[j++]=ch;
26.               cin.get(ch);
27.            }
28.        pattern[j]='\0';
29.    int pattern_length=j;
30.    int index=0;
31.     i=j=0;
32.      while(i<text_length && j<pattern_length)
33.          {    if(text[i]==pattern[j])
34.                 {               //continue matching
35.                     i++; j++;
36.                 }
37.               else              //backtrack and start over
38.                 {               //at next position of
39.                     index++;    //current string (text)
40.                     i=index;
41.                     j=0;
42.                 }
43.           }//end of while
```

```
44.      if(j==pattern_length)
45.        cout<<"The pattern found at character "<<(index+1);
46.      else
47.        cout<<"Not found!";
48.     cout<<endl;
49.    return 0;
50.  }
```

آرایه‌ای از رشته‌ها

از آنجا که رشته‌ها به‌صورت آرایه‌ای از کاراکترها در C++ پیاده‌سازی می‌شوند به‌راحتی می‌توان آرایه‌ای از رشته‌ها را نیز تشکیل داد و در یک ساختار جدول مانند رشته‌های متفاوتی را ذخیره کرد. جهت درک بهتر مطلب به مثال زیر توجه کنید.

مثال ۷-۹: برنامه‌ای که نام روزهای هفته را در یک آرایه قرار داده و آنها را چاپ می‌کند.

```
1.  //This program uses array of string.
2.  #include <iostream>
3.  using namespace std;
4.  int main()
5.  {
6.   const int Days=7;
7.    const int Max_Length=10;
8.     char str_array[Days][Max_Length] = {"Sunday","Monday",
9.     "Tuesday","WednesDay","Thursday","Friday","Saturday"};
10.     for(int i=0;i<Days;i++)
11.         cout<<str_array[i]<<endl;
12.   return 0;
13.  }
```

توضیح مثال: در واقع آنچه را که در خصوص آرایه‌های دو بعدی تصور کردیم را می توان در خصوص آرایهٔ str_array نیز متصور شد که در شکل ۷-۷ نشان داده شده است. در این مثال ثابت Days حاوی تعداد رشته‌های مورد نیاز و ثابت Max_Length حاوی حداکثر طول در رشته‌های مورد نظر است.

	0	1	2	3	4	5	6	7	8	9
0	S	u	n	d	a	y	ø	?	?	?
1	M	o	n	d	a	y	ø	?	?	?
2	T	u	e	s	d	a	y	ø	?	?
3	W	e	n	d	n	s	d	a	y	ø
4	T	h	u	r	s	d	a	y	ø	?
5	F	r	i	d	a	y	ø	?	?	?
6	S	a	t	u	r	d	a	y	ø	?

شکل ۷- ۷ : آرایه‌ای از رشته‌ها

۷-۶ : سرفایل(کلاس) string زبان C++ استاندارد

رشته‌هایی که تا اینجا برروی آنها بحث کردیم به رشته‌های زبان C معروف هستند، چرا که از زبان C به زبان C++ انتقال یافته‌اند. و چنانکه پیشتر در مثال ۷-۷ دیدید عملیات کپی کردن یک رشته از این نوع در رشته‌ای دیگر از همان نوع به واسطهٔ دستور انتساب امکانپذیر نیست و باید عناصر رشتهٔ مبدأ را عنصر به عنصر به رشتهٔ مقصد انتقال داد. همچنین هیچیک از عملگرهای C++ برروی رشته های زبان C عمل نمی کنند، اما در C++ استاندارد کلاسی با نام string در سرفایل string.h یا cstring فراهم شده تا بسیاری از عملیات روی رشته‌هایی که به‌صورت آرایه‌ای از کاراکترها هستند (رشته های زبان C) را تسهیل کند. علاوه بر این با استفاده از این کلاس قادر هستید نوع رشته را به‌عنوان یک نوع کتابخانه‌ای در دسترس داشته باشید و از آن بهره بگیرید. لذا ابتدا به بیان نوع string که توسط این کلاس فراهم شده می‌پردازیم و سپس به بررسی توابع و امکانات این کلاس در امر پردازش انواع رشته خواهیم پرداخت. شایان ذکر است برخی توابع این کلاس تنها برروی رشته هایی که از نوع string هستند عمل می‌کنند و برخی تنها برروی رشته‌های زبان C . لذا توجه به چگونگی کاربرد این توابع و الگوهای کلی آنها بسیار مهم است.

تعریف رشته‌ای از نوع string

تعریف رشته از نوع string بسیار آسان و به‌صورت کلی زیر است:

```
string  نام‌رشته ;
```

چنانکه در دستور فوق مشاهده می‌کنید در هنگام تعریف اینگونه رشته‌ها طول رشته به هیچ عنوان ذکر نمی‌شود و مدیریت حافظه از نظر میزان فضای لازم برای ذخیرهٔ یک رشته به عهدهٔ خود کامپایلر می‌باشد.

مقداردهی به رشته‌هایی از نوع string

مقداراولیه‌دهی به رشته‌ای از نوع string ، دارای حالات زیر است.

۱. می‌توان رشته‌ای از نوع string را به‌صورت مستقیم مقدار اولیه‌دهی کرد:

```
string str="Hello, World";
```

۲. می‌توان رشته‌ای از نوع string را به واسطهٔ رشتهٔ دیگری از نوع رشته‌های زبان C مقداردهی کرد:

```
char S[]="Hello, World";
string str=S;
```

۳. می‌توان رشته‌ای از نوع string را با رشته‌ای از همان نوع مقدار اولیه داد:

```
string s1="Hello, World";
string s2=s1;
```

همچنین می‌توان رشته‌ای از نوع string را در هر کجای برنامه به رشتهٔ دیگری با طول متفاوت مقداردهی کرد و چنانکه پیشتر گفته شد مدیریت حافظه از نظر طول رشته بر عهدهٔ برنامه‌نویس نیست. به‌عنوان مثال برنامهٔ مثال ۷-۱۰ را ببینید:

مثال ۷-۱۰ : برنامه‌ای که چگونگی مقداردهی و عملیات انتساب را در رشته‌هایی از نوع **string** نشان می‌دهد.

```
1.  //This program demonstrates initialized some strings.
2.  #include  <iostream>
3.  #include  <string>
4.  using namespace std;
5.  int main()
6.  {
7.     string S1,S2="Hello, Mother.";
8.      char  S3[ ]="Hello, Dad.";  //Less than S2
9.       S1="Hi boy.";
10.       cout<<S1<<endl;
11.        S1=S2;  //initial S1 with a longer string (S2)
12.       cout<<S1<<endl;
13.        S1=S3;  //initial S1 with a minor string (S3)
14.       cout<<S1<<endl;
15.    return 0;
16. }
```

توضیح مثال : چنانکه در خط ۷ این برنامه می‌بینید ابتدا رشتهٔ **S1** بدون مقدار اولیه و **S2** با مقدار اولیه تعریف شده‌اند. در خطوط ۹ و ۱۰ رشتهٔ **S1** را مقدار اولیه داده و به خروجی برده‌ایم. در خط ۱۱ با آنکه طول رشته **S1** برابر هفت کاراکتر است آن را با رشتهٔ **S2** به طول ۱۲ کاراکتر مقداردهی کرده‌ایم و به عکس در خط ۱۳ رشتهٔ **S1** را با رشتهٔ **S3** که دارای طول ۱۰ کاراکتر است و کوچکتر از رشتهٔ **S1** می‌باشد مقداردهی کرده‌ایم. بنابراین با توجه به این مثال می‌توان تصدیق کرد که مدیریت حافظه در خصوص رشته‌هایی از نوع **string** بر عهدهٔ برنامه‌نویس نیست.

معرفی برخی توابع عضو کلاس string

چنانکه در کاردر کلاس ۷-۲ دیدید تشخیص برابری دو رشته از نوع رشته‌های زبان **C** نیازمند ایجاد یک حلقهٔ تکرار جهت مقایسهٔ تک‌تک کاراکترهای متناظر دو رشته است. از طرفی از آنجا که هر نویسه در زبان‌های طبیعی دارای مکانی خاص درمجموعهٔ حروف الفبا است، رابطهٔ ترتیب را می‌توان برروی حروف الفبا تعریف‌کرد و در نتیجه قادر هستیم دو رشته را از نظر بزرگتری و کوچکتری مقایسه کرد. به‌عنوان مثال هنگامی‌که می‌خواهیم لیست دانشجویان یک کلاس را به ترتیب حروف الفبا مرتب کنیم، باید بتوانیم نام دانشجویان را مقایسه کنیم و آنها را به ترتیب از کوچک به بزرگ مرتب کنیم. اما چنانکه پیشتر دیدید عملیات مقایسه در خصوص دو رشته با دشواری‌هایی همراه است. لذا در هدر فایل **string.h** یا نوع استاندارد آن **string** برخی توابع فراهم شده است که بسیاری ازعملیات‌های مورد نیاز برروی رشته‌های زبان **C** و همچنین رشته‌هایی از نوع **string** را به راحتی امکان‌پذیرمی‌سازند. لذا در این‌قسمت به معرفی برخی از توابع مهم و پرکاربرد مربوط به هدرفایل **cstring** می‌پردازیم. ابتدا توابع مربوط به رشته‌های زبان **C** و سپس توابع مربوط به رشته‌های نوع **string** را مورد بررسی قرار می‌دهیم.

۱. **تابع strcpy :** پیشتر دیدید که با دستور انتساب نمی‌توانستیم یک رشته از نوع آرایه‌ای از کاراکترها را در رشته‌ای دیگر از همان نوع کپی کنیم، اما به‌جای نوشتن یک حلقهٔ تکرار برای انجام عمل کپی می‌توان از تابع **strcpy** استفاده کرد. شکل کلی کاربرد این تابع برای رشته‌های زبان **C** به‌صورت زیر است:

```
strcpy(رشتهٔ مقصد , رشتهٔ مبداء );
```

به‌عنوان مثال اگر بنویسیم:

```
strcpy(str1, str2);
```

محتویات **str2** در **str1** کپی می‌شود. و یا اگر بنویسیم:

```
strcpy(str, "computer");
```

رشتهٔ **"computer"** در **str** قرار می‌گیرد.

دو تذکر مهم :

الف- چنانکه پیشتر دیدید کپی کردن هرگونه رشته در رشته‌ای از نوع **string** به واسطهٔ دستور انتساب صورت می‌گیرد. بنابراین هیچ یک از پارامترهای تابع **strcpy** نمی‌تواند رشته‌ای از نوع **string** باشد.

ب- مراقب باشید که رشتهٔ مقصد دارای طول کوچکتری نسبت به رشتهٔ مبدأ نباشد، زیرا در صورت عدم رعایت این نکته امکان دستکاری داده‌های حافظه توسط تابع **strcpy** وجود دارد. به‌عنوان مثال با اجرای برنامه زیر مشاهده خواهید کرد که محتویات رشتهٔ مبدأ نیز مورد دستکاری قرار گرفته که نتیجهٔ مطلوبی نمی‌تواند در بر داشته‌باشد.

مثال ۷- ۱۱ : برنامه‌ای نحوهٔ استفادهٔ غلط از تابع **strcpy** را نشان می‌دهد.

```
1.  //This program shows usage of strcpy function.
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  int main()
6.  {
7.      char S1[6]="Hello";
8.       char S2[3];
9.        strcpy(S2,S1);
10.        cout<<S2<<"\t"<<S1<<endl;
11.   return 0;
12. }
```

توضیح مثال : خروجی حاصل از اجرای این برنامه به‌صورت زیر خواهد بود:

```
Hello      o
```

چنانکه مشاهده می‌کنید محتویات رشتهٔ **S1** از **"Hello"** به **"o"** تغییر یافته است.

۲. **تابع strcmp :** پیشتر در کار در کلاس ۷-۲ با عملیات مقایسهٔ دو رشته از نوع رشته‌های زبان C آشنا شدید. اما به واسطهٔ تابع **strcmp** می‌توان به‌راحتی دو رشته را با یکدیگر مقایسه کرد. شکل کلی به‌کارگیری این تابع به‌صورت زیر است:

```
int result=strcmp(رشتهٔ ۱ , رشتهٔ ۲);
```

به‌عنوان مثال خط زیر را در نظر بگیرید:

```
int result=strcmp(str1,str2);
```

عملکرد این تابع بدین صورت است که شروع به مقایسهٔ کاراکترهای هر دو رشته از ابتدای آنها می‌کند تا به اولین مورد اختلاف برسد اگر کاراکتر مربوط به رشتهٔ ۱ بزرگتر از کاراکتر مربوط به رشتهٔ ۲ باشد مقداری مثبت یعنی عدد ۱ را برمی‌گرداند. این بدان مفهوم است که **str1 > str2** . همچنین به عکس اگر کاراکتر مربوط به رشتهٔ ۱ کوچکتر از کاراکتر مربوط به رشتهٔ ۲ باشد مقداری منفی یعنی عدد ۱- را به مفهوم اینکه **str1 < str2** است برمی‌گرداند. از طرفی اگر دو رشته با یکدیگر مساوی باشند مقدار صفر برگردانده می‌شود.

تذکر: دوباره متذکر می‌شویم که تابع **strcmp** تنها برروی رشته‌هایی از نوع رشته‌های زبان C عمل می‌کند. اما برای مقایسهٔ یک رشته از نوع **string** با رشته‌ای از نوع رشته‌های زبان C به‌صورت زیر عمل می‌کنیم:

```
string st1 = "computer";
char st2[ ]= "compute";
int result = st1.compare(st2);
```

عملکرد تابع **compare** در این مورد دقیقاً مشابه عملکرد تابع **strcmp** می‌باشد. دیگر کاربردهای تابع **compare** را اندکی بعد خواهید دید.

۳. **تابع strcat :** با استفاده از این تابع می‌توان یک رشته را به انتهای رشتهٔ دیگر افزود. به عبارت دیگر می‌توان دو رشتهٔ زبان C را به واسطهٔ این تابع به یکدیگر الحاق کرد. شکل کلی کاربرد این تابع به‌صورت زیر است:

```
strcat(رشتهٔ مبدأ , رشتهٔ مقصد);
```

به‌عنوان مثال می‌توان نوشت:

```
strcat(str1,str2);
```

عملکرد این تابع به این صورت است که محتویات رشتهٔ **str2** را به انتهای رشتهٔ **str1** از مکانی که علامت **NULL** قرارگرفته اضافه می‌کند. به‌عنوان مثال اگر رشتهٔ **str1** شامل کلمهٔ **"computer"** باشد و رشتهٔ **str2** شامل کلمهٔ "science" باشد حاصل عملیات تابع **strcat** برروی این دو رشته ذخیرهٔ عبارت **"computer science"** در **str1** است.

دو تذکر مهم:

الف- دوباره توجه شما را به این نکته جلب می‌کنیم که رشتهٔ مقصد باید فضای لازم جهت افزوده شدن رشتهٔ مبدأ را داشته باشد. لذا در صورت عدم وجود چنین فضای آزادی، انتظار هرگونه دستکاری در محتویات حافظه و روبه‌رو شدن با نتایج غیره منتظره را باید داشته باشید.

ب- این تابع نیز تنها برای رشته‌های زبان C تعریف شده است و برای الحاق هر گونه رشته‌ای چه از نوع رشته های زبان C و چه از نوع رشته‌های string به رشته‌ای از نوع string باید از عملگر + استفاده کنید. به‌عنوان مثال به برنامهٔ زیر توجه کنید:

مثال ۷- ۱۲: برنامه‌ای که نحوهٔ الحاق یک رشته را به رشته‌ای از نوع **string** نشان می‌دهد.

```
1.  //This program shows concatenation of string objects.
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  int main()
6.  {
7.     char S1[ ]="computer";
8.      string S2=S1;
9.       cout<<S2<<endl;
10.    strcpy(S1," science");
11.     S2=S2+S1;
12.      cout<<S2<<endl;
13.    return 0;
14. }
```

۴. **تابع strncpy:** این تابع تعداد مشخصی از کاراکترهای یک رشته را در یک رشتهٔ دیگر کپی می‌کند. شکل کلی به‌کارگیری این تابع برای رشته‌هایی از نوع رشته‌های زبان C به‌صورت است:

strncpy(رشتهٔ مقصد+ نقطهٔ آغازین عمل کپی , رشتهٔ مبدأ , تعداد کاراکتری که باید کپی شود);

به‌عنوان مثال می‌توان نوشت:

```
strncpy(str1 + m ; str2; n);
```

در مثال فوق تعداد n کاراکتر ابتدایی رشتهٔ str2 را از کاراکتر (m+1)-اُم رشتهٔ str1 تا محل (m+n+1) کپی می‌کند. اگر تعداد کاراکترهایی که در str2 وجود دارد کمتر از مقدار n باشد، به تعداد لازم کاراکتر NULL در انتهای str1 کپی می‌شود.

مثال ۷– ۱۳ : برنامه‌ای که عملکرد تابع **strncpy** را نشان می‌دهد.

```
1.  //This program uses strcpy function
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  int main()
6.  {
7.    char str[ ] = "cats are nice usually.";
8.     cout<<"before operating :\n"<<str;
9.      strncpy(str,"dogs",4);
10.      strncpy(str+9,"mean",4);
11.       cout<<"\nafter operating :\n"<<str<<endl;
12.   return 0 ;
13. }
```

توضیح مثال : خروجی حاصل از اجرای این برنامه به‌صورت زیر خواهد بود:

```
before operating :
cats are nice usually.
after operating :
dogs are mean usually.
```

۵. **تابع strncmp :** این تابع تعداد مشخصی از کاراکترهای دو رشته را با یکدیگر مقایسه کرده و نتیجه را به شکل یک عدد **int** همانند تابع **strcmp** برمی‌گرداند.شکل کلی به‌کارگیری این تابع به‌صورت زیر است:

```
int result=strncmp(رشتهٔ ۱ , رشتهٔ ۲ , تعداد کاراکتری که باید مقایسه گردد);
```

به‌عنوان مثال دستور زیر را در نظر بگیرید:

```
int result=strncmp(str1,str2,n);
```

با اجرای این کد **n** کاراکتر ابتدای رشتهٔ اول با **n** کاراکتر ابتدای رشتهٔ دوم مقایسه شده و نتیجه به‌صورت یک عدد **int** برگشت‌داده می‌شود. البته توجه داشته‌باشید که در این تابع نیز می‌توان نقطهٔ آغازین مقایسه را در هر یک از رشته‌های اول و دوم با استفاده از عملگر + تعیین کرد. به‌عنوان مثال در دستور زیر عمل مقایسه از کاراکتر سوم رشتهٔ **str1** و کاراکتر ششم رشتهٔ **str2** آغاز می‌شود. این مطلب در خصوص تمامی توابعی که پارامتر ورودی آنها رشته باشد صادق است.

```
int result=strncmp(str1+2,str2+5,n);
```

۶. **تابع strnicmp_ :** این تابع همانند تابع **strncmp** عمل کرده با این تفاوت که مقایسهٔ دو رشته را بدون توجه به کوچکی و بزرگی حروف انجام می‌دهد. لذا دو رشتهٔ **"quick"** و **"QUICK"** از نظر این تابع با یکدیگر برابر هستند. شایان ذکر است که این تابع تنها مختص کامپایلرهای شرکت مایکروسافت می‌باشد.

مثال ۷- ۱۴: برنامه‌ای که نحوهٔ عملکرد دو تابع **strncmp** و **strnicmp_** را نشان می‌دهد.

```
1.  //This program shows difference of strncmp and _strnicmp
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  void main(void)
6.  {
7.     char string1[]="The quick brown dog jumps over the
8.                     lazy fox";
9.     char string2[]="The QUICK brown fox jumps over the
10.                    lazy dog";
11.    char tmp[20];
12.    int result;
13.     cout<<"Compare strings:\n\t\tString1= "<<string1
14.         <<"\n\t\tString2= "<<string2<<"\n\n";
15. cout<<"Function:\tstrncmp(first 10 characters only)\n";
16.     result = strncmp( string1, string2 , 10 );
17.      if( result > 0 )
18.          strcpy( tmp, "greater than" );
19.      else if( result < 0 )
20.          strcpy( tmp, "less than" );
21.      else
22.           strcpy( tmp, "equal to" );
23.    cout<<"Result:\t\tString 1 is "<<tmp<<" string 2\n\n";
24. cout<<"Function:\t_strnicmp (first 10 characters only)";
25.     result = _strnicmp( string1, string2, 10 );
26.      if( result > 0 )
27.          strcpy( tmp, "greater than" );
28.      else if( result < 0 )
29.          strcpy( tmp, "less than" );
30.      else
31.          strcpy( tmp, "equal to" );
32. cout<<"\nResult:\t\tString 1 is "<<tmp<<" string 2\n\n";
33. }
```

توضیح مثال: خروجی حاصل از اجرای این برنامه به‌صورت زیر خواهد بود:

```
Compare strings:
        String1= The quick brown dog jumps over the lazy fox
        String2= The QUICK brown fox jumps over the lazy dog
Function:   strncmp (first 10 characters only)
Result:     String 1 is greater than string 2

Function:   _strnicmp (first 10 characters only)
Result:      String 1 is equal to string 2
```

۷. **تابع strncat :** این تابع نیز تعداد مشخصی کاراکتر از ابتدای رشتهٔ مبدأ را به انتهای رشتهٔ مقصد متصل می‌کند، و به شکل کلی زیر به‌کار می‌رود:

```
strncat (رشتهٔ مقصد , رشتهٔ مبداء , تعداد کاراکتری که باید متصل شود);
```

۸. **تابع strlen :** این تابع جهت تشخیص طول یک رشتهٔ زبان C به‌کار می‌رود و دارای شکل کلی زیر است:

```
int length=strlen(رشتهٔ مورد نظر);
```

شایان ذکر است که طول آرایهٔ کاراکتری ممکن است با طول رشتهٔ موردنظر متفاوت باشد، چرا که این تابع تعداد کاراکترهای رشته را از ابتدای رشته تا نقطه‌ای که به علامت NULL برسد می‌شمارد.

مثال ۷-۱۵ : برنامه‌ای که کاربرد تابع strlen را نشان می‌دهد. با اجرای این برنامه عدد ۱۵ به‌عنوان طول رشتهٔ str چاپ می‌شود.

```
1. //This program shows usage of strlen function.
2. #include <iostream>
3. #include <string>
4. using namespace std ;
5. int main()
6. {
7.     char str [16] = "how long am I ?";
8.     int len ;
9.     len = strlen(str) ;
10.       cout<<str<<" is "<<len<<" characters long\n";
11.     return 0;
12.   }
```

۹. **تابع strchr :** این تابع اولین مکان یک کاراکتر را در یک رشته یافته و یک اشاره‌گر از نوع *char به مکان موردنظر در رشته برمی‌گرداند، در صورتی که کاراکتر موردنظر یافت نشود مقدار NULL بازگردانده می‌شود. این تابع به شکل کلی زیر به‌کار می‌رود:

```
char* position = strchr(رشتهٔ مورد نظر , کد اسکی کاراکتر مورد جستجو);
```

۱۰. **تابع strrchr :** تنها تفاوت این تابع با تابع strchr این است که آخرین مکان وقوع یک کاراکتر را در یک رشته برمی‌گرداند. به عبارت دیگر این تابع رشته را از انتها برای یافتن نخستین وقوع کاراکتر موردنظر جستجو می‌کند. به‌عنوان مثال دستور زیر رشتهٔ str را جهت یافتن آخرین وقوع کاراکتر 'b' مورد جستجو قرار می‌دهد:

```
char ch='b';
char* position=strrchr(str, ch);
```

مثال ۷-۱۶: برنامه‌ای که نحوهٔ عملکرد توابع **strchr** و **strrchr** را نشان می‌دهد.

```
1.  //This program shows usage of strchr & strrchr functions.
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  void main(void)
6.  {
7.     char ch='r';
8.     char string[] =
9.  "The quick brown dog jumps over the lazy fox";
10.    char fmt1[] =
11. "         1         2         3         4         5";
12.    char fmt2[] =
13. "12345678901234567890123456789012345678901234567890";
14.    char* pdest;
15.    int result;
16.    cout<<"String to be searched: \n\t\t"<<string<<"\n";
17.     cout<<"\t\t"<<fmt1<<"\n\t\t"<<fmt2<<"\n\n";
18.      cout<<"Search char:\t"<<ch<<"\n";
19. /* Search forward. */
20.    pdest=strchr(string, ch);
21.     result=pdest - string + 1;
22.      if( pdest != NULL )
23.        cout<<"Result:\tfirst "<<ch
24.            <<" found at position "<<result<<"\n\n";
25.     else
26.        cout<<"Result:\t"<<ch<<" not found\n";
27. /* Search backward. */
28.    pdest=strrchr(string, ch);
29.     result=pdest - string + 1;
30.      if( pdest != NULL )
31.        cout<<"Result:\tfirst "<<ch
32.            <<" found at position "<<result<<"\n\n";
33.      else
34.        cout<<"Result:\t"<<ch<<" not found\n";
35. }
```

توضیح مثال: این برنامه رشتهٔ **string** را برای یافتن کاراکتر **r** از ابتدا و انتها مورد جستجو قرار می‌دهد. با اجرای خطوط ۱۶ الی ۱۸ برنامه سه خط زیر چاپ می‌گردد که شما می‌توانید با استفاده از ارقام چاپ شده در خطوط دوم و سوم به‌راحتی شمارهٔ محل کاراکتر **r** را با آنچه توسط برنامه به‌دست می‌آید مطابقت دهید:

```
The quick brown dog jumps over the lazy fox
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
```

تنها نکته‌ای که شاید در این کد عجیب به نظر آید خطوط ۲۱ و ۲۹ می‌باشند که برای تبدیل آدرسی که در **pdest** ذخیره شده به یک عدد آدرس ابتدای رشته (یعنی **string**) را از آن آدرس کم کرده و سپس با ۱ جمع نموده‌ایم.

۱۱. **تابع strupr :** این تابع حروف یک رشته را به حروف بزرگ تبدیل می‌کند و اشاره‌گری به رشتهٔ تبدیل شده، از نوع **char*** را برمی‌گرداند. شکل کلی بکارگیری این تابع به‌صورت زیر است:

```
strupr(نام رشتهٔ مورد نظر);
```

۱۲. **تابع strlwr :** این تابع تمامی حروف رشته را به حروف کوچک تبدیل می‌کند. شکل کلی به‌کارگیری این تابع به‌صورت زیر است:

```
strlwr(نام رشتهٔ مورد نظر);
```

مثال ۷– ۱۷ : برنامه‌ای که نحوهٔ عملکرد دو تابع **strupr_** و **strlwr_** را نشان می‌دهد.

```
1.  //This program uses _strupr and _strlwr function.
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  void main()
6.  {
7.     char string[] = "The String to End All Strings!";
8.       cout<<"Mixed:\n"<<string<<endl;
9.         cout<<"Uper:\n"<<strupr(string)<<endl;
10.          char* lwr_str=strlwr(string);
11.            cout<<"Lower:\n"<<lwr_str<<endl;
12. }
```

۱۳. **تابع strcspn :** این تابع کاراکترهای یک زیر رشته را در یک رشته جستجو کرده و مکان اولین موردی که یکی از کاراکترهای زیر رشته در رشتهٔ اصلی وجود داشته باشد را به‌صورت یک عدد **int** برمی‌گرداند. شکل کلی به‌کارگیری این تابع به‌صورت زیر است:

```
int position = strcspn(رشتهٔ ۱ , رشتهٔ ۲);
```

مثال ۷– ۱۸ : برنامه‌ای که رشتهٔ **abc** را در رشتهٔ **wxyzabc** جستجو می‌کند.

```
1.  //This program uses strcspn function
2.  #include <string>
3.  #include <iostream>
4.  using namespace std;
5.  int main()
6.  {
7.     char string[] = "wxyzabc";
8.       int  pos;
9.         pos = strcspn( string, "abc" );
10.          cout<<"First a, b or c in "<<string
11.              <<" is at character "<<pos+1<<endl;
12.     return 0;
13. }
```

توضیح مثال: خروجی این برنامه به‌صورت زیر خواهد بود:

```
First a, b or c in wxyzabc is at character 5
```

۱۴. **تابع strset :** این تابع محتویات یک رشته را با کاراکتری مشخص پر می‌کند. شکل کلی به‌کارگیری این تابع به‌صورت زیر است.

```
strset(رشتهٔ مورد نظر , کاراکتر جایگزینی);
```

به‌عنوان مثال دستور زیر رشتهٔ **str** را با کاراکتر **x** پر می‌کند:

```
strset(str, 'x');
```

۱۵. **تابع strnset :** این تابع یک کاراکتر را به تعداد دفعات مشخصی در یک رشته کپی می‌کند. به‌عنوان مثال دستور زیر کاراکتر **0** را به تعداد ۸ بار در ابتدای رشتهٔ **str** کپی می‌کند:

```
strnset(str, '0', 8);
```

۱۶. **تابع strrev :** این تابع با دریافت یک رشتهٔ زبان **C** محتویات آن را معکوس می‌کند، به عبارت دیگر کاراکتر ابتدا را به انتها منتقل می‌کند و کاراکتر انتها را به ابتدا و این عمل را برای تمامی کاراکترهای رشته انجام‌می‌دهد. به‌عنوان مثال دستور زیر موجب معکوس‌شدن رشتهٔ **str** می‌شود:

```
strrev(str);
```

۱۷. **تابع strtok** : این تابع نشانه‌های موجود در یک رشته را مشخص می‌کند و بیشتر برای تجزیهٔ رشته‌ها کاربرد دارد. این تابع به‌صورت کلی زیر به‌کار می‌رود:

```
char* token = strtok( str1 , str2 );
```

در الگوی فوق**str1** رشته‌ای است که نشانه‌های موجود در آن باید جداشوند و **str2** رشته‌ای است که جدا کننده‌ها را مشخص می‌کند. این تابع اشاره‌گری را به نشانهٔ یافته‌شده برمی‌گرداند.

مثال ۷– ۱۹ : در برنامهٔ زیر سه کاراکتر \n و \t و **space** به‌عنوان جداکننده هستند. به نحوهٔ تجزیهٔ رشتهٔ **string** توجه کنید.

```
1. //strtok function finds the next token in a string.
2. #include <iostream>
3. #include <string>
4. using namespace std;
5. void main()
6. {
7.   char string[]="A string\tof ,,tokens\nand some tokens";
8.    char delimiters[]   = " ,\t\n";
9.     char* token;
10.     cout<<string<<"\n\nTokens:\n";
11.      /* Establish string and get the first token: */
12.      token = strtok( string, delimiters );
13.       while( token != NULL )
14.           {
15.                /* While there are tokens in "string" */
16.                 cout<<" "<<token<<"\n";
17.               /* Get next token: */
18.                 token = strtok( NULL, delimiters );
19.           }
20. }
```

توضیح مثال: خروجی این برنامه به‌صورت زیر خواهد بود:

```
 A string   of ,,tokens
and some  more tokens

Tokens:
 A
 string
 of
 tokens
 and
 some
 more
 tokens
```

اکثر توابعی که تا اینجا بررسی کردیم برروی رشته‌هایی از نوع رشته‌های زبان **C** عمل می‌کردند اما توابع و عملگرهایی که از اینجا به بعد بررسی خواهیم کرد، اکثراً در خصوص رشته‌هایی از نوع **string** کاربرد دارند. لذا این توابع را در پنج دسته کلی مورد بررسی قرار می‌دهیم، در تمامی این موارد فرض کنید **str** رشته‌ای از نوع **string** است که به‌صورت زیر تعریف شده است:

```
string str;
```

الف– جستجوی یک زیر رشته در رشته‌هایی از نوع string

۱۸. **تابع find :** این تابع جهت یافتن یک زیر رشته در یک رشته از نوع **string**، به‌کار می‌رود. این تابع به‌صورت کلی زیر به‌کار می‌رود:

```
int pos = str.find(نقطهٔ شروع جستجو , زیر رشتهٔ مورد جستجو) ;
```

عملکرد این تابع به این صورت است که اگر زیر رشتهٔ داده شده به‌عنوان ورودی تابع در رشتهٔ **str** یافت شود موقعیت آن را در رشتهٔ **str** و درغیر این صورت عدد ۱- را برمی‌گرداند.

۱۹. **تابع rfind :** این تابع از نظر به‌کارگیری دقیقاً مشابه تابع **find** است با این تفاوت که عمل جستجو را از نقطهٔ شروع به‌طور معکوس (به سمت ابتدای رشته) انجام می‌دهد.

۲۰. **تابع find_first_of :** این تابع رشته را به جهت یافتن اولین موقعیتی که یک کاراکتر از عناصر یک مجموعه کاراکتر، وجود داشته باشد مورد جستجو قرار می‌دهد و به‌صورت کلی زیر به کار می‌رود:

```
int pos;
pos = str.find_first_of(نقطهٔ شروع جستجو , یک مجموعه کاراکتر به صورت رشته) ;
```

به‌عنوان مثال در قطعه کد زیررشتهٔ **str** به جهت یافتن اولین موقعیتی که یکی از کاراکترهای مجموعهٔ **"xquae"** وجود داشته باشد مورد جستجو قرار می‌گیرد.

```
int pos = str.find_first_of("xquae");
```

۲۱. **تابع find_first_not_of :** این تابع همانند تابع **find_first_of** عمل کرده با این تفاوت که رشته را به جهت یافتن اولین کاراکتری که داخل یک مجموعه از کاراکترها نباشد مورد جستجو قرار می‌دهد. در صورت یافته شدن چنین کاراکتری در رشته، موقعیت کاراکتر را در رشته به‌صورت یک عدد **int** بازمی‌گرداند و در غیر این صورت مقدار ۱- را برمی‌گرداند.

۲۲. **توابع find_last_of و find_last_not_of :** این دو تابع دقیقاً مشابه دو تابع قبلی عمل می‌کنند با این تفاوت که همانند تابع **rfind** رشته را از انتها به ابتدا، به جهت یافتن آخرین کاراکتر موردنظر جستجو می‌کند.

مثال ۷- ۲۰ : برنامه‌ای که عملکرد توابع جستجو در رشته‌هایی از نوع **string** را نشان می‌دهد.

```
1.  //This program finds a substring of a string.
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  int main()
6.  {
7.      string s1 = "In Canada did Kublai Kahn a stately
8.                   pleasure dome decree";
9.      int n;
10.   ////////////////////
11.     n=s1.find("Kublai");
12.      cout<<"Found Kublai at "<<n+1<<endl;
13.   ////////////////////
14.     n=s1.find_first_of("spde");
15.      cout<<"First of spde at "<<n+1<<endl;
16.   ////////////////////
17.     n=s1.find_first_not_of("aeiouAEIOU");
18.      cout<<"First consonant at "<<n+1<<endl;
19.   ////////////////////
20.   return 0;
21. }
```

توضیح مثال: خروجی حاصل از اجرای این برنامه به‌صورت زیر خواهد بود:

```
Found Kublai at 15
First of spde at 8
First consonant at 2
```

ب– اعمال تغییرات در رشته‌هایی از نوع string

۲۳. **تابع erase :** با استفاده از این تابع می‌توان کاراکترهایی را از رشته حذف کرد. این تابع به‌صورت‌های کلی زیر به کار می‌رود:

```
str.erase();
str.erase(نقطهٔ شروع عملیات , تعداد کاراکترهایی که باید پاک شود);
```

در شکل اول تمامی کاراکترهای رشتهٔ **str** پاک می‌شود و در شکل دوم تعداد مشخصی کاراکتر را از نقطهٔ شروع عملیات پاک می‌کند.

۲۴. **تابع replace :** به وسیلهٔ این تابع قسمتی از یک رشته را با زیر رشتهٔ دیگری تعویض کرد. این تابع به‌صورت‌های کلی زیر به کار می‌رود:

```
str.replace(نقطهٔ شروع جایگزینی , تعداد کاراکترهایی که باید جایگزین شود , رشتهٔ مبداء);
```

در این شکل از کاربرد تابع **replace** تعداد مشخصی از کاراکترهای رشتهٔ **str** از نقطهٔ شروع با کاراکترهای موجود در رشتهٔ مبدأ تعویض می‌شود. شکل دیگری از بکارگیری این تابع به‌صورت زیر است:

```
str.replace(نقطهٔ شروع جایگزینی , تعداد کاراکترهایی که باید جایگزین شود , رشتهٔ مبداء
, نقطهٔ آغاز خواندن کاراکترهااز رشتهٔ مبداء , تعداد کاراکترهایی که باید از رشته مبداء خواندهشود);
```

۲۵. **تابع insert :** با استفاده از این تابع می‌توان کاراکترهایی را در هر جایی از رشته درج کرد. شکل کلی به‌کارگیری این تابع به‌صورت‌های زیر است:

```
str.insert(نقطهٔ آغاز عملیات درج , رشته‌ای که باید درج شود);
str.insert(نقطهٔ آغاز عملیات درج , رشتهٔ مبدأ , تعداد کاراکترهایی که باید درج شود);
str.insert(نقطهٔ آغاز عملیات درج , رشتهٔ مبدأ , نقطهٔ آغاز خواندن کاراکترها از رشتهٔ مبدأ
, تعداد کاراکترهایی که باید از رشتهٔ مبدأ درج شود);
```

در حالت اول رشته‌ای که باید درج شود به‌طور کامل از نقطهٔ آغاز درج در رشتهٔ **str** درج می‌شود. در کاربرد دوم تعداد مشخصی از کاراکترهای ابتدایی رشتهٔ مبدأ، در رشتهٔ **str** درج می‌شود. و در کاربرد سوم می‌توان بخش خاصی از رشتهٔ مبدأ را در رشتهٔ **str** با تعیین نقطهٔ آغازین عملیات خواندن از رشتهٔ مبدأ، درج کرد.

مثال ۷– ۲۱ : برنامه‌ای که نحوهٔ عملکرد توابع **erase** و **replace** و **insert** را نشان می‌دهد.

```
1.  //This program changes a string with some functions.
2.  #include <string>
3.  #include <iostream>
4.  using namespace std ;
5.  int main()
6.  {
7.     string s1("Quick! Send for Count Graystone.");
8.      cout<<"Befor operation s1 = "<<s1<<endl;
9.       string s2("Lord");
10.       string s3("Don't ");
11.   s1.erase(0,7);           //remove "Quick!"
12.    s1.replace(9 , 5 , s2); //remove "Count" with "Lord"
13.     s1.replace(0,1,"s");   //insert 'S' with 's'
```

```
14.       s1.insert(0,s3);       //insert "Don't " at beginning
15.    s1.erase(s1.size()-1,1); //remove '.'
16.     cout<<"Befor loop s1 = "<<s1<<endl;
17.      int x=s1.find(' ');     //find first space
18.       while(x < s1.size())   //loop while space remain
19.        {
20.           s1.replace(x,1,"/");        //replace with slash
21.           x=s1.find(' ');             //find next space
22.        }
23.       cout<<"After loop s1 = "<<s1<<endl;
24.     return 0;
25. }
```

توضیح مثال: خروجی حاصل از این برنامه به‌صورت زیر است:

```
Befor operation s1 = Quick! Send for Count Graystone.
Befor loop s1 = Don't Send for Lord Graystone
Befor loop s1 = Don't/Send/for/Lord/Graystone
```

۲۶. **تابع append :** به‌واسطهٔ این تابع می‌توان رشته‌ای را به انتهای رشتهٔ دیگری الحاق کرد. گرچه به‌واسطهٔ عملگر + نیز می‌توان عمل الحاق رشته‌ای را به رشته‌هایی از نوع **string** افزود، اما این تابع این مزیت را دارد که می‌توان بخش خاصی از یک رشته را به‌رشته‌ای از نوع **string** افزود. این تابع به‌صورت‌های کلی زیر به‌کار می‌رود:

```
str.append (رشته‌ای که باید الحاق شود) ;
str.append (تعداد کاراکتری که باید الحاق شود , رشته ای از نوع آرایه ای از کاراکترها) ;
str.append (نقطهٔ شروع خواندن از رشتهٔ مبداء , رشتهٔ مبدأ از نوع string ,
تعداد کاراکتری که باید الحاق شود) ;
str.append (اشاره گری به انتهای زیر رشتهٔ مبدأ , اشاره گری به ابتدای زیر رشتهٔ مبدأ) ;
```

در کاربرد اول رشته‌ای از نوع زبان **C** یا رشته‌ای از نوع **string** را به رشتهٔ **str** الحاق می‌کند. در کاربرد دوم تعداد مشخصی از کاراکترهای ابتدای یک رشتهٔ زبان **C** را به رشتهٔ **str** الحاق می‌کند. کاربردهای سوم وچهارم تنها در مورد رشته‌هایی از نوع **string** به‌کارمی‌روند. در کاربرد سوم نقطهٔ شروع خواندن از رشتهٔ مبدأ و تعداد کاراکتری که باید الحاق شود ذکرمی‌گردد. در کاربرد چهارم باید دو اشاره‌گر(یا تکرارگر) به ابتدا و انتهای مکانی از حافظه که رشتهٔ مبدأ قرار دارد ذکر می‌گردد. در حالتی دیگر می‌توان یک کاراکتر خاص را به تعداد مشخصی به انتهای یک رشته افزود، در این حالت تابع **append** به‌صورت کلی زیر به کار می‌رود:

```
str.append (کداسکی کاراکتر , تعداد دفعاتی که باید کاراکتر به انتهای رشته افزوده شود) ;
```

۲۷. **تابع c_str :** این تابع رشته‌ای از نوع **string** را به رشته‌ای از نوع رشته‌های زبان **C** (آرایه‌ای از کاراکترها) تبدیل می‌کند و یک اشاره‌گر ثابت از نوع **char** را به خانهٔ اول آرایهٔ کاراکتری برمی‌گرداند زیرا چنانکه در مبحث اشاره‌گرها خواهید دید نام آرایه معادل اشاره‌گری ثابت به خانهٔ اول آرایه است. این تابع به‌صورت کلی زیر به‌کار می‌رود:

```
const char* s = str.c_str();
```

به‌عنوان مثال دستور فوق رشتهٔ **str** را به آرایه‌ای از کاراکترها تبدیل کرده و آدرس آن را در **s** قرار می‌دهد.

مثال ۷– ۲۲: برنامه‌ای که کاربرد دستور **append** را نشان می‌دهد.

```
1.  //This program uses append function.
2.  #include <string>
3.  #include <iostream>
4.  using namespace std ;
5.  void main()
6.  {
7.    string str1("012");
8.     string str2("345");
9.      cout<<"str1 = "<<str1.c_str()<<endl;
10.      // appends str2 to str1
11.       str1.append(str2);
12.        cout<<"str1 = "<<str1.c_str()<<endl;
13.      // appends last 2 items in str2 to str1
14.       str2 = "567";
15.      // begins at pos 1, appends 2 elements
16.       str1.append(str2, 1, 2);
17.        cout<<"str1 = "<<str1.c_str()<<endl;
18.      //appends first 2 items from an array of
19.      //the element type
20.        char achTest[] = {'8', '9', 'A'};
21.         str1.append(achTest, 2);
22.          cout<<"str1 = "<<str1.c_str()<<endl;
23.      // appends all of a string literal to str1
24.        char szTest[] = "ABC";
25.         str1.append(szTest);
26.          cout<<"str1 = "<<str1.c_str()<<endl;
27.      // appends one item of the element type
28.        str1.append(1, 'D');
29.         cout<<"str1 = "<<str1.c_str()<<endl;
30.     // appends str2 to str1 using iterators
31.       str2 = "EF";
32.        str1.append (str2.begin(), str2.end());
33.         cout<<"str1 = "<<str1.c_str()<<endl;
34. }
```

توضیح مثال: خروجی حاصل از این برنامه به‌صورت زیر خواهد بود:

```
str1 = 012
str1 = 012345
str1 = 01234567
str1 = 0123456789
str1 = 0123456789ABC
str1 = 0123456789ABCD
str1 = 0123456789ABCDEF
```

۲۸. **تابع assign :** این تابع نیز عمل انتساب هر نوع رشته‌ای را به رشتهٔ دیگری از نوع **string** امکانپذیر می‌سازد. تفاوت این تابع با عملگر = در انتساب رشته‌ها این است که به‌واسطهٔ این رشته می‌توان قسمتی از یک رشته را به رشتهٔ دیگری انتساب داد. از نظر شکل کلی به‌کارگیری این تابع دقیقاً مشابه تابع **append** می‌باشد. به مثال ۷-۲۳ توجه کنید:

مثال ۷- ۲۳ : برنامه‌ای که چگونگی کاربرد تابع **assign** را نشان می‌دهد.

```
1.  // This program uses assign function compile with: /EHsc
2.  #include <string>
3.  #include <iostream>
4.  int main( )
5.  {
6.    using namespace std;
7.    // The first member function assigning the
8.    // characters of a C-string to a string
9.     string str1a;
10.     char cstr1a[] = "Out There";
11.      cout<<"The C-string cstr1a is: "<<cstr1a<<"."<<endl;
12.       str1a.assign(cstr1a);
13.        cout<<"Assigning the C-string cstr1a to string
14.               str1 gives: "<<str1a<<"."<<endl<<endl;
15.   // The second member function assigning a specific
16.   // number of the of characters a C-string to a string
17.    string  str1b;
18.     char cstr1b[] = "Out There";
19.      cout<<"The C-string cstr1b is: "<<cstr1b<<endl;
20.       str1b.assign(cstr1b,3);
21.    cout<<"Assigning the 1st part of the C-string cstr1b "
22.        <<"to string str1 gives: "<<str1b<<".\n\n";
23.   //The third member function assigning a specific number
24.   // of the characters from one string to another string
25.    string str1c="Hello ", str2c="Wide World ";
26.     cout<<"The string str2c is: "<<str2c<<endl;
27.      str1c.assign(str2c, 5, 5);
28.        cout<<"The newly assigned string str1 is: "
29.            <<str1c<<"."<<endl<< endl;
30.    // The fourth member function assigning the characters
31.    // from one string to another string in two equivalent
32.    // ways, comparing the assign and operator =
33.     string str1d="Hello", str2d="Wide", str3d="World";
34.      cout<<"The original string str1 is: "<<str1d<<".\n";
35.       cout<<"The string str2d is: "<<str2d<<endl;
36.        str1d.assign(str2d);
37.         cout<<"The string str1 newly assigned with string
38.                str2d is: "<<str1d<<"."<<endl;
39.         cout<<"The string str3d is: "<<str3d<<"."<<endl;
40.          str1d=str3d;
41.         cout<<"The string str1 reassigned with string
42.                str3d is: "<<str1d<<"."<<endl<<endl;
43.    // The fifth member function assigning a specific
```

```
44.     // number of characters of a certain value to a string
45.      string str1e="Hello ";
46.       str1e.assign(4,'!');
47.        cout<<"The string str1 assigned with exclamations
48.               is: "<<str1e<<endl<<endl;
49.     // The sixth member function assigning the value from
50.     // the range of one string to another string
51.      string str1f="Hello ",str2f="Wide World ";
52.       cout<<"The string str2f is: "<<str2f<<endl;
53.        str1f.assign(str2f.begin()+5,str2f.end()-1);
54.         cout<<"The string str1 assigned a range of string
55.                str2f is: "<<str1f<<"."<<endl<<endl;
56.     return 0;
57. }
```

توضیح مثال : خروجی حاصل از این برنامه به‌صورت زیر خواهد بود:

```
The C-string cstr1a is: Out There.
Assigning the C-string cstr1a to string str1 gives: Out There.

The C-string cstr1b is: Out There
Assigning the 1st part of the C-string cstr1b to string str1
gives: Out.

The string str2c is: Wide World
The newly assigned string str1 is: World.

The original string str1 is: Hello.
The string str2d is: Wide
The string str1 newly assigned with string str2d is: Wide.
The string str3d is: World.
The string str1 reassigned with string str3d is: World.

The string str1 assigned with exclamations is: !!!!

The string str2f is: Wide World
The string str1 assigned a range of string str2f is: World.
```

ت– عملگرهای مختلف تعریف شده برای رشته‌هایی از نوع string

اگر به‌خاطر داشته باشید عملگرهایی چون > برروی رشته‌هایی از نوع رشته‌های زبان C عمل نمی‌کردند. اما بسیاری از این عملگرها برای رشته‌هایی از نوع string قابل استفاده می‌باشند. که عبارتند از:

کوچکتر یا مساوی	<=	انتساب	=
بزرگتر	>	الحاق	+
بزرگتر یا مساوی	>=	انتساب الحاق	+=
اندیس دسترسی به کاراکترهای رشته	[ ]	تساوی	==
عملگر خروجی با cout	<<	نا برابری	!=
عملگر ورودی با cin	>>	کوچکتر	<

ث– مقایسهٔ رشته‌هایی از نوع string با یکدیگر

۲۹. **تابع compare :** به واسطهٔ این رشته می توان بخشی از یک رشته را با رشتهٔ دیگری مقایسه کرد. این تابع به صورت‌های کلی زیر به کار می‌رود:

```
int r;
r= str.compare(رشتهٔ دوم);
r=str.compare(str رشتهٔ در شروع مقایسه نقطهٔ , تعداد کاراکتر مورد مقایسه , رشتهٔ دوم);
r=str.compare(str رشتهٔ در شروع مقایسه نقطهٔ , str رشتهٔ از مقایسه مورد کاراکتر تعداد ,
string نوع از دوم رشتهٔ , دوم رشتهٔ در مقایسه شروع نقطهٔ , دوم رشتهٔ در مقایسه مورد کاراکتر تعداد);
r=str.compare(str رشتهٔ در شروع مقایسه نقطهٔ , str رشتهٔ از مقایسه مورد کاراکتر تعداد ,
C زبان رشته‌های نوع از دوم رشتهٔ , دوم رشتهٔ در مقایسه شروع نقطهٔ);
```

نتیجهٔ تمامی این توابع بدین صورت است که اگر رشتهٔ **str** بزرگتر از رشتهٔ دوم باشد مقداری مثبت و اگر برابر باشند مقدار صفر و اگر رشتهٔ **str** کوچکتر باشد مقداری منفی برگشت داده می‌شود.

۳۰. **تابع substr :** این تابع زیر رشته‌ای را از داخل یک رشته استخراج کرده و برمی‌گرداند. و دارای شکل کلّی زیر است:

```
string sub_str = str.substr(نقطهٔ شروع زیر رشته , نقطهٔ پایان زیر رشته);
```

مثال ۷– ۲۴ : برنامه‌ای که عمل مقایسهٔ رشته‌ها را با عملگرها و تابع **compare** نشان می‌دهد.[۱]

```
1.  //This program compares string objects
2.  #include <iostream>
3.  #include <string>
4.  int main()
5.  {
6.     using namespace std;
7.      string name="George";
8.       string username;
9.        cout<<"Enter your first name: ";
10.       cin>>username;
11.         if(username==name)
12.            cout<<"Getting's, George\n";
13.         else if(username<name)
14.            cout<<"You come before George\n";
15.         else
16.            cout<<"You come after George\n";
17.    //compare with function
18.       int r=username.compare(0,2,name,0,2);
19.        cout<<"The first two letters of your name ";
20.         if(r==0)
21.            cout<<"match ";
```

۱. در لوح فشردهٔ شمارهٔ ۱ همراه کتاب، برنامه‌ای با نام **compare** وجود دارد که مثال بسیار خوبی از نحوهٔ به‌کارگیری این تابع است.

```
22.          else if(r<0)
23.             cout<<"come before ";
24.          else
25.             cout<<"come after ";
26.       cout<<name.substr(0,2)<<endl;
27.    return 0;
28. }
```

توضیح مثال : خروجی حاصل از این برنامه به‌صورت زیر خواهد بود:

```
Enter your first name: Alfred
You come before George
The first tow letters of your name come before Ga
```

چنانکه در خط ۲۶- اُم این کد می‌بینید به واسطهٔ تابع **substr** دو کاراکتر اول رشتهٔ **name** را به‌صورت یک زیر رشته به خروجی فرستاده‌ایم. همچنین به به‌کارگیری عملگرها در مقایسهٔ رشته‌ها در خطوط ۱۱ و ۱۳ توجه کنید.

کار در کلاس ۷-۳ :

برنامه‌ای به زبان **C++** *بنویسید که عمل جستجو و جایگزینی در یک متن را انجام دهد. این عمل مشابه عمل* **replace** *در نرم افزار* **MS-WORD** *یا* **Text** *در* **Dos** *باید پیاده سازی شود.*

ج– دیگر توابع سرفایل string

توابع دیگری نیز وجود دارند که اعمال مفیدی را برروی رشته‌های نوع **string** انجام می‌دهند. با بسیاری از این‌توابع در قسمت وکتورها آشنا شده‌اید، لذا در این قسمت تنها به بیان نام این توابع می‌پردازیم و بررسی چگونگی به‌کارگیری این توابع را به‌عنوان یک عمل تحقیقاتی به عهدهٔ خوانندگان عزیز می‌گذاریم.

دو تابع **size** و **length** تعداد کاراکترهای موجود در رشته‌ای از نوع **string** را برمی‌گرداند. مقدار حافظهٔ اشغال شده توسط یک رشته معمولاً اندکی بیشتر از حافظهٔ واقعی مورد نیاز برای کاراکترها است. تابع **capacity** حافظهٔ واقعی اشغال شده را برمی‌گرداند. و تابع **max_sixe** حداکثر طول مجاز یک رشته از نوع **string** را برمی‌گرداند. این مقدار بستگی به طول متغیرهای **int** سیستم کامپیوتری دارد. همچنین با استفاده از تابع **at** به‌راحتی می‌توان به کاراکترهای یک رشته همچون عملگر [ ] دسترسی پیدا کرد، با این تفاوت که تابع **at** مرز رشته را نیز چک می‌کند. در زیر برخی توابع کاراکتری پرکاربرد لیست شده‌اند.

نام تابع	عملکرد تابع
isdigit	این تابع کد اسکی کاراکتری را به‌عنوان ورودی گرفته اگر کاراکتر دریافتی یک رقم باشد مقدار ۱ و در غیر این صورت مقدار صفر را بازمی گرداند.
isalpha	این تابع کد اسکی کاراکتری را به‌عنوان ورودی پذیرفته اگر کاراکتر دریافتی جزء حروف الفبا باشد مقدار ۱ را بازمی گرداند.
isalnum	این تابع کد اسکی کاراکتری را به‌عنوان ورودی دریافت کرده، اگر یک رقم یا یک حرف انگلیسی باشد مقدار ۱ و در غیر این صورت مقدار صفر را باز می‌گرداند.
isascii	این تابع کاراکتری را به‌عنوان ورودی پذیرفته و تشخیص می‌دهد که آیا کاراکتر در بازهٔ کدهای اسکی (صفر تا **7fH**) قرار دارد یا خیر.
islower و **isupper**	تابع **islower** کد اسکی کاراکتری را به‌عنوان ورودی گرفته اگر کاراکتر دریافتی یک حرف کوچک لاتین باشد مقدار ۱ و در غیر این صورت مقدار صفر را بازمی گرداند. و تابع **isupper** به جهت تشخیص حروف بزرگ به‌کار می رود.
isprint	این تابع کد اسکی کاراکتری را به‌عنوان ورودی گرفته اگر کاراکتر دریافتی قابل چاپ باشد مقدار ۱ و در غیر این صورت مقدار صفر را بازمی‌گرداند.
isgraph	مشابه تابع **isprint** عمل می‌کند با این تفاوت که **blank** را جزء کاراکترهای قابل چاپ نمی‌داند. کاراکترهای قابل چاپ در محدودهٔ ۳۳ تا ۱۲۶ جدول اسکی قرار دارد.
isspace	این تابع تشخیص می‌دهد که آیا کاراکتر ورودی یک کاراکتر فضای خالی مثل کاراکترهای **blank** یا **tab** یا **carriage return** یا **new line** یا **form feed** است یا خیر.
ispunct	این تابع تشخیص می‌دهد که آیا کاراکتر ورودی یک کاراکتر ویرایش مثل ، یا . است یا نه.

۷-۷ : تمرین

۱. کدام یک از جملات زیر درست و کدام یک نادرست است:

الف- در یک آرایه نمی‌توان مقادیر متفاوتی را ذخیره کرد اما در یک وکتور دو بعدی می‌توان.

ب- اندیس یک آرایه نمی‌تواند از نوع اعشاری باشد.

ج- اگر تعداد مقادیر اولیهٔ آرایه، کمتر از تعداد عضوهای آرایه باشد مقدار اولیهٔ عضوهای باقی‌مانده به‌طور اتوماتیک، آخرین مقدار داخل لیست مقادیر اولیه است.

ث- اگر لیست مقادیر اولیه بیشتر از طول آرایه باشد کامپایلر پیغام خطایی را صادر می‌کند.

ج- تعریف آرایه‌هایی با طول متغیر به راحتی امکانپذیر است.

۲. در یک بازی حکم ۵۲ ورق بازی بین چهار نفر به‌طور مساوی تقسیم می‌شود. برنامه‌ای بنویسید که پس از شبیه سازی عمل بُرزدن ورق‌ها به وسیلهٔ مولد اعداد تصادفی، ورق‌ها را به‌طور تصادفی بین چهار بازی‌کن تقسیم کند و ورق‌های هر بازی‌کن را در خروجی نمایش دهد. نماد انواع ورق‌ها در جدول کد اسکی موجود است.

۳. برنامه‌ای بنویسید که یک تاس را ۳۶۰۰۰ مرتبه بریزد و فراوانی هریک از اعداد ۱ تا ۶ را در یک آرایهٔ ۷ عنصری ذخیره کند. سپس با رسم نمودار هیستوگرام (نمودار میله‌ای عمودی) مربوط به این جدول فراوانی نشان دهید که توزیع جدول دارای یک توزیع یکنواخت و طبیعی است و مولد اعداد تصادفی در کامپیوتر اعدادی واقعاً تصادفی تولید می‌کند.

۴. برنامهٔ بنویسید که یک آرایه از اعداد صحیح با تعداد زوجی عنصر را به دو آرایهٔ مساوی بدین صورت تقسیم کند که حاصل جمع عناصر هر یک از این دو آرایه با یکدیگر حداکثر اختلاف را داشته باشند.

۵. مسئلهٔ قبل را برای حداقل اختلاف نیز حل کنید.

۶. برنامه‌ای بنویسید که توسط مرتب‌سازی درجی عناصر یک بردار را در حین دریافت از ورودی مرتب کند.

۷. برنامه‌ای بنویسید که مثلث خیام- پاسکال را در یک آرایهٔ دو بعدی تا ردیف بیستم ایجاد کند و به خروجی ببرد.

۸. برنامه‌ای بنویسید که قادر باشد یک عنصر را از یک آرایهٔ متشکل از اعداد صحیح حذف کند.

۹. برنامه‌ای بنویسید که یک دیکشنری با صد لغت انگلیسی را پیاده‌سازی کند. صد لغت را خودتان از یک فرهنگ لغت انتخاب کنید و به‌همراه توضیحات آنها در یک بردار چند بعدی از نوع **string** ذخیره‌سازی کنید. سپس کاربر با وارد کردن یک کلمه آن کلمه را جستجو می‌کند. اگر کلمه موجود بود باید معنای آن نمایش داده شود والا باید کاربر بتواند آن کلمه را به دیکشنری اضافه کند.

۱۰.برنامه‌ای به زبان C++ بنویسید که روش غربال اراتوستن (sieve of Eratosthenes) را برای پیدا کردن اعداد اول تا جایی که کلمات کامپیوتر ظرفیت دارد را پیاده‌سازی کند.

۱۱.برنامه‌ای بنویسید که ۵ نمرهٔ امتحانی ۲۰ دانشجو را به همراه نام آنها از ورودی بخواند و آنها را در یک آرایه یا بردار ذخیره کند. سپس معدل هر دانشجو را به‌همراه نام او به خروجی بفرستد. همچنین معدل کلاس را در هر درس محاسبه نماید.

۱۲.برنامه‌ای به زبان C++ بنویسید که یک عدد را به‌صورت رشته‌ای شامل ۱۰ رقم بخواند.(ممکن است علامت نقطه نیز به‌عنوان ممیز عدد اعشاری وارد شود.) سپس عدد معادل این رشته را در یک متغیر از نوع float ذخیره کند و به خروجی ببرد.

۱۳.برنامه‌ای بنویسید که رشته‌ای را از ورودی خوانده تمام کاراکترهای تکراری رشته را حذف کند.

۱۴.برنامه‌ای بنویسید که تقارن یک رشتهٔ وارد شده توسط کاربر را تشخیص دهد.

۱۵.برنامه‌ای بنویسید که رشته‌ای را بدون محدودیت در طول رشته خوانده و ذخیره کند. سپس موارد زیر را با استفاده از توابع کتابخانه‌ای مشخص نماید:

الف- تعداد حروف کوچک

ب- تعداد حروف بزرگ

ج- تعداد حروف صدا دار

د- تعداد ارقام موجود در رشته

و- تعداد کاراکترهای ویرایشی

هـ- تعداد کاراکترهای فضای خالی

۱۶.برنامه‌ای بنویسید که رشته‌ای را به همراه کاراکتر ch و عدد n از ورودی بخواند. سپس محل n-اُمین وقوع کاراکتر ch را در رشته مشخص نماید.

۱۷.فرمول محاسبهٔ آدرس را برای عنصر $H[i_1][i_2]\ldots[i_n]$ بیابید که H یک آرایهٔ n بعدی به‌صورت زیر است:

```
Element_Type H[N1][N2]…[Nn] ;
```

میزان حافظهٔ هر عنصر برابر است با C = sizeof (Element_Type)

۱۸.اگر در یک ماتریس پایین مثلثی تنها عناصر قطر اصلی و زیر آن را در ذخیره کنیم در حافظه صرفه‌جویی شده است. فرمول محاسبهٔ آدرس را برای m[i][j] به ازای i >= j ارائه دهید. فرض کنید هر عنصر m در ۸ بایت ذخیره می‌شود و نحوهٔ ذخیره نیز سطری است و آدرس پایه نیز b می‌باشد.

۱۹.مسئلهٔ ۱۸ را برای یک ماتریس بالا مثلثی تکرار کنید.

۲۰.ماتریس سه قطری یک ماتریس مربعی مثل m است که به جزء عناصر قطر اصلی و قطر بالای قطر اصلی و قطر پایین قطر اصلی بقیه صفر هستند. یعنی به ازای 1 < | i–j | ، m[i][j] برابر با صفر است. الگوی مناسبی را برای ذخیرهٔ این ماتریس در نظر بگیرید و فرمول محاسبهٔ آدرس آن را پیدا کنید.

۷-۸ : موارد مطالعاتی

۱. (پروژه برنامه نویسی): برنامه‌ای بنویسید که کلیهٔ اعمال چهارگانهٔ ریاضی را برروی اعداد صحیح و اعشاری با طول‌های بسیار بزرگ امکانپذیر سازد.

۲. (پروژه برنامه‌نویسی) : جهت محاسبهٔ معکوس یک ماتریس مانند ماتریس **M** در شکل زیر، می‌توان ابتدا ماتریس افزودهٔ **M** را تشکیل داد و سپس با انجام اَعمال سطری مقدماتی (که در هندسهٔ تحلیلی در دورهٔ پیش‌دانشگاهی فراگرفته‌اید) کاری کرد که به جای ماتریس **M**، یک ماتریس واحد معادل آن قرارگیرد. در این صورت آنچه که به‌جای درایه‌های ماتریس افزوده پدید می‌آید همان معکوس ماتریس **M** می‌باشد. به مثال زیر توجه کنید.

$$\begin{bmatrix} 3 & 2 & -4 & 1 & 0 & 0 \\ 1 & -5 & 2 & 0 & 1 & 0 \\ -2 & 3 & 1 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \frac{1}{15} \times \begin{bmatrix} 15 & 0 & 0 & 11 & 14 & 16 \\ 0 & 15 & 0 & 5 & 5 & 10 \\ 0 & 0 & 15 & 7 & 13 & 17 \end{bmatrix}$$

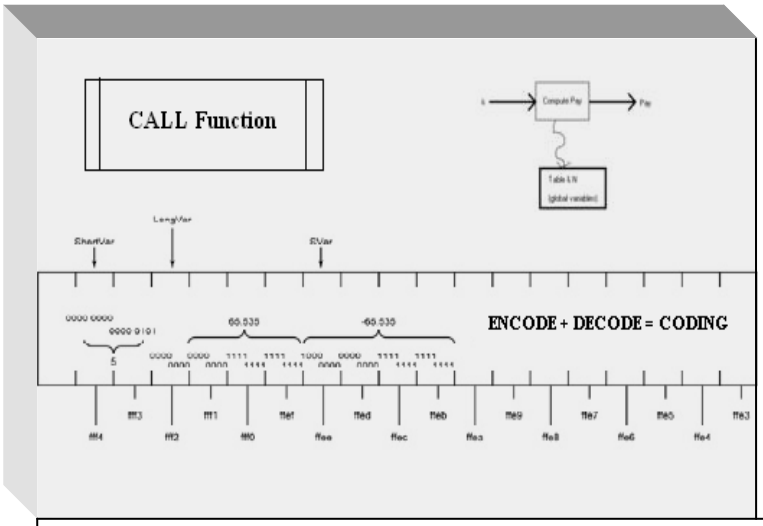
برنامه‌ای به زبان C++ بنویسید که اعمال جمع و تفریق و ضرب را برروی دو ماتریس با طول دلخواه انجام دهد. همچنین قادر به محاسبهٔ دترمینان، ترا نهاده و معکوس یک ماتریس مربعی باشد.

۳. در مورد نحوهٔ استفاده از کلیدهای **F1** تا **F12** و یا کلیدهای بالا و پایین و چپ و راست و یا کلیدهای ترکیبی تحقیق کنید. سپس در پروژه‌های فوق از این کلیدها استفاده کنید.

۴. سرفایل **conio.h** مربوط به سیستم عامل **Dos** می‌باشد. لذا توابع وابسته به این هدرفایل را نمی‌توان در کامپایلرهای مربوط به لینوکس به کار برد. تحقیق کنید که چگونه می‌توان توابعی چون **getch** را شبیه‌سازی کرد یا معادل آنها را برای برنامه‌نویسی در **Kdevelop** پیدا کنید.

۵. به نظر شما آیا تعریف یک آرایه با طول متغیر به‌صورت زیر امکانپذیر است؟

```
int n;
  cin>>n;
const int m=n;
  int a[m];
```



فصل هشتم

مفهوم رویّه در الگوریتم، مقدّمه ای بر رمزنگاری و نمودار N-S

اهداف فصل و چکیده مطالب :

۱. اهمیت تجزیهٔ گام‌به‌گام در حل مسائل
۲. آشنایی با روش حل کل به جزء
۳. پیدایش مفهوم رویه(تابع) در الگوریتم
۴. نگاره‌سازی و رسم نمودار در محیط متنی
۵. روش‌های رمزگذاری با جانشینی
۶. رمزگذاری به روش جایگشت
۷. نمودار N-S

۸-۱ : تجزیهٔ گام به گام

در فصول قبل با نحوهٔ ایجاد کارنما و بسیاری از جعبه‌های متداول الگوریتم‌نویسی آشنا شدید. یاد گرفتید که چگونه یک مسئله را قدم به قدم پیش ببرید و از ساده‌ترین حالت تلاش خود را آغاز کنید تا آن‌که در نهایت موفق به حل حالات پیچیده‌تر و بزرگتری گردید. با الگوریتم‌های ساخت‌یافته آشنا شدید و دیدید که چگونه جعبه‌های تکرار ما را در خوش ساخت‌کردن کارنماها یاری می‌دهند. همچنین مشاهده‌کردید که چگونه کارنماها را اصلاح کنیم تا تنها از یک نقطه به سمت دکمهٔ پایان بروند. همهٔ این تلاش‌ها در جهت خوش ساخت کردن الگوریتم ها مؤثر بودند. اما در این فصل می‌خواهیم تولید الگوریتم های ساخت یافته را از منظری دیگر مد نظر قرار دهیم. در این فصل از ایدهٔ "تفرقه بینداز و حکومت کن" استفاده خواهیم کرد، و یاد خواهید گرفت که چگونه یک مسئلهٔ بزرگ را به زیر مسئله های کوچکتری تقسیم کنید و به چه شکلی زیر مسئله‌ها را در کنار یکدیگر قرار دهید تا منجر به تولید حلی برای یک مسئلهٔ بزرگتر گردد.

روش حل کل به جز

در روش حل کل به جزء در ابتدا یک مسئله را با دیدی کل نگرانه حل می‌کنیم. در این مرحله تنها کارهای کلی که برای حل مسئله باید انجام گیرد را نام می‌بریم و به جزئیات پیاده‌سازی الگوریتم هیچ کاری نداریم. در مرحلهٔ بعد برای هریک از قسمت‌هایی که در مرحلهٔ نخست نام برده‌ایم یک الگوریتم مجزا در نظر می‌گیریم که به آن زیر الگوریتم گفته می‌شود. البته ممکن است یک زیر الگوریتم نیز از زیر الگوریتم های دیگری تشکیل شود. به‌عنوان مثال مسئله محاسبهٔ مجموع تمامی درایه های یک ماتریس را در نظر بگیرید. در شکل ۸-۱ چگونگی حل این مسئله را به‌واسطهٔ روش کل به جز می‌بینید.

چنانکه در شکل ۸-۱ می‌بینید روند اصلی الگوریتم از دکمهٔ **start** در شکل(الف) آغاز شده و در جعبه‌های ۱ و ۲ طول و عرض ماتریس و سپس درایه‌های آن خوانده می‌شود. مفهوم جعبهٔ ۳ که جدید به نظر می‌رسد این است که زیر الگوریتم کناری آن (یعنی جعبه‌های ۵ تا ۸) به جای جعبهٔ ۳ قرار گیرد. بنابراین می‌توانید چنین تصور کنید که جعبهٔ ۳ برداشته می‌شود و جعبه‌های ۵ تا ۸ به جای آن قرار می‌گیرند. چنانکه در شکل ۸-۱ نیز مشاهده می‌کنید زیر الگوریتم قسمت (ب) خود از یک زیر الگوریتم دیگر تشکیل شده است به‌طوری که در قسمت (ب) نیز می‌توانید چنین تصور کنید به‌جای جعبهٔ ۷ جعبه‌های ۹ تا ۱۱ قرار می‌گیرد. لذا مسئلهٔ جمع تمامی درایه‌های یک ماتریس در ابتدا به دو مسئلهٔ ذخیره‌سازی ماتریس در حافظه و سپس جمع درایه‌ها و چاپ مجموع به‌دست‌آمده تقسیم شد که در زیر نشان داده شده است.

(الف) مسئله: جمع تمامی درایه‌های یک ماتریس:

۱) طول و عرض ماتریس را دریافت کن.

۲) درایه‌های ماتریس را بخوان و در آرایهٔ **A** ذخیره کن.

۳) مجموع درایه‌ها را محاسبه کن و در متغیر **total** قرار بده.

۴) مقدار **total** را به‌عنوان مجموع محاسبه شده چاپ کن.

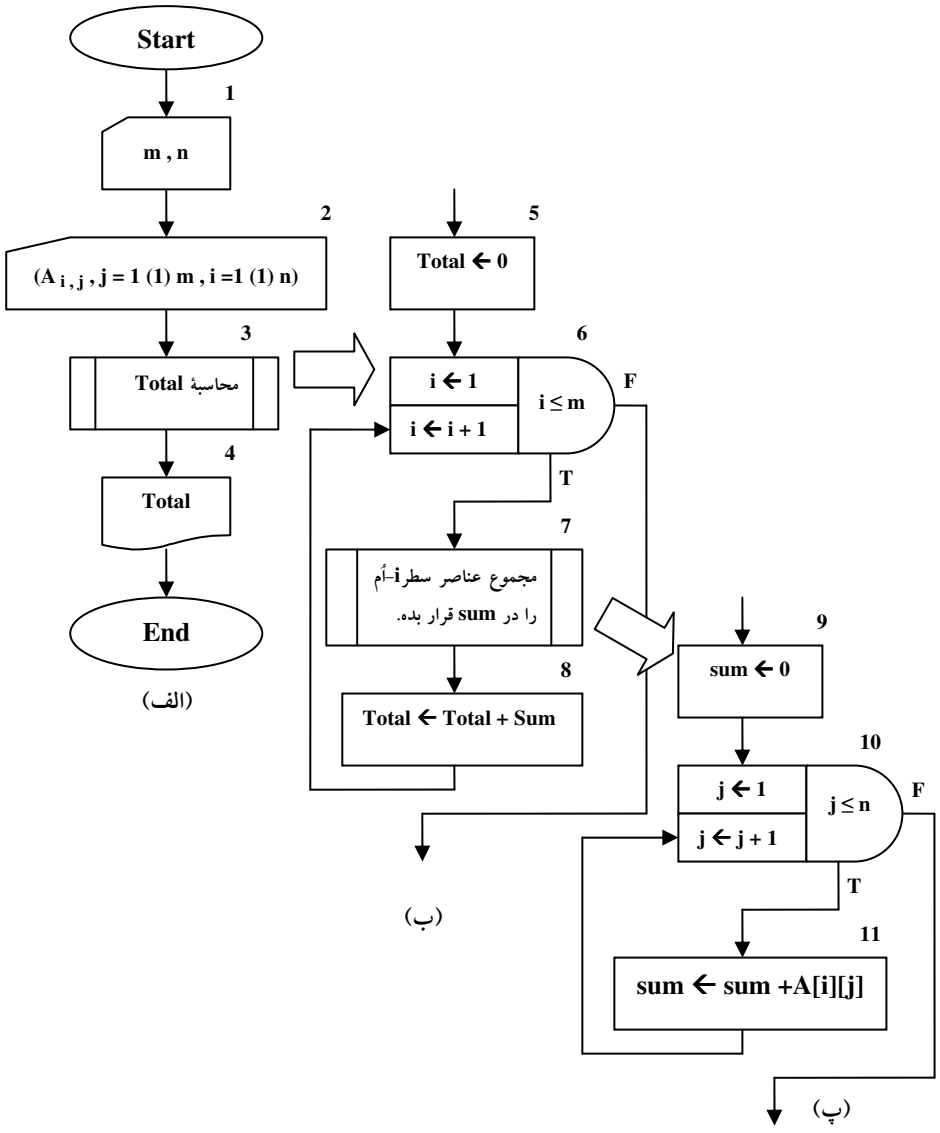
(ب) در دومین مرحله نیز مسئلهٔ محاسبهٔ مقدار **total** به نوبهٔ خود به مسئله محاسبهٔ حاصل جمع مجموع درایه‌های هر سطر ماتریس تبدیل می‌شود، که به شرح زیر است:

۵) مقدار **total** را با صفر مقدار اولیه بده.

۶) در یک حلقهٔ تکرار مراحل ۷ و ۸ را تکرار کن.

۷) مجموع عناصر هر سطر را محاسبه کن.

۸) مجموع عناصر هر سطر را به **total** اضافه کن.



شکل ۸-۱ : جمع تمامی درایه‌های ماتریس با روش کل به جزء

(ج) و در آخرین مرحله مسئلهٔ محاسبهٔ مجموع عناصر هر سطر را به‌واسطهٔ یک حلقه حل کردیم که شامل مراحل زیر می شود:

۹) مقدار **SUM** را با صفر مقدار اولیه بده.

۱۰) در یک حلقهٔ تکرار مرحلهٔ ۱۱ را برای کلیه عناصر سطر جاری انجام بده.

۱۱) هر درایه از سطر جاری را به متغیر **sum** بیفزای.

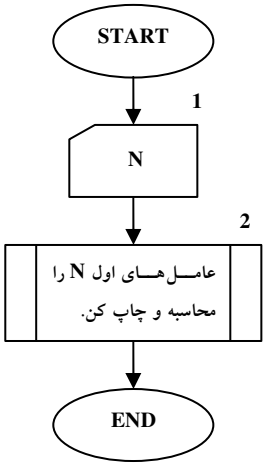
روندی که ملاحظه کردید به روشنی در شکل۸-۱ نشان داده شد. این ایده که یک مسئله را به مسائل کوچکتری تقسیم‌کنیم ما را کمک می‌کند تا به‌جای رویارویی با یک مسئلهٔ بزرگ با تعدادی مسئلهٔ کوچکتر سروکار داشته باشیم.

الگوریتم تجزیهٔ یک عدد به عامل‌های اول

به‌عنوان یک مثال دیگر از حل مسئله به روش کل به جزء می‌خواهیم الگوریتم تجزیهٔ یک عدد به عامل‌های اول را بررسی کنیم.

چنانکه پیشتر در ریاضیات دیده‌اید هر عدد یا اول است و یا مرکب. اعداد مرکب را می‌توان به‌صورت حاصل ضرب تعدادی عدد اول نوشت، که به این اعداد عامل‌های اول عدد گفته می‌شود. حال می‌خواهیم الگوریتمی ارائه کنیم که عامل‌های اول یک عدد مرکب را محاسبه و چاپ کند.

در نخستین مرحله از ساخت این الگوریتم می‌توان کارنمای کلی زیر را در نظر گرفت:



شکل ۸-۲ - الف

چنانکه در شکل۸-۲- الف می‌بینید هنوز هیچ روشی درخصوص نحوهٔ یافتن عوامل اول **N** درنظر نگرفته‌ایم. ولی به‌راحتی مبادرت به ساخت یک الگوریتم کلی کرده‌ایم. حال نوبت آن رسیده که به ساخت زیر الگوریتم‌هایی برای کارهای مطرح شده در کارنما شکل ۸-۲-الف بپردازیم. لذا ابتدا باید روشی را برای محاسبهٔ عوامل اول **N** پیدا کنیم.

پیشتر در بخش ۴-۳ با مسئلهٔ یافتن هم عامل‌های یک عدد صحیح آشنا شدید. با توجه به آنکه هر دو مسئله برروی عامل‌های یک عدد صحیح بحث می‌کنند، احتمالاً شباهت‌هایی می‌توانند به هم داشته باشند و ممکن است بتوان در حل این مسئله از تجارب به‌دست آمده در حل مسئلهٔ یافتن هم‌عامل‌های یک عدد صحیح استفاده کرد. به‌عنوان مثال هم عامل‌های عدد ۶۰ را در نظر بگیرید که در زیر به‌صورت زوج‌های مرتب لیست شده‌اند:

(۱۰ و ۶) ، (۱۲ و ۵) ، (۱۵ و ۴) ، (۲۰ و ۳) ، (۳۰ و ۲) ، (۶۰ و ۱)

همچنین تجزیهٔ عدد ۶۰ به عامل‌های اول به‌صورت زیر است:

$$۶۰ = ۲ \times ۲ \times ۳ \times ۵$$

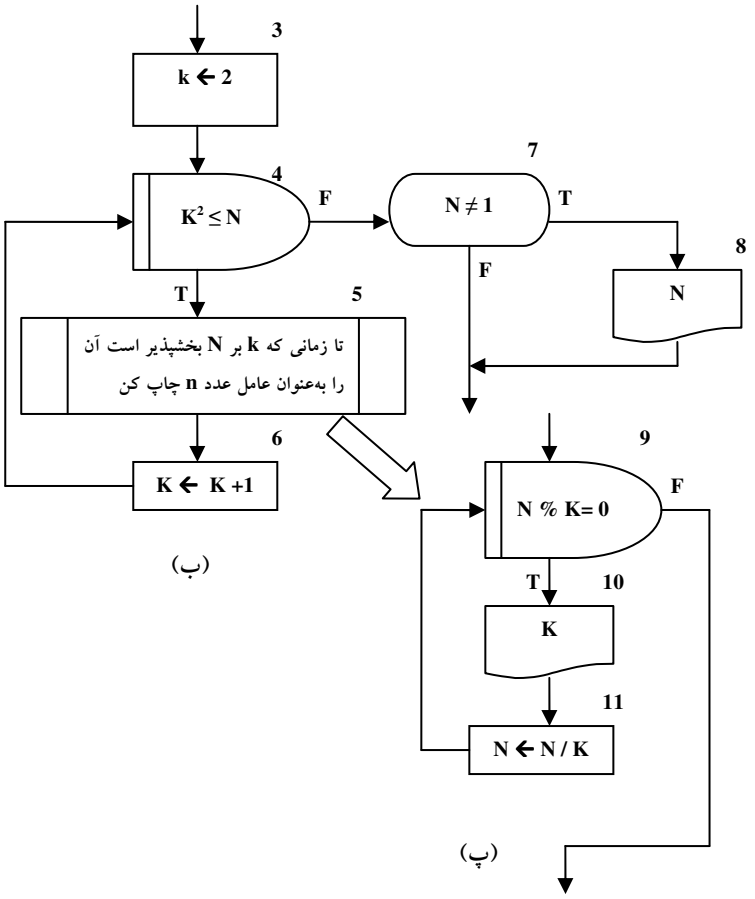
اگر توجه کرده باشید ترکیب عامل‌های اول عدد ۶۰ هم‌عامل‌های آن را به‌دست می‌دهد. به‌عنوان مثال ۵×۳ و ۲×۲ همان زوج مرتب (۱۵ و ۴) را می‌سازند. بنابراین ارتباط بسیار نزدیکی بین این دو مسئله وجود دارد. اما چگونه عامل های اول عدد ۶۰ را در ریاضیات دورهٔ دبیرستان پیدا می‌کردیم. برای یافتن روشی برای حل مسئلهٔ عامل‌های اول یک عدد، در زیر روند تجزیهٔ عدد ۶۰ به عامل‌های اول آن، نشان داده شده است:

شمارهٔ مرحله	عددی که قرار است تجزیه شود N	مقسوم علیه آزمایشیK	عامل‌های اول پیدا شده
۱	۶۰	۲	۲
۲	۳۰	۲	۲
۳	۱۵	۳	۳
۴	۵	۴	–
۵	۵	۵	۵
۶	۱	–	–

چنانکه در این مسئله می‌بینید در هر لحظه از محاسبات، یک عدد N داریم، که در حال محاسبهٔ عوامل اول آن هستیم، و یک مقسوم‌علیه داریم، که بخشپذیری N را بر آن عدد آزمایش می‌کنیم. اگر به روند تجزیه توجه کرده باشید ابتدا عدد ۲ را به‌عنوان مقسوم‌علیه آزمایشی در نظر گرفته‌ایم و تا زمانی که عدد در حال تجزیه بر ۲ بخشپذیر باشد، تقسیم را انجام داده و مقدار N/K که همان باقی‌ماندهٔ عدد است که هنوز به عوامل اول تجزیه نشده را، جهت تجزیه در مرحله بعد، به‌عنوان عدد N قرار می‌دهیم. پس از آنکه دیگر عدد N بر مقسوم‌علیه آزمایشی بخشپذیر نبود مقدار مقسوم‌علیه آزمایشی را یک واحد می‌افزاییم. این روند تا جایی که مقدار N برابر ۱ بشود ادامه می‌یابد. اما در حقیقت هیچ نیازی نیست که عمل تقسیم تا زمانی‌که N برابر یک گردد ادامه یابد، چرا که در الگوریتم هم‌عامل‌های یک عدد به یاد دارید که عدد N نمی‌تواند دارای مقسوم علیه‌ای بزرگتر از $\sqrt{N}$ باشد. لذا هنگامی که شرط

$$K^2 \le N$$

برقرار نباشد می‌توان گفت تنها عامل اول باقی‌مانده در صورتی که N≠1 باشد برابر خود N می‌باشد. لذا می‌توان الگوریتم شکل ۸-۲- الف را به‌واسطهٔ زیرالگوریتم‌های زیر تکمیل کرد:



ادامه شکل ۸-۲ : تجزیهٔ گام‌به‌گام مسئلهٔ یافتن عامل‌های اول یک عدد

کار در کلاس ۸-۱ :

اگر توجه کرده باشید غیر از ۲ هیچ عدد زوج دیگری اول نیست. درحالی‌که ما بی‌جهت در این الگوریتم در مثال عامل‌های اول عدد ۶۰ ، مقدار **K** *را برابر ۴ اختیار کردیم و به زیر الگوریتم (پ) رفتیم. پیشنهادی ارائه کنید که از انجام محاسبات بیهوده در الگوریتم فوق جلوگیری کند و سرعت اجرای روندنما بیشتر گردد.*

کار در کلاس ۸-۲ :

کارنمایی رسم کنید که الگوریتم مرتب سازی حبابی را به روش تجزیهٔ گام به گام بر روی لیستی از اعداد به طول n *عنصر پیاده سازی کند.*

۸-۲: تمرین

۱. در ریاضیات یک مجموعهٔ مرتب از اعداد را بردار می‌نامند. ملاحظه می‌شود که آن چیزی را که ریاضیدانان بردار می‌گویند ما لیست مقادیر عددی می‌خوانیم. حاصل‌ضرب دو بردار (با طولی مساوی **M**) را می‌توان به طریق زیر تعریف کرد.

$$A \otimes B = A[1]\times B[1] + A[2]\times B[2] + \ldots A[M]\times B[M]$$

عامل ⊗ نشان‌دهندهٔ ضرب بردارها است درحالی‌که × نشان دهندهٔ ضرب معمولی دو عدد است. توجه کنید که نتیجهٔ ⊗ یک عدد است.

این مفهوم را می‌توان برای تعریف ضرب یک ماتریس در یک بردار تعمیم داد. اگر **C** یک ماتریس $M \times N$ (**M** سطر و **N** ستون)، و **A** برداری به‌طول **N** باشد، آنگاه طبق تعریف $A \otimes C$ عبارت است از بردار **D** به‌طول **M** که به طریق زیر محاسبه می‌شود.

D[1] حاصل‌ضرب برداری(یعنی عمل ⊗) سطر اول **C** و **A** است ،

D[2] حاصل‌ضرب برداری سطر دوم **C** و **A** است،

⋮

D[M] حاصل‌ضرب برداری سطر**M**- اُم **C** و **A** است.

الف- مقدار عبارت:

$$\begin{bmatrix} 6 & 2 \\ -4 & 5 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

را بنویسید.

ب- قطعهٔ روندنمایی رسم کنید که با فرض اینکه قبلاً به **M** و **N** ، بردار **A** و ماتریس **C** مقادیری منسوب شده باشد، $A \otimes C$ را محاسبه و نتیجه را به **D** منسوب کند. در ترسیم این روندنما از روش تجزیهٔ گام به گام استفاده کنید.

۲. با استفاده از روش تجزیهٔ گام به گام، روندنمایی برای الگوریتمی رسم کنید که تعیین کند آیا یک ماتریس مربعی مفروض، یک «مربع جادویی» است یا نه. مربع جادویی ماتریسی است که حاصل جمع عناصر زیر مجموعه‌های معینی از آن مساوی باشد. بالاخص، یک ماتریس مربعی را جادویی می‌گوییم اگر مجموع عناصر هر سطر، ستون و قطر اصلی آن یک مقدار باشد. مثلاً، الگوریتم شما باید کارهای زیر را انجام دهد:

الف- **n**، «مرتبه» (یعنی تعداد سطرها و ستون‌های) مربع جادویی مورد آزمون را به داخل بیاورد.

ب- تمام درایه‌های ماتریس $n\times n$ را به داخل بیاورد.

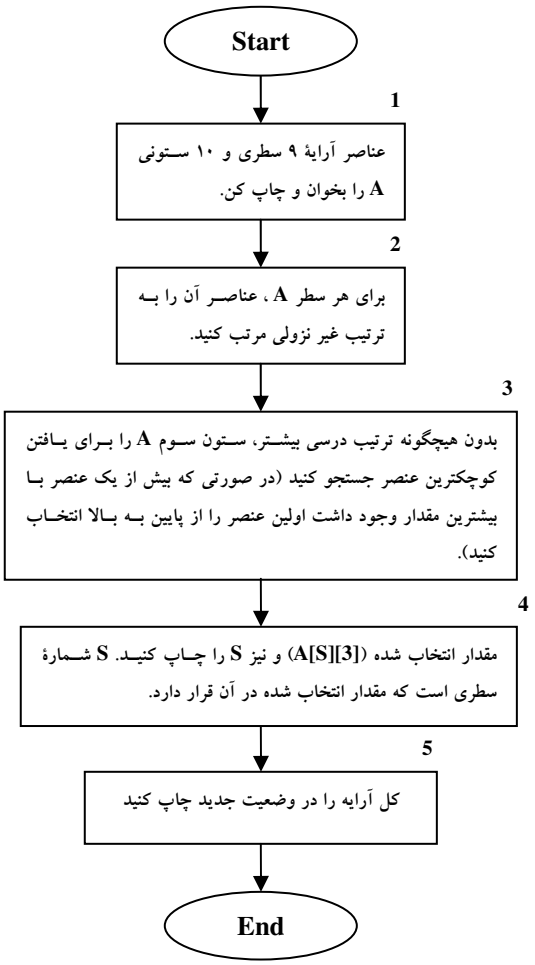
ج- یک نسخه از ماتریس را در خروجی بنویسد.

د- تشخیص دهد که ماتریس خوانده شده، جادویی است یا نه. (در هر مورد پیغام مناسبی چاپ کند)

ه- به قسمت (أ) برای امتحان مربع دیگری باز گردد.

$$\begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \quad \begin{bmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \end{bmatrix} \quad \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

۳. الگوریتم داده شده در شکل پایین را مطالعه کنید.



شکل ۸-۳ : کارنمای تمرین ۳

الف- اولین تکلیف شما، تهیه گام‌های جزئی روندنمایی معادل جعبهٔ ۲ است.

ب- تکلیف بعدی، تهیه گام‌های جزئی روندنمایی معادل جعبه‌های ۳ و ۴ و ۵ است.

توجه کنید که قسمت‌های (الف) و (ب) فوق از نظر منطقی از یکدیگر مستقل‌اند. یعنی روش جستجو در جعبهٔ ۳ بستگی به اینکه داده‌ها قبلاً مرتب شده‌اند یا نه ندارد. در روندنماهای خود تا حد امکان از جعبه‌های تکرار استفاده کنید.

۴. توضیح دهید که چگونه در جعبهٔ ۷ شکل ۸-۲- ب می‌توانیم **N=1** داشته باشیم.

۵. روندنمای صفحهٔ بعد را مطالعه کنید و به پرسش‌های زیر پاسخ دهید:

الف- اگر اجرا به جعبهٔ ۵ برسد، آیا **n** یک عدد حقیقی است یا صحیح؟چرا؟

> در پاسخ‌دهی به قسمت‌های (ب) تا (ز)، فرض کنید مقدار **n** برابر ۲۵ است.

ب- مقادیری را که در جعبهٔ ۶ به زیرنویس **j** منسوب می‌شوند، بنویسید.

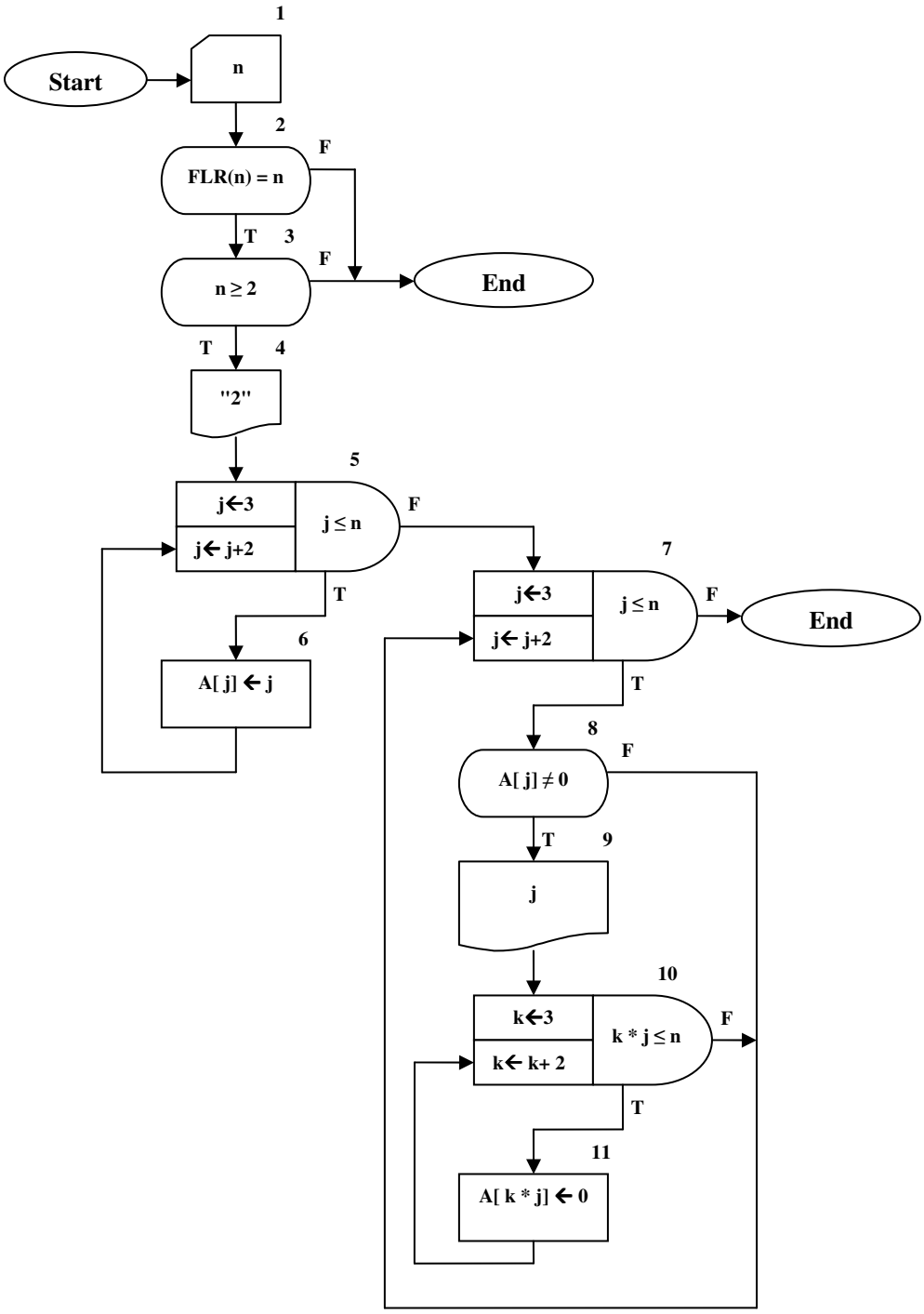
ج- مقادیری را که در جعبهٔ ۶ به **A[j]** منسوب می‌شوند، بنویسید.

د- اگر جعبهٔ ۹ را با چاپ مقدار «۵ » برای **j** به پایان ببریم، **A[j]** در جعبهٔ ۱۱ برای چه مقدار زیرنویسی مساوی صفر می‌شود؟

و- ۷ مقدار اولی که توسط الگوریتم چاپ می‌شوند کدام‌اند؟

ه - الگوریتم چه کاری انجام می‌دهد؟

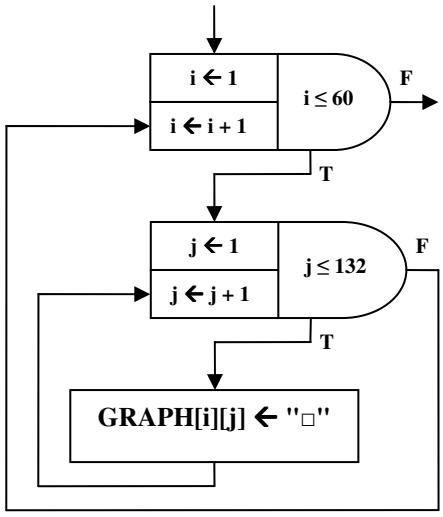
ز- برای **k** در جعبهٔ ۱۰، مقدار اولیه‌ای پیشنهاد کنید که بر کارایی الگوریتم از طریق حذف احتمال بیش از یکبار قرار دادن مقدار صفر در یک **A[j]** ، بیفزاید.



شکل ۸-۴ : کارنمای تمرین ۵

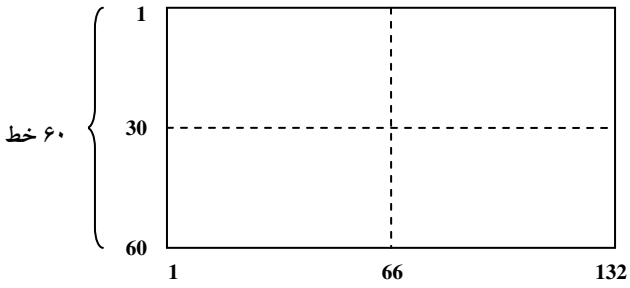
۸–۳ : نگاره سازی با کامپیوتر

فرض کنید می‌خواهیم نمودار یک تابع ریاضی را توسط کامپیوتر رسم کنیم. یک راه ساده برای این کار، استفاده از آرایه است. این آرایه را **GRAPH** می‌نامیم. چون چاپگر خطی تقریباً ۶۰ خط در هر صفحه و ۱۳۲ نویسه (کاراکتر) در هر خط چاپ می‌کند، فرض کنید **GRAPH** دارای ۶۰ سطر و ۱۳۲ ستون باشد. با در دست داشتن چنین آرایه‌ای می‌توان کد اسکی هر کاراکتر خواسته شده را به ۱۳۲ × ۶۰ عنصر آن نسبت داد. برای شروع، باید مطمئن بود که آرایه، خالی و «پاک» است. تجربهٔ خوبی است که کار را با پاک کردن آرایه شروع کنیم. این کار را می‌توان با دو حلقهٔ زیر که نویسهٔ فضای خالی یعنی □ را به تمام عناصر **GRAPH** منسوب می‌کند، انجام داد.



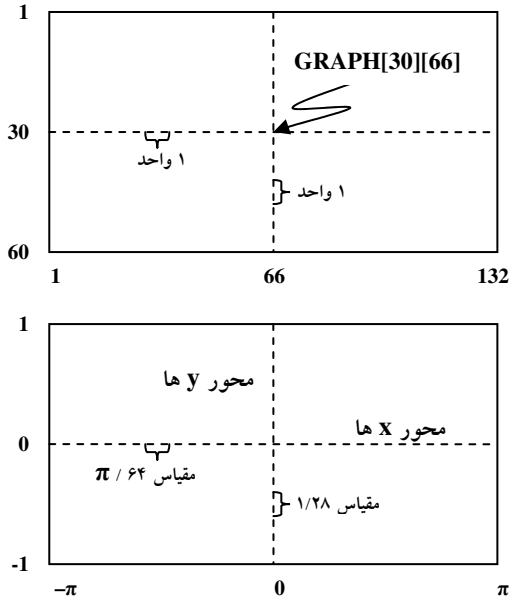
شکل ۸–۵ : کارنمای پاک کردن آرایه

سپس، باید محورها را رسم کنیم. فرض کنید مبدأ را در مرکز صفحه، مثلاً سطر ۳۰ و ستون ۶۶ قرار دهیم. می‌توانیم از نویسهٔ خط فاصله «- » برای رسم محور **x**ها استفاده کنیم. به همین نحو، برای رسم محور **y** ها می‌توانیم « ا » را به تمام عناصر ستون شصت و ششم نسبت دهیم.



شکل ۸–۶ : آرایهٔ GRAPH با محورهای مختصات

فرض کنید تابعی که می‌خواهیم منحنی آن را رسم کنیم $y = \sin(x)$ است و می‌خواهیم قلمرو آن $-\pi \le x \le \pi$ باشد. باید ۱۳۲ زیر نویس مربوط به ستون‌های ماتریس GRAPH را طوری تعبیر کنیم که با حدود 2π واحد برروی نمودار متناظر شود. اگر واحد بین دو زیرنویس افقی را π / ۶۴ بگیریم، آنگاه ۱۲۸ تا از این واحدها 2π خواهد بود. این تعبیر را تعیین مقیاس می‌گوییم. در جهت عمودی، ۶۰ زیرنویس (واحد) داریم و برد تابع سینوسی $-1 \le y \le 1$ است. اگر یک واحد عمودی را ۱/۲۸ بگیریم آنگاه ۵۶ تا از این واحدها نشان‌دهندهٔ برد کامل تابع سینوسی خواهد بود. مقیاس عمودی، ۱/۲۸ در هر واحد است.



شکل ۸-۷ : آرایهٔ GRAPH با مقیاس‌های افقی و عمودی

حال ببینیم چگونه می‌توان یک نقطه را روی نمودار «ترسیم» کنیم. فرض کنید $x = \pi/4$. تابع کتابخانه‌ای سینوس مقدار $y = \sin(\pi/4) =$ ۰/۷۰۷۱۱ را محاسبه خواهد کرد. می‌خواهیم علامت خاصی مثل ستاره (*) را به یکی از عناصر ماتریس نسبت دهیم که نشان دهندهٔ زوج (۰/۷۰۷۱۱ و $\pi/4$) باشد. ولی کدام عنصر ماتریس باید ستاره را دریافت کند؟ برای تبدیل $\pi/4$ به واحد ماتریسی باید آن را به $\pi/64$ تقسیم کنیم که نتیجه عبارت می‌شود از ۱۶ واحد در سمت راست مبدأ مختصات. بنابراین با در نـظر گرفتن مقیاس محور x ها، $\pi/4$ متناظر است با ۸۲ = ۶۶ + ۱۶ به‌عنوان زیرنویس ستون در نماد ماتریسی است. (۶۶ را چون زیرنویس ستون مربوط به مبدأ مختصات است، اضافه می‌کنیم.)

حال باید سطر نظیر مقدار y ، یعنی ۰/۷۰۷۱۱ ، را در ماتریس پیدا کنیم. ابتدا مقدار y را به ۱/۲۸ تقسیم می‌کنیم تا به واحد زیرنویس ماتریس تبدیل شود. ولی از آنجا که زیر نویس ماتریس باید عدد صحیح باشد، مقدار به‌دست آمده را به نزدیک‌ترین عدد صحیح گرد می‌کنیم.

بنابراین با اضافه کردن ۰/۵ و سپس قطع کردن، خواهیم داشت FLR(۰/۷۰۷۱۱ × ۲۸ + ۰/۵) . در اینجا به این نکته توجه می‌کنیم که تابع سینوسی در جهت بالا مثبت است درحالی‌که سطرهای ماتریس در جهت پایین دارای شماره های مثبت هستند. بنابراین برای محاسبهٔ زیرنویس سطر عنصری از ماتریس که باید به آن ستاره نسبت داد، مقدار y را که در بالا محاسبه شد از شمارهٔ سطر مبدأ مختصات یعنی ۳۰ کم می‌کنیم.

بنابراین سه دستورالعمل انتسابی که برای رسم نقطه لازم است عبارت‌اند از:

```
row ← 30 – FLR(sin(x) × 28 + 0. 5)
column ← x × (64/ π) + 66
GRAPH[row][column] ← '*'
```

شکل ۸- ۸ : دستورات انتساب لازم برای رسم نمودار

اگر به x تمام مقادیر بین π- و π را با قدم‌های π/۶۴ نسبت دهیم و به ازای هر مقدار x انتساب‌های بالا را اجرا کنیم، آنگاه با اجرای دستور زیر می‌توانیم نمودار را چاپ کنیم.

```
( ( GRAPH [i][j] ; j=1 (1) 132 ) i= 1 (1) 60)
```

شکل ۸- ۹ : چاپ محتویات آرایهٔ GRAPH

با تغییر مناسب در تابع و در صورت لزوم تغییر ضرایب مقیاس و مبدأ مختصات، می‌توان بسیاری توابع دیگر را نیز در محدودهٔ موردنظر رسم کرد. تمرین‌های بخش ۸-۴ در شروع این کار به شما کمک خواهند کرد.

۸-۴: تمرین

۱. روندنمایی بسازید که مطابق آنچه در متن فوق تشریح شد، محورهای مختصات را برروی آرایه GRAPH «رسم» کند به‌طوری که مبدأ مختصات در ردیف ۳۰- اُم و ستون ۶۶- اُم واقع شود.

۲. با به‌کار گرفتن حل تمرین ۱، روندنمایی بسازید که ماتریس رسم نمودار را پاک کند، محورهای مختصات را رسم و نمودار y=sin(x) را به ازای $-\pi \leq x \leq \pi$ ترسیم کند. اگر کامپیوتری در اختیار دارید، برنامه‌ای به زبان C++ برای روندنمای خود را نوشته و آن اجرا کنید.

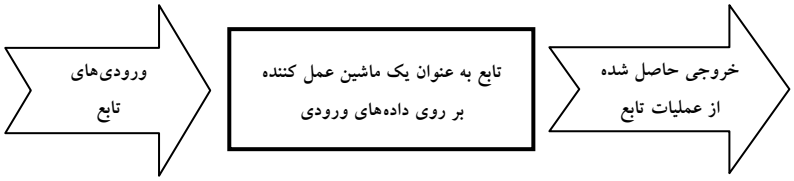
۳. روندنمای تمرین ۲ را طوری تغییر دهید که نمودار y=cos(x) را به ازای $-\pi \leq x \leq \pi$ ترسیم کند. برای رسم نمودار از علامت دیگری غیر از ستاره استفاده کنید.

۴. پس از اتمام تمرین‌های ۲ و۳ روندنمایی بسازید که هر دو نمودار y=sin(x) و y=cos(x) را به ازای $-\pi \leq x \leq \pi$ روی یک صفحهٔ مختصات مشترک و با استفاده از نمادهای متفاوتی برای هر نمودار، ترسیم کند.

۵. روندنمای تمرین۴ را طوری تغییردهید که نه تنها y=sin(x) وy=cos(x) بلکه y=sin(x) + cos(x) را نیز روی همان صفحهٔ مختصات و با استفاده از سه نماد مختلف رسم‌کند. در این مورد لازم است مقیاس محورyها را تغییرداد زیرا برد y=sin(x)+cos(x) برابر $-\sqrt{2} \leq y \leq \sqrt{2}$ است. مقیاس عمودی را از ۱/۲۸ به ۱/۱۴ تغییر دهید. روندنمایی بسازید که این نمودارها را در قلمرو $-\pi \leq x \leq \pi$ رسم کند و برنامهٔ آن را روی کامپیوتر اجرا کنید.

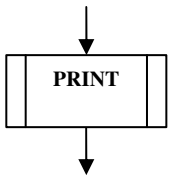
۸– ۵ : گسترش تجزیهٔ الگوریتم ها و پیدایش مفهوم رویه (تابع)

با مفهوم تابع در ریاضیات دورهٔ دبیرستان آشنا شده‌اید. اما مفهوم تابع در الگوریتم و برنامه‌نویسی کمی کلی‌تر از مفهوم تابع در ریاضیات است. اما با این حال می‌خواهیم شما را متوجه یک مفهوم اساسی در خصوص تابع بکنیم. «تابع را می‌توان همانند ماشینی در نظر گرفت که تعدادی ورودی دارد و بر حسب ورودی‌های خود عکس‌العمل نشان داده، و با عملیات برروی ورودی‌ها، خروجی مناسب را تولید می‌کند.» در شکل ۸–۱۰ نمادی برای این تعریف از تابع نشان داده شده است.



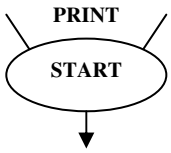
شکل ۸–۱۰ : نمادی برای تابع

اینگونه تعریف کردن تابع می‌تواند بسیار از مشکلات ما را در جهت فهم مفهوم رویه (تابع) در الگوریتم یاری دهد. در حقیقت مفهوم رویه یاتابع چیزی جزء همان مفهوم زیرالگوریتم نیست. اما چنانکه از تعریف اخیر برمی‌آید نباید از تابع در الگوریتم انتظار داشته باشیم که حتماً عددی را به ازای ورودی‌ها به ما برگرداند بلکه ممکن است خروجی تابع شامل چاپ پیامی در صفحه نمایش باشد. همچنین تابع در الگوریتم لزوماً نیازی به ورودی ندارد. بنابراین تابع را به‌عنوان یک زیرالگوریتم در نظر بگیرید. هر تابع نیازمند یک نام برای فراخوانی آن به جهت عمل کردن آن است. لذا توابع ما باید دارای یک نام منحصر به فرد باشند. هنگامی که می‌خواهیم یک زیر الگوریتم در قسمتی از یک کارنما اجرا گردد کافی است نام تابع را در جعبه‌ای همانند شکل زیر موسوم به «جعبهٔ فراخوانی تابع» ذکر کنیم. به‌عنوان مثال در زیر تابعی با نام **PRINT** را فراخوانی کرده‌ایم:

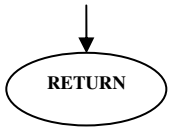


شکل ۸–۱۱ : نمونه‌ای از فراخوانی تابع

همچنین برای آنکه زیرالگوریتم مربوط به یک تابع به وضوح قابل شناسایی باشد و از طرفی برای اینکه هر رویه یک زیر روندنمای کامل، که خودش هویت خویش را معلوم کند بسازیم، در آغاز زیر الگوریتم یک جعبهٔ شروع همراه با نام تابع به‌صورتی زیر قرار می‌دهیم.

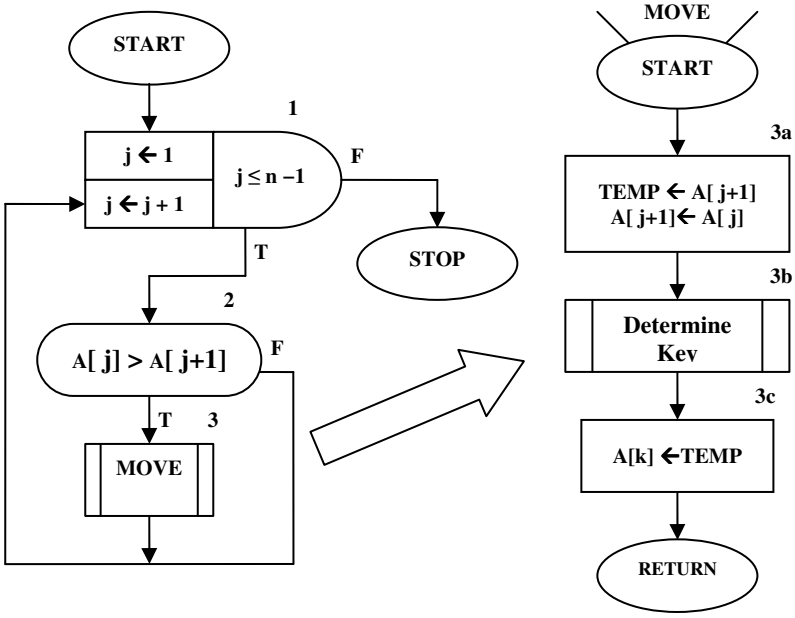


همچنین جهت مشخص شدن پایان زیر الگوریتم یک جعبهٔ بازگشت به‌صورت زیر در انتهای زیر الگوریتم قرار می دهیم:



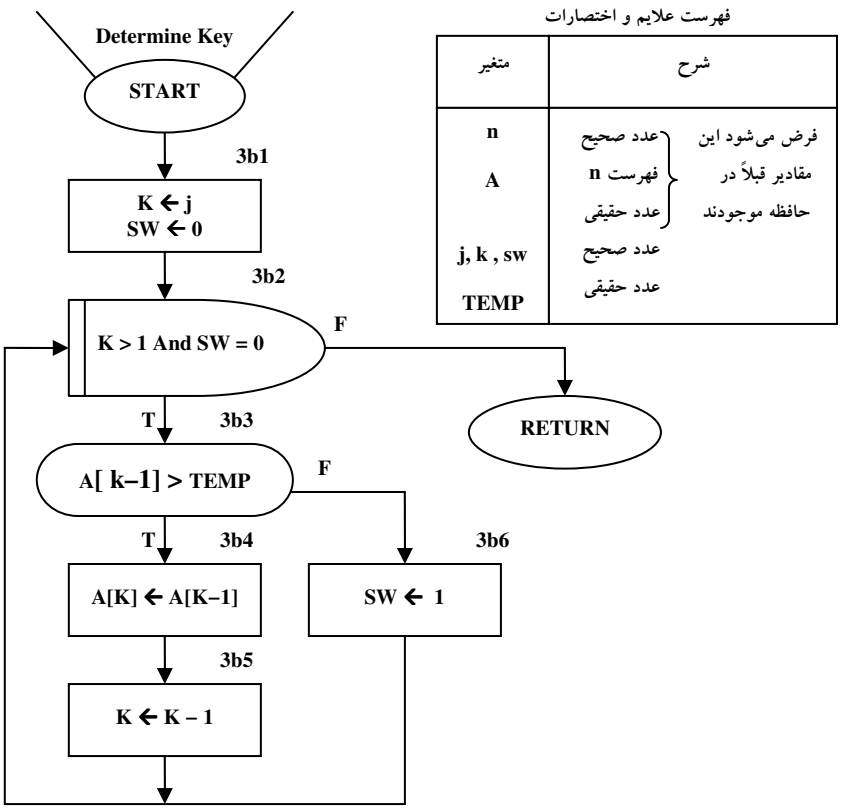
اما هنگامی که روند اجرای الگوریتم به جعبهٔ فراخوانی تابع می‌رسد. روند اجرای الگوریتم به نقطهٔ شروع رویه انتقال یافته و اجرای الگوریتم از ابتدای رویه دنبال می‌شود. هنگامی که روند اجرای الگوریتم به دکمهٔ بازگشت برسد، بدین مفهوم است که روند اجرای الگوریتم باید به جعبه‌ای از روندنمای اصلی (قبلی) که بلافاصله پس از نقطهٔ فراخوانی قرارگرفته‌است برگردد. لذا باز می‌توانید چنین تصور کنید که به‌جای جعبهٔ فراخوانی تابع، الگوریتم موجود در کارنمای رویهٔ موردنظر جایگزین می‌شود.

به‌عنوان یک مثال مرتب‌سازی‌حبابی را که پیشتر در بخش۶-۸ با آن آشنا شده‌اید، را یک بار دیگر با استفاده از رویه‌ها حل می‌کنیم. چگونگی حل این مسئله با استفاده از رویه‌ها در شکل ۸-۱۲ آمده است.



(الف) الگوریتم مرتب کردن (روندنمای اصلی) (ب) رویه موسوم به **MOVE** (زیرالگوریتم)

شکل ۸-۱۲ : پیاده‌سازی مرتب سازی حبابی با رویه



شکل ۸-۱۲- ج : رویهٔ موسوم به Determine Key

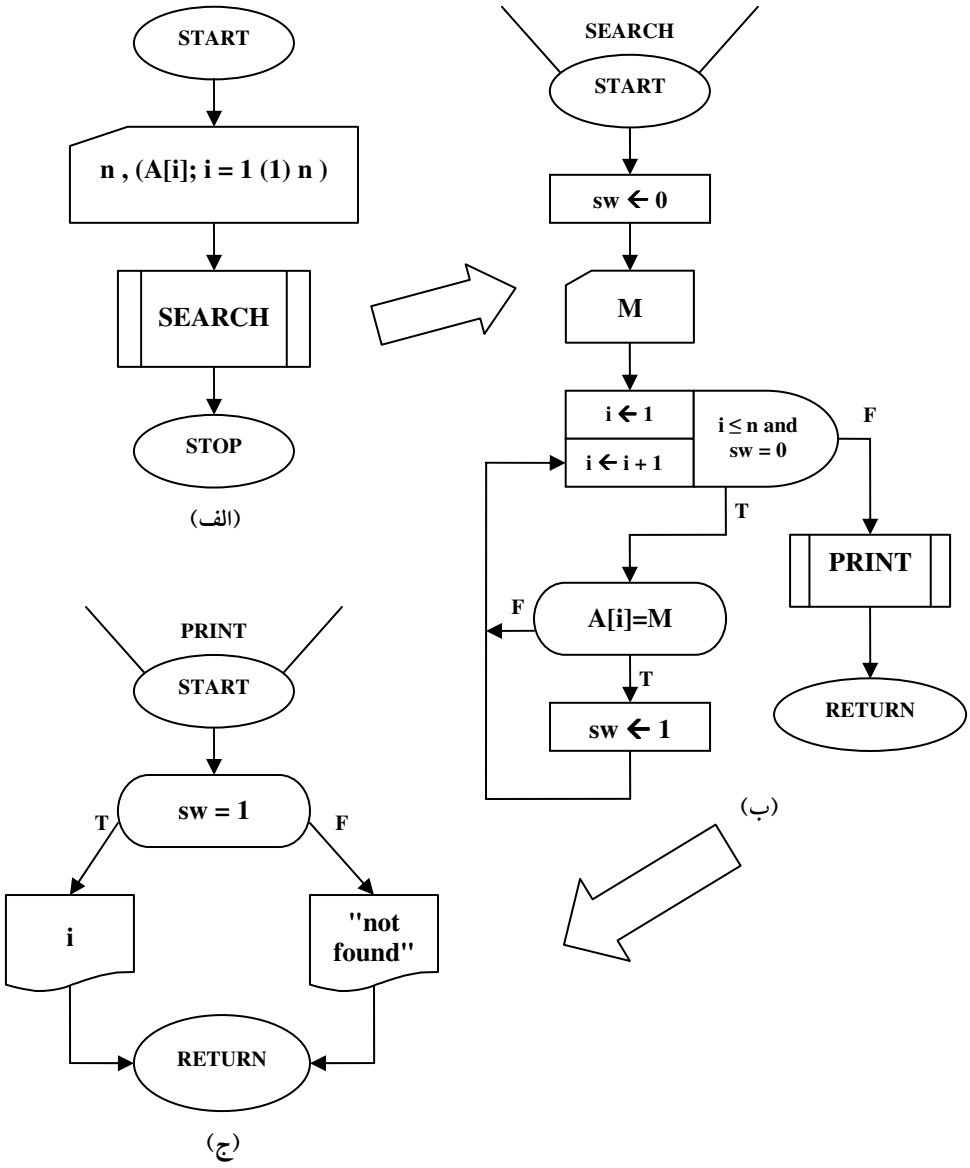
نوشتن زیر الگوریتم یا رویه برای حل مسائل، امتیازاتی دارد که بعضی از آنها را در زیر برمی‌شماریم:

۱. همکاری افراد را امکان‌پذیر می‌سازد به‌طوری که یک مسئله را می‌توان ابتدا به‌صورت اجمالی در یک کارنمای کلی حل کرد و سپس هر بخش را به عهدهٔ فرد یا افراد خاصی قرار دهند و در نهایت الگوریتم‌ها را در کنار یکدیگر قرار دهند.

۲. عیب‌یابی و اشکال‌زدایی الگوریتم‌ها ساده‌تر می‌شود. این مورد در زمینهٔ برنامه‌نویسی بسیار با اهمیت می‌شود. چرا که بسیاری از اشکالات زمان اجرا با وجود رویه‌ها به راحتی قابل مکان‌یابی هستند، چرا که وجود رویه‌ها ما را به مکان اشکال و خطای برنامه رهنمون می‌سازد.

۳. زیر الگوریتم‌هایی را که در یک الگوریتم نوشته شده‌اند به‌همان صورت و یا با اندکی تغییر می‌توان در الگوریتم‌های دیگر به‌کار برد. بنابراین از تکرار نوشتن دستورالعمل‌ها جلوگیری می‌شود.

مثال ۸-۱ : الگوریتمی که با دریافت یک لیست از ورودی، با استفاده از یک زیر الگوریتم یک عدد را در آرایه جستجو می‌کند، و در صورت یافتن عدد موقعیت عدد را در لیست چاپ کرده، و در غیر این صورت پیغام مناسبی را چاپ می‌کند.



شکل ۸-۱۳ : روندنماهای مثال ۸-۱

۸– ۶: تمرین

۱. الگوریتمی بنویسید که یک ماتریس m×n را خوانده سپس با فراخوانی یک زیرالگوریتم، کوچکترین عنصر هر سطر و ستون ماتریس را مشخص کند.

۲. زیرالگوریتمی بنویسید که لیستی به نام L و عددی به نام X را دریافت کند و سپس اگر مقدار X در لیست وجود داشته باشد آن را حذف کند.

۳. زیر الگوریتمی بنویسید که دو ماتریس را دریافت کرده و آنها را با هم مقایسه کند. اگر دو ماتریس با هم مساوی بودند مقدار ۱ و گرنه مقدار ۰ را در متغیر sw ذخیره کند و پس از بازگشت از زیر الگوریتم پیغام مناسب در الگوریتم اصلی چاپ گردد.

۴. الگوریتمی بنویسید که با استفاده از زیر الگوریتم ها یک لیست نامرتب را به داخل بیاورد و سپس به روش جستجوی دودویی یک مقدار دریافت شده از کاربر را جستجو کرده و محل آن را به الگوریتم اصلی بازگرداند تا پیغام مناسب درج شود.

۵. الگوریتمی بنویسید که توسط یک رویه اوّل بودن یک عدد را تشخیص دهد.

۶. زیرالگوریتمی بنویسید که در یک لیست، عناصر تکراری را حذف کند و هیچ محلی غیر از محل‌های آخر خالی نباشد.

۷. زیر الگوریتمی بنویسید که عناصر یک لیست را معکوس کند.

۸. زیر الگوریتمی بنویسید که دو متغیر را از ورودی دریافت کرده و محتویات آنها را با یکدیگر تعویض کند.

۹. زیرالگوریتمی بنویسید که سطرهای یک ماتریس را یکی در میان به ترتیب صعودی و نزولی مرتب کند.

۱۰.الگوریتمی بنویسید که با استفاده از رویه ضرب دو ماتریس را انجام دهد.

۱۱.الگوریتمی بنویسید که عدد صحیح N را خوانده، سپس توسط زیر الگوریتمی تعداد زیر مجموعه‌های ۴ عضوی یک مجموعهٔ N عضوی را پیدا کرده، در متغیر K قرار دهد.

۱۲.الگوریتمی بنویسید که عدد N را بخواند، سپس توسط یک زیر الگوریتم تعداد زیرمجموعه‌های ۱ عضوی یک مجموعهٔ N عضوی را در L[1] ، تعداد زیرمجموعه‌های ۲ عضوی یک مجموعهٔ N عضوی را در L[2] ، ... و در نهایت تعداد زیرمجموعه‌های N عضوی یک مجموعهٔ N عضوی را در L[N] قرار دهد، و در نهایت مقادیر ذخیره شده در لیست L را در الگوریتم اصلی در خروجی چاپ کند.

۱۳.الگوریتمی بنویسید که عدد صحیح N را خوانده، سپس زیرمجموعه های چهار عنصری مجموعهٔ {1,2,...,N} را تولید کرده و در یک ماتریس K×4 ذخیره نماید، که K نیز تعداد زیرمجموعه‌های ۴ عضوی مجموعه‌ای با N عضو است.

(راهنمایی: در هر ردیف باید عناصر یک زیرمجموعه را ذخیره کرد.)

۸– ۷ : مقدمه ای بر رمزنگاری

منظور از رمزگذاری، تبدیل اطلاعات به کدهایی است که آن را به‌صورت سری در می‌آورد. در رمزگذاری کاراکترهایی که اطلاعات را تشکیل می‌دهند به رشتهٔ جدیدی تبدیل می‌شوند. در صورت لزوم می‌توان آن را از حالت رمز خارج کرد و متن ساده (متن بدون رمز) را به‌دست آورد.

رمزگذاری از زمان ژولیوس سزار برای ارسال پیام های نظامی و سیاسی مورد استفاده قرار می گرفت. امروزه کاربردهایی مثل ارسال پست الکترونیکی، انتقال سرمایه به‌صورت الکترونیکی، بانک های اطلاعاتی و غیره وجود دارند که امنیت داده ها اهمیت زیادی دارد. همچنین رمزگذاری نامه‌های سیاسی و پیام های محرمانهٔ نظامی از حساسیت و اهمیت ویژه‌ای برخوردار است. از طرفی تلاش دولت‌ها برابر کشف رمز پیام‌های رد و بدل شده بین گروه‌های مخالف و دیگر کشورها رو به فزونی است، به‌طوری که بسیاری از دول قدرتمند کامپیوترهای بسیار قول پیکری را به این امر اختصاص داده‌اند که حتی کوچک‌ترین پیام مخابره شده در فضای گیتی را دریافت و رمز گشایی می کنند. لذا تلاش برای رمزنگاری (شامل رمزگذاری و رمزگشایی) از گذشته تا به امروز امری متداول بوده است. به همین جهت در این بخش به بیان برخی روش‌های رمزنگاری داده‌ها می‌پردازیم.

رمزگذاری به روش جانشینی

ساده‌ترین الگوی رمزگذاری، براساس عملیات رشته‌ای جانشینی انجام می‌شود، که در آن، رشته ساده پیمایش می‌شود و براساس یک قاعدهٔ معین، به‌جای هر کاراکتر، کاراکتر دیگری قرار می‌گیرد. به‌عنوان مثال در الگوی رمز سزار، هر کاراکتر، توسط کاراکتر دیگری که در فاصلهٔ **k** از آن قراردارد جایگزین می‌شود (حروف **A** تا **Z** به‌صورت یک دایره منظور می شوند). در رمز اولیه سزار، **k** مساوی ۳ بود، به‌طوری که هر کاراکتر **A** در متن ساده، با **D** جایگزین می‌شد، هر **B** با **E**، ... و هر **Y** با **B** ، و هر **Z** با **C** جایگزین می‌شد. به‌عنوان مثال، با استفاده از مجموعه کاراکترهای **A** تا **Z** می‌توانیم رشتهٔ "**IDESOFMARCH**" را به‌صورت زیر رمزگذاری کنیم:

متن ساده	**I**	**D**	**E**	**S**	**O**	**F**	**M**	**A**	**R**	**C**	**H**
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
متن رمزی	**L**	**G**	**H**	**V**	**R**	**I**	**P**	**D**	**U**	**F**	**K**

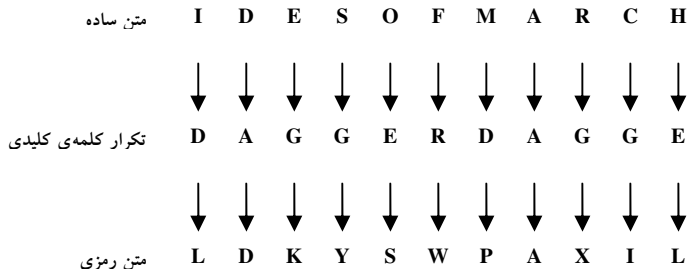
برای رمزگشایی این متن رمزی، گیرندهٔ پیام باید با دانستن **K**، هر کاراکتر را با **K** کاراکتر قبل از خودش جایگزین کند.

در شکل بهتری از روش جانشینی، از یک کلمه کلیدی برای مشخص کردن تفاوت مکان حروف، به‌جای استفاده از فاصله **K** در رمزگذاری سزار، استفاده می‌شود. در این روش رمزگذاری، یک کلمه کلیدی، کاراکتر به کاراکتر به رشته متن ساده اضافه می‌شود، به‌طوری که هر کاراکتر توسط موقعیتش در مجموعه کاراکترها و جمع به پیمانه ۲۶ نشان داده می‌شود. فرض کنید، مجموعه کاراکتر و موقعیت کاراکترها به‌صورت زیر باشد:

موقعیت	0	1	2	3	4	5	6	7	8	9	10	11	12
کاراکتر	A	B	C	D	E	F	G	H	I	J	K	L	M

13	14	15	16	17	18	19	20	21	22	23	24	25
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

حال کلمه کلیدی **DAGGER** را برای رمزگذاری در نظر بگیرید. می‌خواهیم متن ساده **IDESOFMARCH** را به متن رمزی تبدیل کنیم:

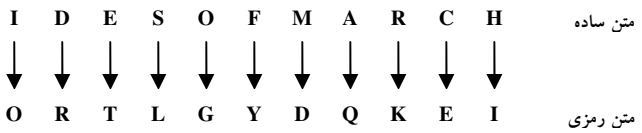


به‌عنوان مثال، برای تبدیل حرف**O** به حرف رمزی**S** ، موقعیت**O** یعنی ۱۴ با موقعیت**E** ، یعنی ۴ جمع می‌شود و کاراکتر**S** (موقعیت ۱۸=۱۴+۴) را به‌وجود می‌آورد. برای رمزگشایی این پیام، گیرنده پیام باید کلید رمز را بداند و عمل معکوس رمزگذاری را انجام دهد. یک روش جانشینی دیگر، جدول جانشینی است، که پیشتر در تمرین۳۴ بخش ۶-۹ الگوریتمی را برای رمزگذاری داده‌ها بدین طریق مشاهده کردید. در آن الگوریتم دو الفبا یکی شامل الفبای استاندارد و دیگری شامل الفبای رمز (کلید) داشتیم، وروند الگوریتم به این صورت بود که کاراکترها (نویسه‌های) متن دریافتی را به کاراکتر(نویسهٔ) متناظر در الفبای رمز تبدیل می‌کرد. بنابراین در جدول جانشینی مشخص می‌کنیم که هر کاراکتر باید به چه کاراکتری تبدیل شود. به‌عنوان مثال جدول زیر را در نظر بگیرید:

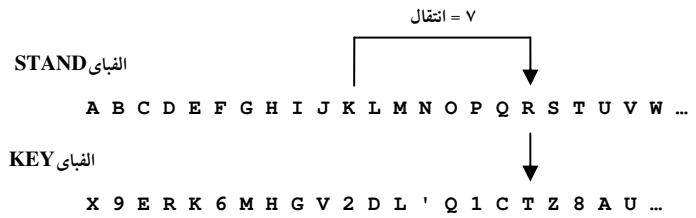
کاراکتر اصلی	A	B	C	D	E	F	G	H	I	J	K	L	M
کاراکترجانشین	Q	W	E	R	T	Y	U	I	O	P	A	S	D

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
F	G	H	J	K	L	Z	X	C	V	B	N	M

با توجه به این جدول رمز، متن **IDESOFMARCH** به‌صورت زیر رمزگذاری می‌شود:



برای رمزگشایی این پیام، گیرندهٔ پیام باید کلیدرمز، یعنی جدول جانشینی را بداند. چون !۲۶ (تقریباً $۱۰^{۲۸}$) جدول جانشینی می‌تواند وجود داشته باشد، بنابراین این روش، از حالت رمزگذاری ساده سزار، پیچیده‌تر است. اما با این وجود رمزگذاران ماهر به راحتی می‌توانند کلید اینگونه رمزگذاری را کشف کنند. چرا که اگر قرارباشد هر نویسه همواره به یک نویسهٔ ثابت از الفبای رمز ترجمه شود، کشف رمز بسیار آسان است. بنابراین باید به دنبال روش‌های بهتر و مشکل‌تری برای رمزگذاری متون باشیم تا متقابلاً رمزگشایی متون رمزی نیز برای دیگران ناممکن و یا حداقل بسیار مشکل باشد. در الگوریتم‌های رمزگذاری خوب از اعمالی چون شیفت دادن (انتقال)، جابه‌جایی (جایگشت) نویسه‌ها و امثال آن استفاده می‌شود تا الگوریتم رمزی کردن به مراتب سخت‌تر گردد. روش‌های رمزگذاری که در اینجا معرفی خواهیم کرد منجر به متن رمزی می‌شود که کشف آن به مراتب مشکل‌تر خواهد بود. لذا ابتدا به بررسی یک روش ساده در رمزگذاری می‌پردازیم، و سپس به بیان روش‌های مشکل‌تر خواهیم پرداخت. مبنای این روش یک عمل «انتقال» است که مفهوم آن را از شکل ۸-۱۴ می‌توان دریافت.

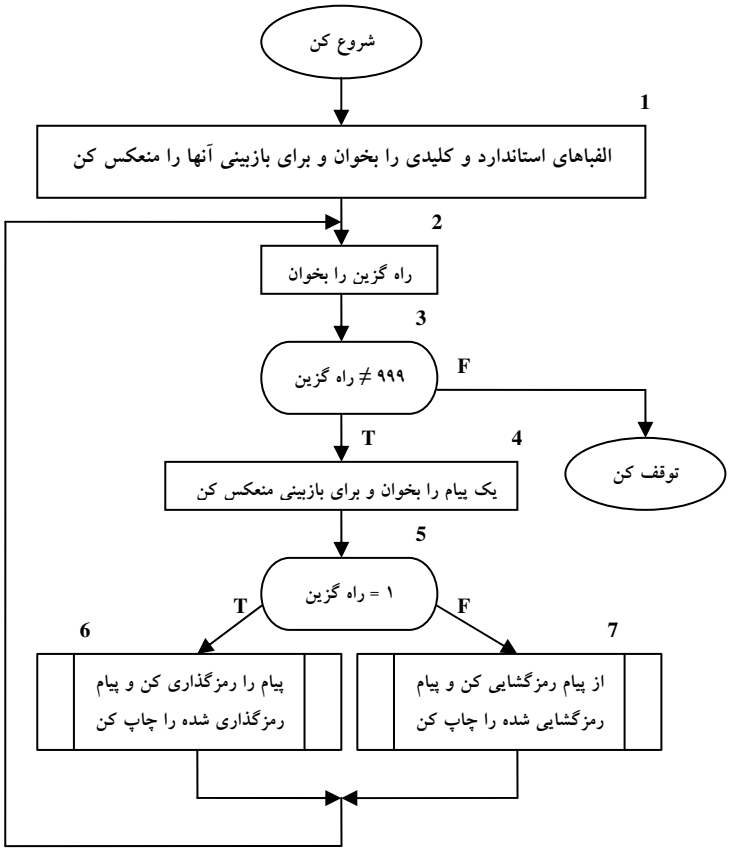


شکل ۸-۱۴ : عمل انتقال در رمزنگاری

به‌طوری که در این شکل مشاهده می‌کنید، هنگامی که مقدار انتقالی ۷ در مورد حرف **K** در الفبای استاندارد اعمال شود، این حرف به حرف **T** در الفبای کلید رمزگذاری می‌شود. ما در روش خود مقدار انتقال را به محل نویسه نه در هیچ یک از دو الفبا، بلکه در پیامی که قرار است رمز گذاری شود، وابسته می‌کنیم. بنابراین روش ما نسبت به روش سزار یک مزیت بسیار مهم دارد و آن این که مقدار انتقال همواره ثابت نیست.

الگوریتمی که می‌خواهیم بسازیم ابتدا الفباهای استاندارد و کلید را به داخل می‌آورد. سپس سری نامحدودی از پیام‌ها را به داخل می‌آورد و مورد پردازش قرار می‌دهد یعنی بر مبنای مقدار یک راه‌گزین یا علامت که قبل از هر پیام واقع می‌شود، پیام خوانده شده را رمزگذاری و یا رمزگشایی می‌کند. ساخت این الگوریتم در شکل ۸-۱۵ دیده می‌شود. الفبای استاندارد (**STAND**) تشکیل شده است از ۲۶ حرف القبای انگلیسی به ترتیب معمولی و به دنبال آن ارقام ۱ تا ۹ و ۰ و به دنبال آن آپوستروف (')، ویرگول (,) و نشان ممیز(/). الفبای کلیدی (**KEY**) جایگشت منتخبی از این ۳۹ نویسه است.

اکنون طریقهٔ رمزگذاری یک نویسه از پیام را شرح می‌دهیم. در الفبای استاندارد، نویسهٔ موردنظر را جستجو می‌کنیم تا مقدار زیر نویس محل آن را تعیین کنیم. سپس مقدار صحیح انتقـال را به این زیرنویس اضافه می‌کنیم. مجموعی که به‌دست می‌آید مقدار زیرنویس نویسهٔ رمزگذاری شده را در الفبای کلیدی تعیین می‌کند.



شکل ۸-۱۵ :روندنمای کلی در رمزنگاری

در مواردی که عمل انتقال منجر به زیرنویسی خارج از محدودهٔ (۳۹-۱) شود، می‌توان از حساب پیمانه‌ای[1] برای برگرداندن زیرنویس به داخل محدوده استفاده کرد، در این حالت حساب پیمانه‌ای را می‌توان به طریق زیر تعبیر کرد.

می‌توان چنین تصور کرد که هر الفبا از دو طرف به دفعات زیاد تکرار شده است. پس از محاسبهٔ زیرنویس نویسهٔ موردنظر در الفبای رمزی شده، به‌راحتی می‌توان با کم کردن یا اضافه کردن ۳۹، یک الفبای کامل را «به عقب یا جلو لغزانید» و به این ترتیب زیرنویسی در محدودهٔ ۱ تا ۳۹ به‌دست آورد. این مقدار، نویسهٔ مطلوب در الفبای مقصد را مشخص خواهد کرد. مطالب بیشتر در مورد چگونگی تعیین مقدار صحیح انتقال، بعداً مورد بحث قرار خواهد گرفت.

1. modular

هنگام به خارج دادن پیام‌هایی که رمزگذاری شده‌اند، باید به دنبال هر پنج نویسه یک جای خالی قرار داد تا خط چاپی به گروه‌های شبیه کلمه تقسیم شود. در پیام‌هایی که باید رمزگذاری شوند باید کلمات را با علامت (/) از یکدیگر جدا کرد، زیرا علامت جای خالی (□) جز هیچ یک از دو الفبا نیست. شکل ۸-۱۶ نمونه‌ای از خروجی موردنظر از برنامهٔ رمزی کردن را نشان می‌دهد. ابتدا نسخه‌هایی از هر دو الفبا چاپ می‌شود. سپس به دنبال هر پیام ورودی نسخه‌ای از فرم رمزی شدهٔ آن، که بسته به مورد، ممکن است رمزگذاری یا رمزگشایی شده باشد چاپ می‌شود.

THE STANDARD ALPHABET IS (الفبای استاندارد **)**

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 1 2 3 4 5 6 7 8 9 0 ' , /

THE KEY ALPHABET IS (الفبای کلیدی **)**

X 9 E R K 6 M H G V 2 D L ' Q 1 C T Z 8 A U , J I Y 7 N F 0 O S / B 4 W 3 P 5

THE MESSAGE IS (پیام **)**

MR/COLLINS/WAS/NOT/A/SENSIBLE/MAN/AND/THE/DEFFICIENCY/OF/NATURE/HAD/ BEEN/BUT/LITTLE/ASSISTED/BY/EDUCATION/OR/SOCIETY//PRIDE/AND/PREJUDICE/ BY/JANE/AUSTEN/

ENCODE , IT READS (پیام رمز شده **)**

TNQ,X EHV80 1K72W 1USC, 7DEC7 U84// DK,'8 P/B8L QQJ0W 9VG8A 4FCWC LDO,H '1HHA Z7N4R G,'A6 VBD'0 4SD0W 8I849 O4WRS DUYG4 P,CNS ,RDP, JCNFV 85KY/ FF449 8Q13K WM,2A 877B, ET'SN NB8,F Z/J

THE MESSAGE IS (پیام رمزی **)**

TNQ,X EHV80 1K72W 1USC, 7DEC7 U84// DK,'8 P/B8L QQJ0W 9VG8A 4FCWC LDO,H '1HHA Z7N4R G,'A6 VBD'0 4SD0W 8I849 O4WRS DUYG4 P,CNS ,RDP, JCNFV 85KY/ FF449 8Q13K WM,2A 877B, ET'SN NB8,F Z/J

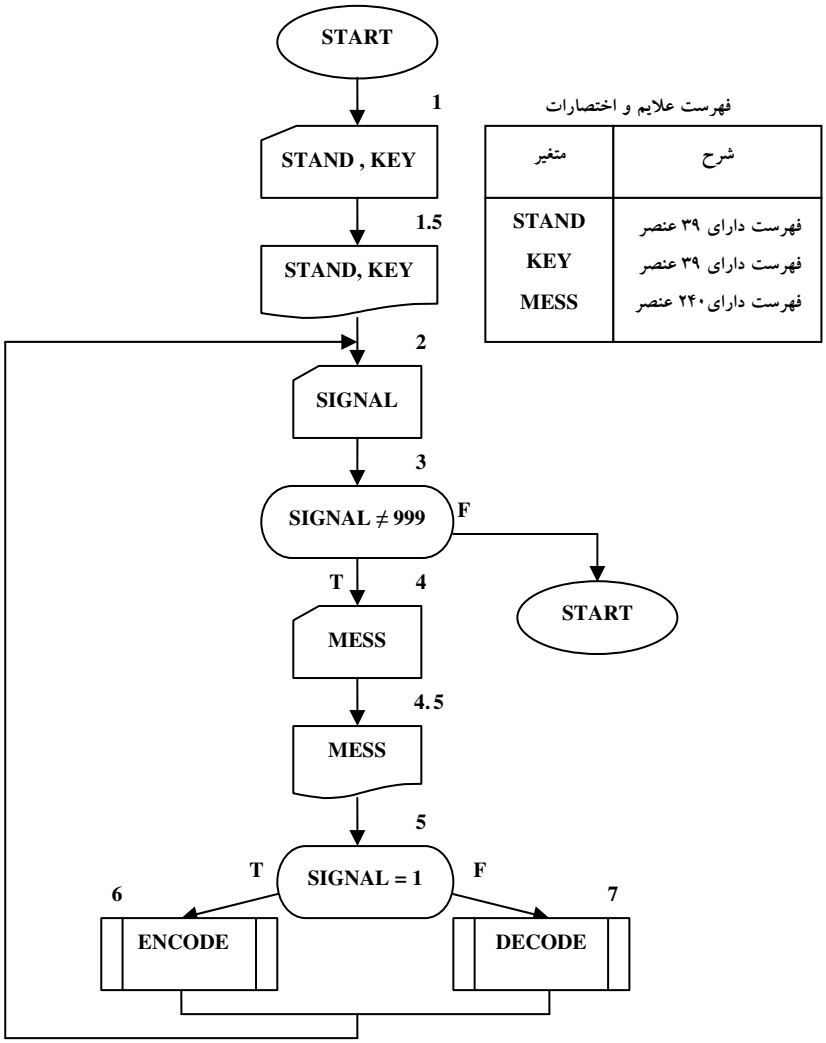
DECODED , IT READS(پیام رمز گشایی شده**)**

MR/COLLINS/WAS/NOT/A/SENSIBLE/MAN/AND/THE/DEFFICIENCY/OF/NATURE/HAD/ BEEN/BUT/LITTLE/ASSISTED/BY/EDUCATION/OR/SOCIETY//PRIDE/AND/PREJUDICE/ BY/JANE/AUSTEN/

شکل ۸-۱۶ :نمونهٔ خروجی از برنامهٔ رمزی کردن

شکل ۸-۱۷ بخشی از جزئیات روندنمای کلی شکل ۸-۱۵ را نشان می‌دهد و به‌طوری که در فهرست علائم و اختصارات مشاهده می‌شود، ساخت‌های داده‌های لیستی و ابعاد آنها در مورد متغیرهای مهم قید شده است. چنانکه ملاحظه می‌شود تصمیمات جزئی در مورد عمل رمزگذاری و رمزگشایی به بعد موکول شده است.

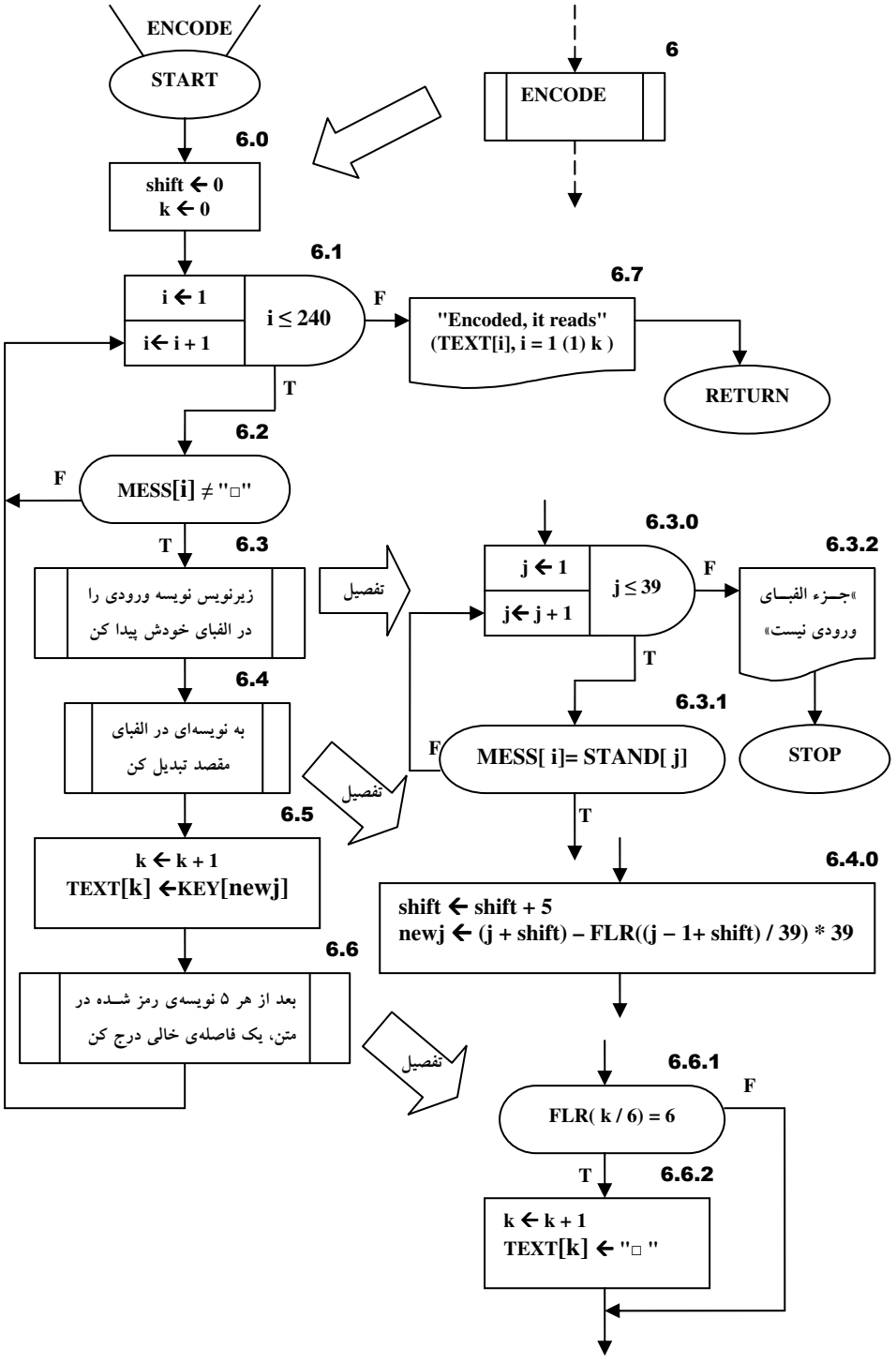
یکی از مشخصات روش تجزیهٔ گام به گام در حل مسئله که تقریباً در کلیهٔ موارد به‌کار گرفته می‌شود این است که می‌توان توسعه از یک سطح جزئیات به سطح دیگر را تکه تکه انجام داد و این کار تقریباً همواره انجام می‌شود. بدین معنی که هر قدم از الگوریتم را می‌توان مستقل از دیگر قدم‌ها برای بسط بیشتر جزئیات انتخاب کرد.



شکل ۸-۱۷ : تفضیل روندنمای اصلی

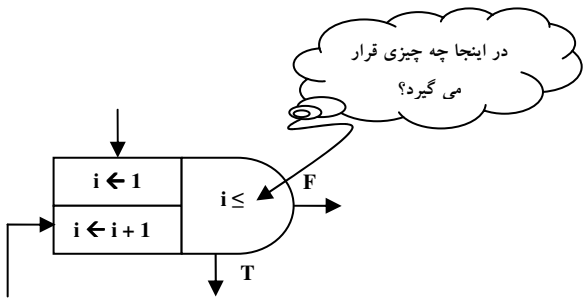
رویهٔ رمزگذاری

اکنون آماده‌ایم که در مورد رویهٔ رمزگذاری فکرکنیم. این قسمت از روندنما باید پیام را نویسه به نویسه بپوید، هر نویسه را در الفبای استاندارد مشخص کند، انتقال و ترجمه به نویسهٔ مربوطه در الفبای کلیدی را انجام دهد و پیام رمزی شده‌ای را برای چاپ آماده کند. شکل ۸-۱۸ الگوریتم رمزگذاری را نشان می‌دهد که با استفاده از بحث زیر گسترش یافته است.



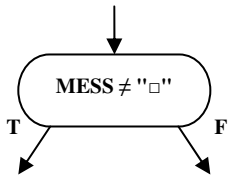
شکل ۸-۱۸ : رویهٔ رمزگذاری

«پویش نویسه به نویسه پیام» با فرض دانستن طول پیام، حلقه‌ای را پیشنهاد می‌کند که به‌وسیلهٔ جعبهٔ تکرار کنترل شده باشد.



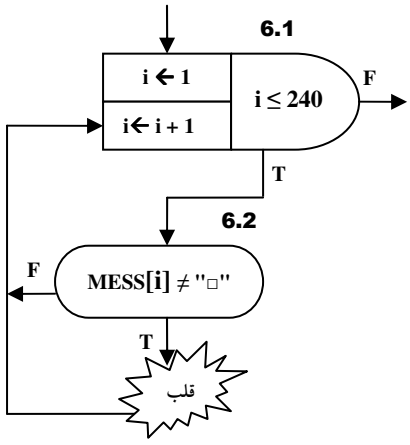
شکل ۸- ۱۹

اکنون دو راه در مقابل ما وجود دارد. می‌توانیم N یعنی تعداد نویسه‌های پیام را شمارش کنیم و این مقدار را به داخل بیاوریم و از آن به‌عنوان حد بالایی شرط کنترل کنندهٔ حلقه استفاده کنیم، و یا می‌توانیم بعد لیست پیام، یعنی ۲۴۰ را به‌عنوان حد بالایی به کار بریم، و بنابراین نیازی به شمردن نویسه‌های پیام نداشته باشیم. ولی در صورت دوم باید شرط:



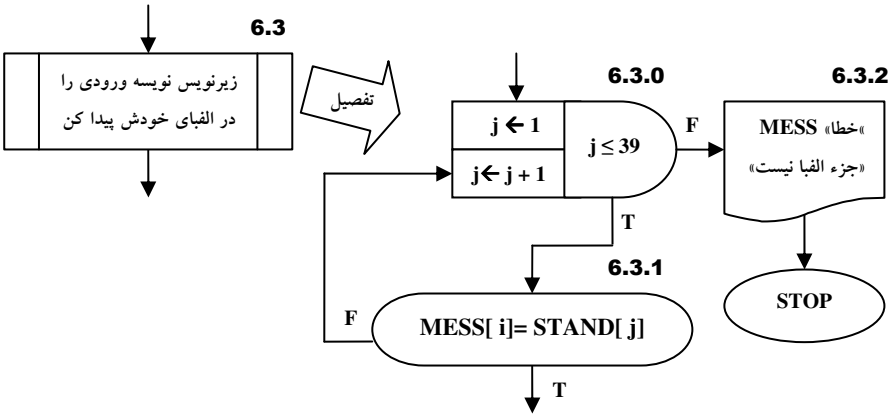
شکل ۸- ۲۰

را نیز اضافه کنیم تا هر نویسهٔ خالی که در هر جای پیام یافت شود نادیده گرفته شود (جعبه‌های 6.1 و 6.2 در شکل ۸- ۱۸).



شکل ۸- ۲۱

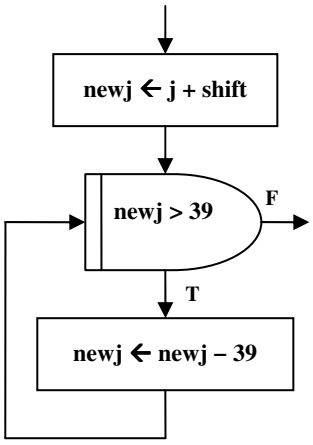
پس از یافتن نویسه‌ای غیر از نویسهٔ خالی در پیام، باید **j** یعنی محل نویسهٔ مطابق با آن را در الفبای استاندارد تعیین کنیم. طرز انجام این کار در شکل ۸-۲۲ نشان داده شده است.



شکل ۸- ۲۲

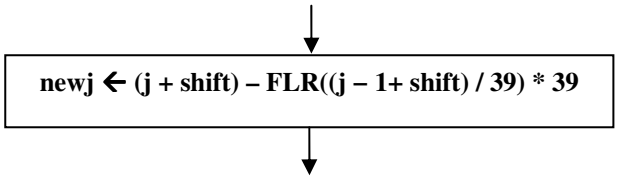
هنگامی که زیرنویس نویسهٔ مفروض در الفبای خودش تعیین شد، آماده‌ایم که آن را به نویسهٔ نظیرش در الفبای مقصد تبدیل کنیم.

محل نویسه در الفبای مقصد، **newj**، به طریق زیر محاسبه می‌شود. ابتدا مقدار صحیح انتقال را به‌دست می‌آوریم و سپس آن را به **j** یعنی محل نویسه در الفبای استاندارد اضافه می‌کنیم. برای اینکه مطمئن شویم **newj** بین ۱ و ۳۹ واقع شده است از مقدار (انتقال + **j**) مکرراً ۳۹ را کم می‌کنیم تا هنگامی که در محدودهٔ صحیح قرار گیرد. یک راه انجام این کار به این صورت است:



شکل ۸-۲۳

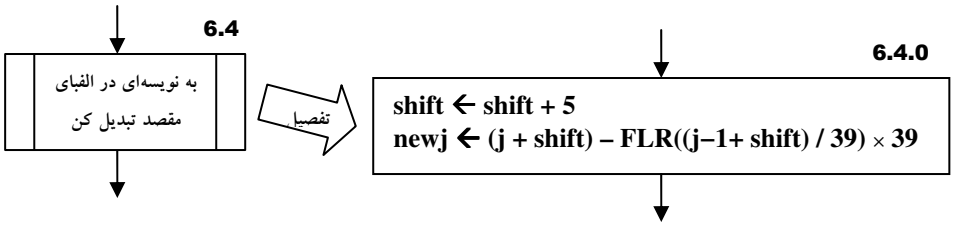
با به‌خاطر آوردن تمرین‌های مربوط به چرخ بازی در بخش ۲-۱۱، باید به راحتی بتوانید خود را قانع کنید که از طریق جعبهٔ شکل ۸-۲۴ نیز مقداری مشابه جعبهٔ شکل ۸-۲۳ برای **newj** به‌دست می‌آید.



شکل ۸- ۲۴

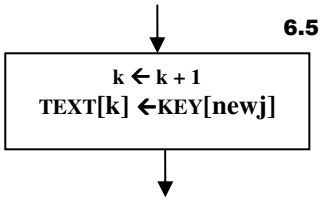
حتماً توجه کرده‌اید که هنوز در مورد نحوهٔ تعیین مقدار خود انتقال تصمیمی نگرفته‌ایم. این تصمیم را تا به حال به تعویق انداخته‌ایم ولی اکنون به این انتظار پایان می‌دهیم.

بنا را برآن می‌گذاریم که مقدار انتقال برای اولین نویسهٔ پیام ۵ محل به طرف راست و برای نویسه‌های بعدی هر کدام ۵ محل بیشتر از مقدار انتقال نویسهٔ قبلی باشد یعنی مقدار انتقال برای نویسهٔ دوم برابر ۱۰ محل خواهد بود و به همین ترتیب تا آخر. اگر در ابتدای رویه به **shift** (انتقال) مقدار اولیهٔ صفر منسوب کنیم، آنگاه قدم‌های لازم برای محاسبهٔ **newj** را به طریق زیر بهتر می‌توان نشان داد:



شکل ۸- ۲۵

بالاخره، پس از محاسبهٔ **newj** می‌توانیم از **KEY[newj]** نسخه‌برداری و آن را به فهرست خروجی که **TEXT** نامیده می‌شود اضافه کنیم.

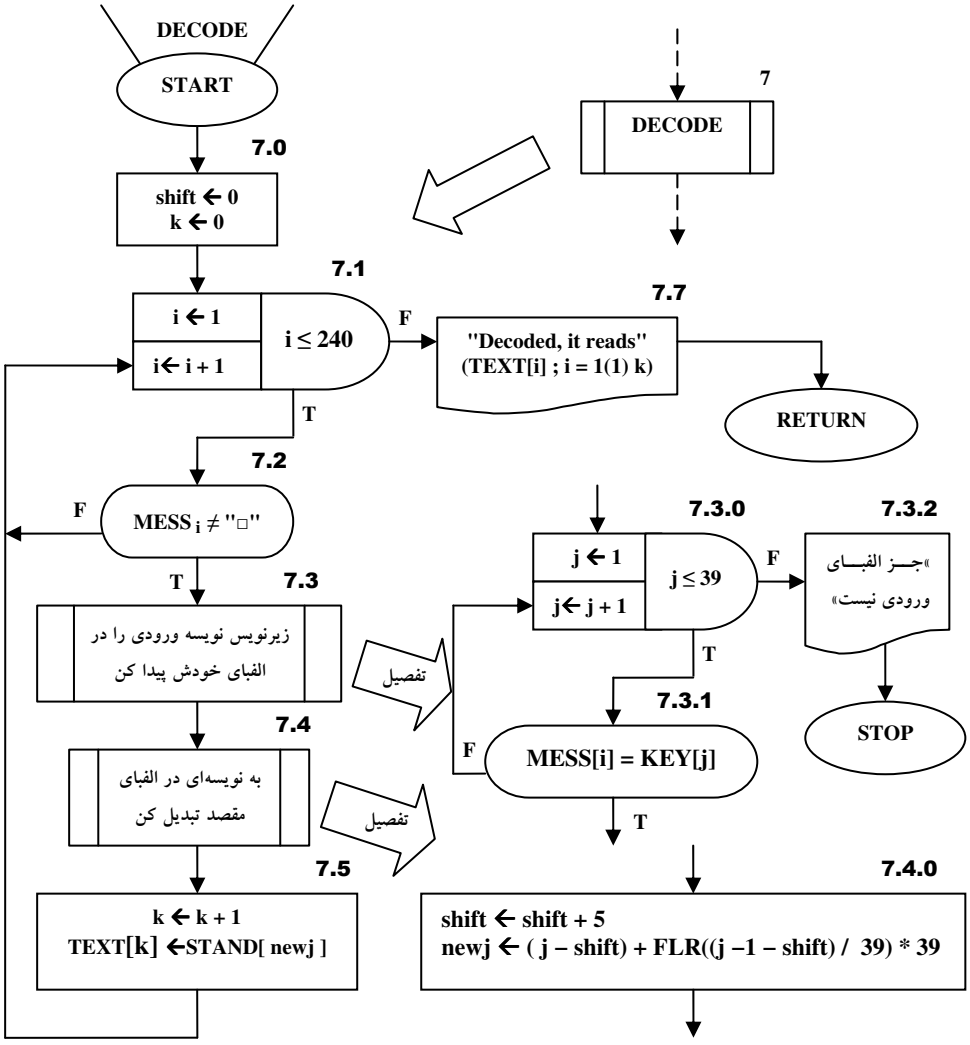


شکل ۸- ۲۶

هنگامی که الگوریتم آماده است تا پیام رمز شده را چاپ کند یک اشاره‌گر زیرنویس، **K** ، باید به آخرین نویسه‌ای که به فهرست خروجی منسوب شده است اشاره کند. بنابراین، مقتضی است که در شروع عمل رمزگذاری به **K** مقدار اولیهٔ صفر را نسبت دهیم و درست قبل از اینکه نویسهٔ جدیدی به **text** منسوب شود یکی به **K** اضافه کنیم (جعبهٔ **6.5**). قسمت محاسبهٔ حلقه‌ای که هر کدام از نویسه‌ها را پردازش می‌کند، با اضافه‌کردن یک جای خالی بعد از هر پنج نویسه (جعبهٔ **6.6**) تکمیل می‌شود. شکل ۸-۱۸ طرح **ENCODE** را که هم اکنون تکمیل کردیم به‌طور خلاصه نشان می‌داد. جعبهٔ **6.7** که قبلاً ذکری از آن به میان نیامده است، پیام رمزگذاری شده را چاپ می‌کند.

رویهٔ رمز گشایی

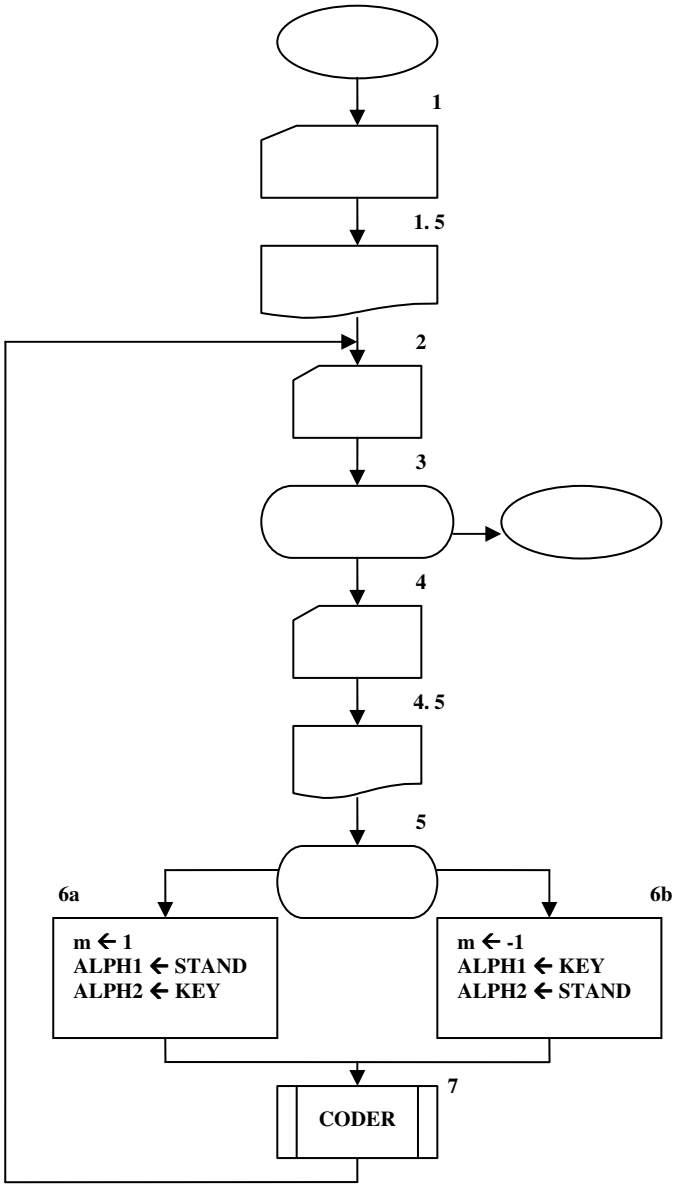
رویهٔ **DECODE** که در شکل ۸-۲۷ نشان داده شده است عمل رمزگشایی را که مشابه و تقریباً قرینهٔ عمل رمزگذاری است انجام می‌دهد. محل نویسه‌های پیام ورودی (به استثنای جای خالی) باید در الفبای کلیدی پیدا شود. آنگاه به زیرنویس مربوطه انتقال معکوس داده می‌شود، یعنی مقدار انتقال از آن کم می‌شود. سپس نویسهٔ نظیر آن از الفبای استاندارد به فهرست خروجی اضافه می‌شود. فقط برخی جعبه‌ها در الگوریتم **DECODE** با جعبه‌های نظیر خود در شکل ۸-۱۸ تفاوت دارند. توجه داشته باشید که هنگام رمزگشایی نیازی به اضافه کردن جاهای خالی برای قسمت کردن فهرست خروجی نیست زیرا خطوط مایل (/) کلمات را از یکدیگر جدا می‌کنند.



شکل ۸- ۲۷ : رویهٔ رمزگشایی

کار در کلاس ۸-۳ :

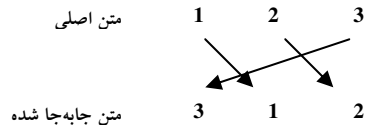
سعی کنید زیر الگوریتمی به نام **CODER** *بسازید که از دوباره کاری ها، جلوگیری کند. یعنی با دریافت یک متن برحسب مورد رمزگذاری یا رمزگشایی را انجام دهد. احتمالاً کارنمایی که در شکل ۸-۲۸ آمده است می تواند شما را در ساخت این زیر الگوریتم یاری دهد.*



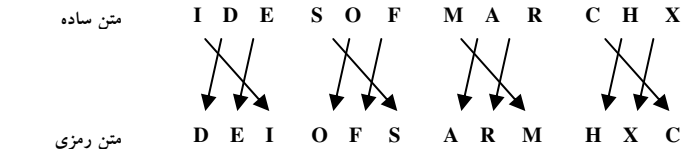
شکل ۸-۲۸ : صورت اصلاح شده روندنمای اصلی که از CODER به‌عنوان ENCODE یا DECODE استفاده می‌کند به‌صورت فوق است. پیش انتساب‌های جعبهٔ 6a ، برنامه را برای رمزگذاری آماده می‌کند، و پیش انتساب‌های جعبهٔ 6b برنامه را برای رمزگشایی آماده می‌کند.

رمزگذاری به روش جایگشت

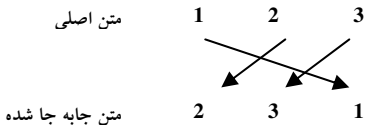
یک عمل رشته‌ای دیگر که در الگوهای رمزگذاری استفاده می‌شود، جایگشت است. در این روش، آرایش کاراکترهای موجود در متن ساده یا بلوکی از متن ساده، تغییر می‌کند. به‌عنوان مثال، ممکن است متن ساده را به بلوک‌هایی به اندازه ۳ کاراکتر تقسیم کرده، کاراکترهای هر بلوک را به‌صورت زیر جابه‌جا کنیم:



بدین ترتیب برای رمزگذاری **IDESOFMARCH** به‌صورت زیر عمل می‌شود (چون تعداد کاراکترها مضربی از سه نبود، حرف **X** به‌طور تصادفی انتخاب کرده، به متن اضافه می‌کنیم):



برای رمزگشایی، گیرنده پیام باید کلید جایگشت را بداند و عکس عمل رمزگذاری را انجام دهد:



۸- ۸: تمرین

۱. برنامه‌ای به زبان C++ برای الگوریتم رمزی‌کردن شکل ۸-۱۷ بنویسد[۱] و دو پیام زیر را رمزنگاری کنید. (اگر کامپیوتر ندارید سعی کنید الگوریتم را به طور دستی پیگیری کنید.)

پیام برای رمزگذاری:

TO/BE/SURE/ANY/COMPUTER/BEHAVIOR/CAN/ULTIMATELY/BE/DESCRIBED/IN/TERMS/ OF/THE/UNDERLYING/CIRCUITRY/BUT/THIS/IS/OFTEN/NO/MORE/HELPFUL/THAN/IT/ WOULD/BE/TO/DESCRIBE/MAN'S/BEHAVIOR/IN/TERMS/OF/THE/ATOMS/MAKING/UP/HIS/ BODY/

پیام برای رمزگشایی:

IIQU0 0QUJ1 1UXCW KUI/R H31EV TYJ2E 'UGC8 PDF1DT ZNA64 HY6LA A0W1K IAZ70 SMCH/ ,99LP ZQ'5S BBAZ' 8P31K QL2Z0 KBT27 0EUF' TENA7 ENETR J85FR B'8VN 4R44G 2,S7H RWUQL OUWL3 Y,B07 N4RRF FZNB' E,ZCI IRWQR 01,05 3J174 UWM3, LOOEN GKRFU JJ0QQ TFQ/Y 9KQL2 33Y56 AM'SY LS

۲. با انتخاب نویسه‌های خاصی که مایل هستید از آن استفاده کنید، الفبایی برای خود طرح کنید. نویسه‌ها را به نحوی جابه‌جا کنید تا الفبای کلید به دست آید. با استفاده از CODER و رمز خود، پیام‌هایی را رمز گذاری کنید. پردازش خود را با رمزگشایی از پیام‌های رمزگذاری شده بازبینی کنید، سپس از دوستان خود بخواهید که از پیام های رمزگذاری شدهٔ شما کشف رمز کنند.

توجه : چون پس از هر پنج نویسهٔ رمزی شده یک جای خالی اضافه می‌شود پیام رمزگذاری شده همواره از پیام اصلی طولانی‌تر است. این بدان معنی است که اگر پیام اصلی بیش از ۲۰۰ عنصر از MESS را پر کند، فهرست TEXT باید بیشتر ۲۴۰ عنصر داشته باشد.

۳. الگوی رمزگذاری جایگشتی محض، سرّی نیست. توضیح دهید چرا با توصیف چگونگی الگوی رمزگذاری که عمل جایگشت را در رشته n بیتی انجام می‌دهد، با مطالعه چگونگی رمزگذاری رشته‌هایی از بیت‌ها، قابل کشف است. برای $n = 4$ مسئله را توضیح دهید.

---

۱. در لوح‌فشرده همراه کتاب یک برنامه‌ی ویژوالی در پرونده‌ای با نام CODING قرار دارد که با اجرای آن می‌توانید نحوه‌ی رمزگذاری و رمزگشایی را به طور عملی ببینید.

۸-۹ : نمودار N-S

تا کنون با نحوهٔ نمایش گرافیکی الگوریتم‌ها یعنی استفاده از فلوچارت یا کارنما آشنا شدید. چنان که تجربه نمودید استفاده از نمایش گرافیکی بسیار آسان‌تر و بهتر از استفاده از جملات برای بیان یک الگوریتم است.

اما یک سؤال : آیا شما می‌توانید نمایش دیگری برای بیان الگوریتم ابداع کنید؟

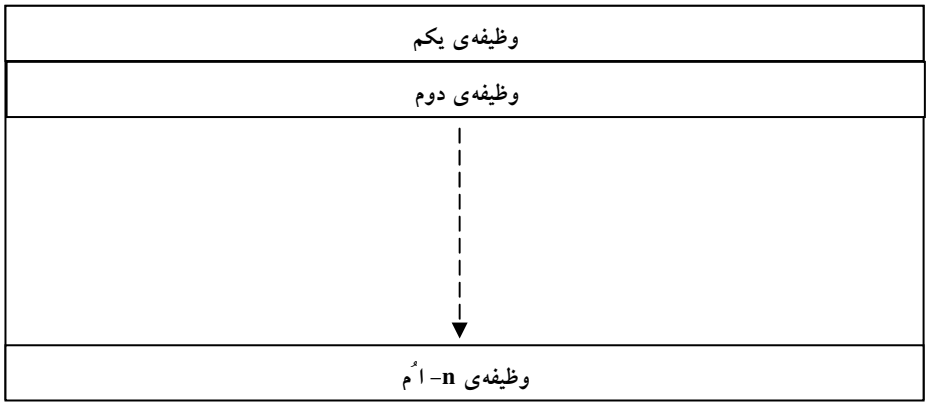
واقعیت آن است که کارنما یا همان فلوچارت یک قرار داد بیش نیست، و چه بسا بتوان با کمی تلاش و تغییر قراردادها شیوه‌ای نو و جدید را برای نشان دادن الگوریتم‌ها ابداع نمود. نمودار N-S که توسط ناسی و اشنایدرمن ابداع و توسط چاپین بسط داده شد شیوه‌ای دیگر و شاید بهتر از فلوچارت برای نمایش الگوریتم‌ها است. این نمودار مستطیلی شکل از ساختارهای ساخت‌یافته نشأت گرفته است و بدین دلیل پیچیدگی‌ها و مشکلاتی که پیشتر در خصوص کارنما مطرح شد را، ندارد. دیگر مزایای این روش چنین است:

۱. دامنه عملیاتی (یعنی دامنه تکرار یا شروط یک ساختار شرطی) به خوبی تعیین می‌شود.[۱]

۲. انتقال اختیاری کنترل غیر ممکن است زیرا نحوهٔ اجرای خط به خط دستورات از انتقال‌های نامنظم که موجب ایجاد حالت‌های غیرساخت‌یافته می‌شد جلوگیری می‌کند و تقریباً به زبان‌های برنامه‌نویسی ساخت یافته نزدیک‌تر از فلوچارت است.

۳. نمایش دادن بازگشتی - که در فصول نهم و دهم با آن آشنا خواهید شد- آسان است.

اما چگونه از این نمودار برای نمایش الگوریتم‌ها استفاده کنیم!؟ عنصر بنیادی این نمودار، مستطیل است و تمامی ساختارها به نوعی به واسطهٔ مستطیل نمایش داده می‌شوند، لذا در برخی کتب به نمودار N-S ، نمودار مستطیلی نیز گفته شده است. برای شروع کار باید یک مستطیل همچون شکل زیر در صفحهٔ کاغذ خود بکشید.



شکل ۸- ۲۹ : شکل کلی دنباله دستورات در نمودار N-S

۱. البته چنان که پیشتر در فصل چهارم مشاهده کردید ما در این کتاب یک ساختار تکرار منحصر به فرد و خارج از استانداردهای مشخص شده برای رسم فلوچارت را پیشنهاد کردیم که این ویژگی‌ها را به خوبی برآورده می‌کرد. اما ساختارهای تکرار موجود در نمودار N-S جزء استاندارد این نمودار می‌باشد و برای همه قابل درک است.

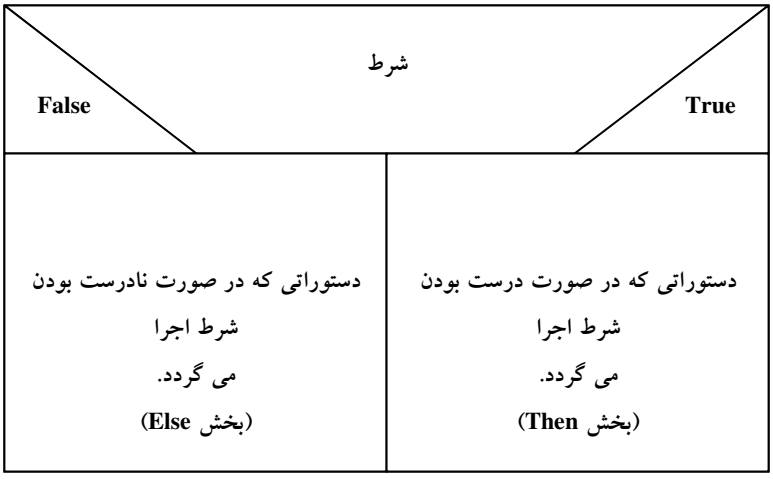
در نمودار **N-S** کل الگوریتم را یک مستطیل فرامی‌گیرد و هر دستور در یک خط مجزا نوشته می‌شود. خط اول شامل دستور شروع یا همان **Start** و خط آخر شامل دستور **Stop** یا **End** می‌باشد. برای هر دستور جدید کافی است یک خط کشیده و فضای لازم را برای نوشتن دستور ایجاد کنید و سپس دستور خود را در فضای ایجاد شده بنویسید. دستور نوشته شده می‌تواند به زبان فارسی، انگلیسی و یا شبه کد و یا همانند دستورات فلوچارت‌ها باشد.

مثال ۸-۲: در زیر نمودار **N-S** برای الگوریتمی که ضرایب یک معادله خطی **aX+b = c** را خوانده و مقدار **X** را محاسبه و چاپ می‌کند رسم شده است.

Start.
Read a, b, c.
X= (c – b) ÷ a. //suppose 'a' is not zero.
Print X.
End.

شکل ۸- ۳۰ : نمودار N-S مثال ۸-۲

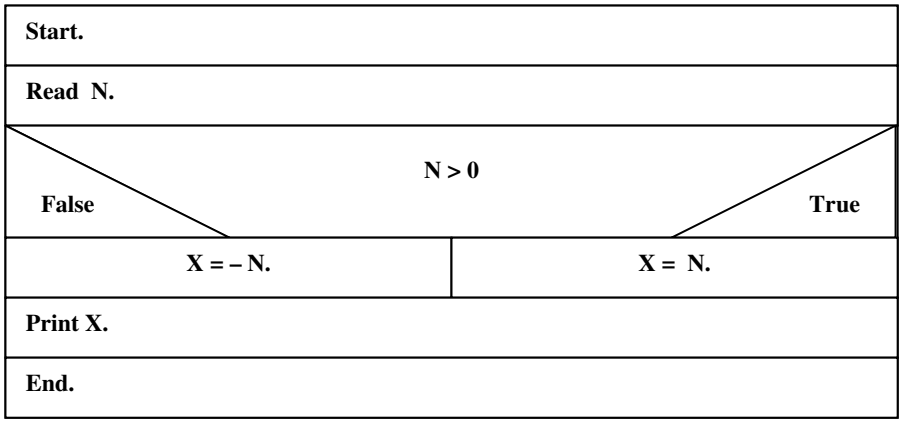
برای نشان دادن ساختار تصمیم گیری **If then else** از ساختار زیر استفاده می‌کنیم:



شکل ۸- ۳۱ : ساختار If then else در نمودار N-S

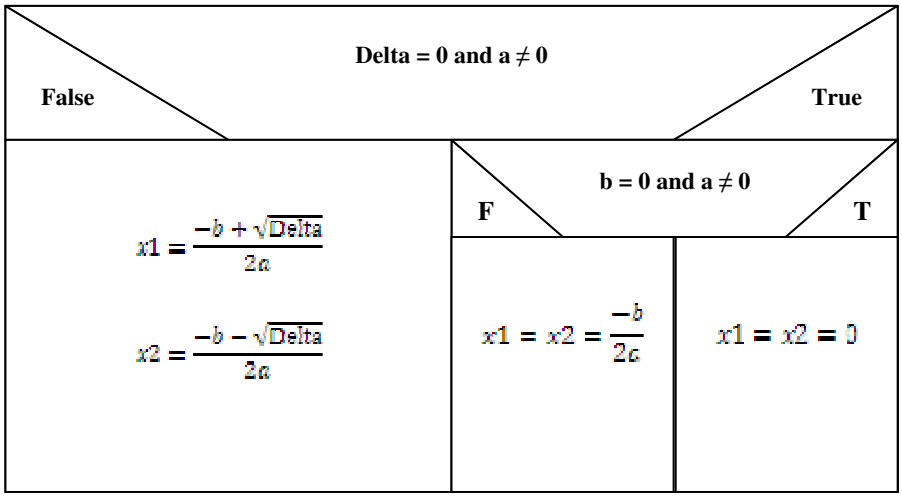
البته چنان که در مثال‌های بعدی خواهید دید هر یک از مسطیل‌های بخش‌های **Else** و **Then** می‌توانند شامل مستطیل های دنباله دستورات باشند.

مثال ۸–۳ : نمودار N-S برای الگوریتمی که با دریافت عدد N قدر مطلق آن را محسبه و چاپ می‌کند.



شکل ۸– ۳۲ : نمودار N-S مثال ۸–۳

مثال ۸–۴ : مثالی که چگونگی نشان دادن شرط‌های تودرتو را در نمودار N-S نشان می‌دهد.

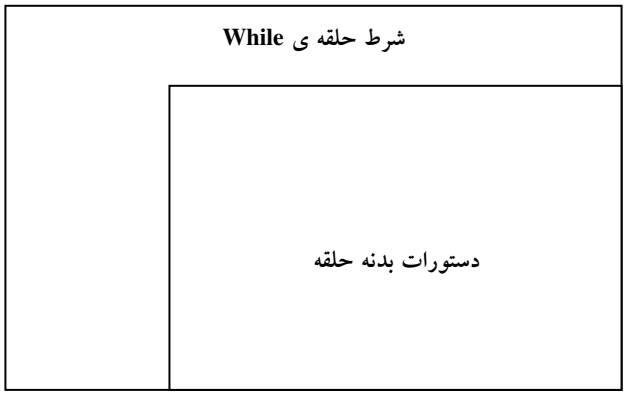


شکل ۸– ۳۳ : نمودار N-S مثال ۸–۴

توضیح مثال : چنان که مشاهده می‌کنید در قسمت Then شرط اول، یک If دیگر داریم و به واسطهٔ دو If تودرتو مقادیر ریشه‌های معادلهٔ درجه دوم را تعیین کرده‌ایم.

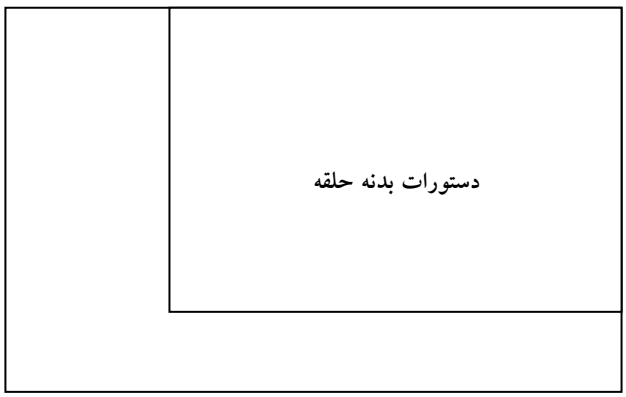
برای نشان دادن ساختارهای تکرار به دو روش می‌توان عمل کرد:

۱. **حلقه‌های While :** در این نوع از حلقه‌ها شرط حلقه در ابتدای حلقه قرار می‌گیرد و تا زمانی که شرط حلقه درست باشد دستورات داخل بدنه حلقه مکرراً تکرار می‌شوند. این ساختار به صورت زیر نشان داده می شود.



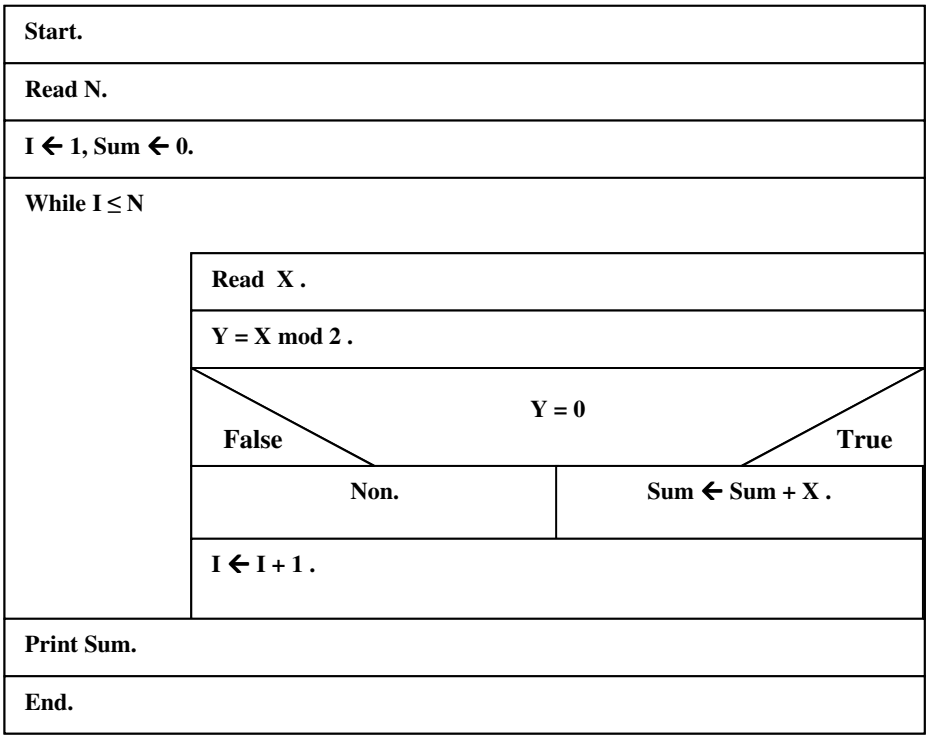
*شکل ۸- ۳۴ : ساختار **While** در نمودار **N-S***

۲. **حلقه های Repeat Until :** در این نوع از حلقه‌ها که معادل حلقه‌های **do…while** در زبان **C++** است شرط حلقه در انتها قرار می‌گیرد و تنها تفاوت آن با حلقه‌های **While** این است که دستورات موجود در بدنه حلقه حداقل یک بار تکرار می‌شود.



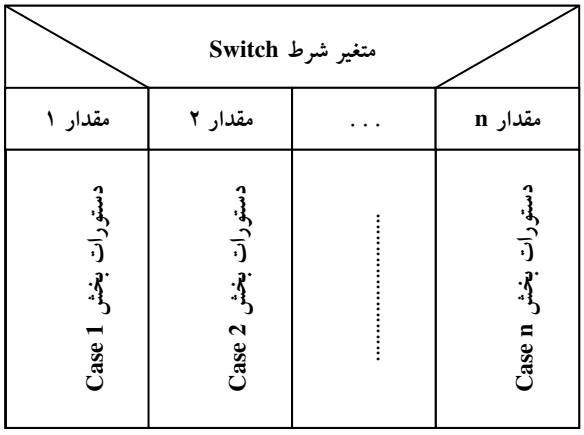
*شکل ۸- ۳۵ : ساختار **Repeat Until** در نمودار **N-S***

مثال ۸–۵ : نمودار N-S برای الگوریتمی که مجموع اعداد زوج N عدد را محاسبه و چاپ می‌کند.



شکل ۸– ۳۶ : نمودار N-S مثال ۸–۵

در نمودار N-S برای ساختارهای چند انتخابی مثل **Switch** در زبان C++ به صورت زیر عمل می‌شود.



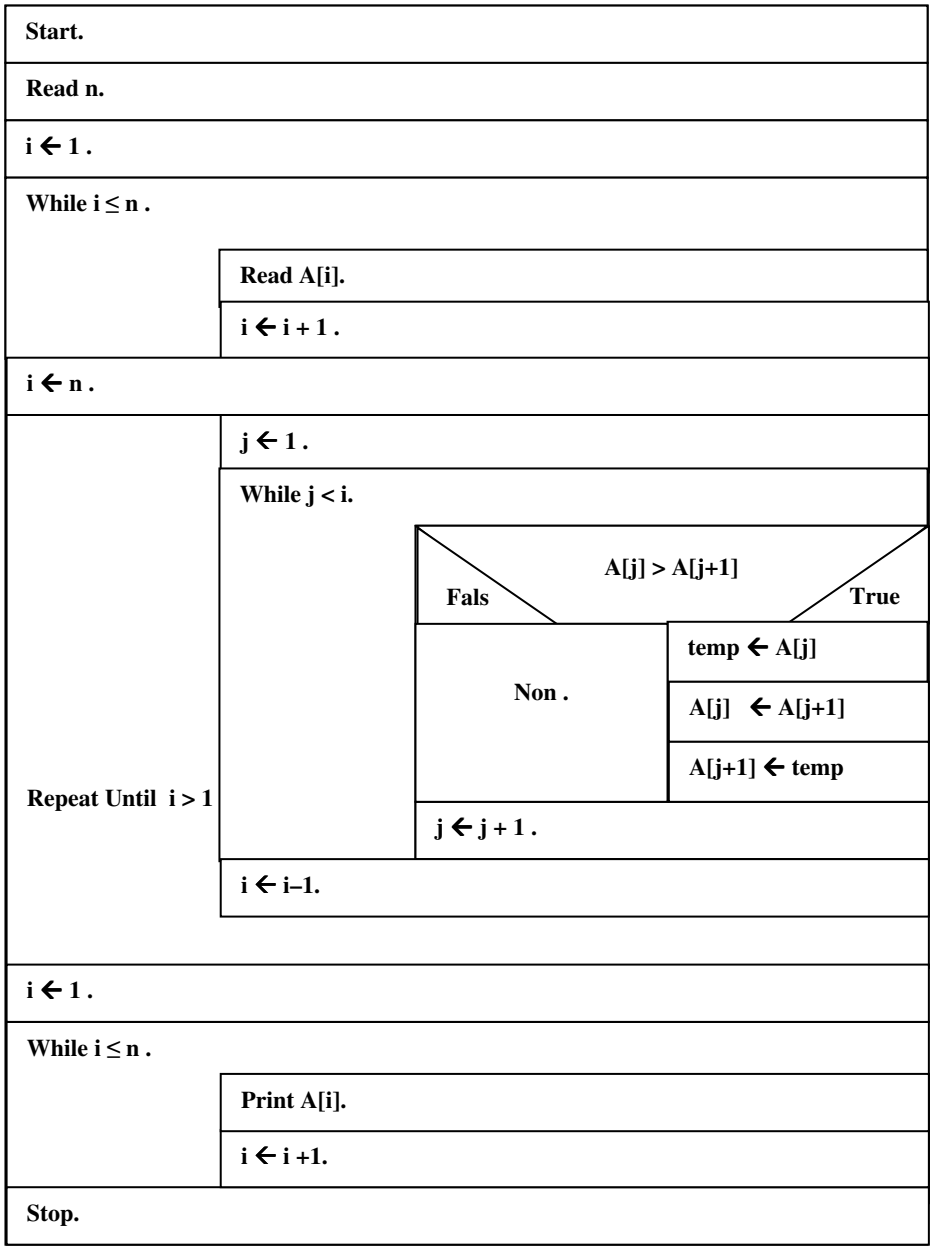
شکل ۸– ۳۷ : ساختار Switch در نمودار N-S

کار در کلاس ۸-۴ :

۱. *یکی از محاسن نمودار* **N-S** *آن است که شما به راحتی می‌توانید ساختارهای مورد نظر خود را در این سیستم ابداع کرده و آن را گسترش دهید. به عنوان مثال سعی کنید ساختاری متناسب با نمودار* **N-S** *و آن چه تاکنون دیدید برای ساختار تکرار* **for** *ابداع کنید. مثلاً یک نمونه ابداعی می تواند به صورت زیر باشد:*

۲. *چه ساختاری را در نمودار* **N-S** *برای فراخوانی توابع پیشنهاد می‌کنید؟ تحقیق کنید که آیا روش استانداردی برای این منظور در نمودار* **N-S** *وجود دارد یا خیر.*

مثال ۸-۶: پیشتر با مرتب سازی حبابی آشنا شدید در زیر نمودار N-S را برای این الگـوریتم مشـاهده مـی کنید.



شکل ۸- ۳۸ : نمودار N-S مثال ۸-۶

کار در کلاس ۸-۵:

نمودار **N-S** *لازم برای یک جستجوی دودویی بر روی آرایه نامرتب* **A[n]** *بنویسید. پس از خواندن آرایه با فراخوانی تابع مرتب‌سازی حبابی فرضی که به عناصر* **A[n]** *دسترسی دارد عناصر این آرایه را مرتب کرده و سپس با خواندن عدد* **x** *اندیس آن را در آرایه مذکور جستجو کرده و در صورت یافتن یا نیافتن عنصر* **x** *در آرایه پیام مناسب را چاپ کنید.*

راهنمایی: در ابتدای نمودار **N-S** *مربوط به یک زیر تابع به جای کلمه* **Start** *نام تابع و به جای کلمه* **End** *در پایان نمودار کلمه* **Return** *باید نوشته شود. و برای فراخوانی تنها کافی است در هر سطری که می‌خواهید تابع را فراخوانی کنید کلمهٔ* **Call** *و سپس نام تابع را بنویسید.*

۸-۱۰: تمرین

۱. نمودار N-S برای الگوریتمی رسم کنید که شماره دانشجویی و معدل دانشجویان یک کلاس را به ترتیب در دو آرایه A و B ذخیره کرده و سپس بالاترین و پایین ترین معدل، معدل میانه را محاسبه و به همراه شماره دانشجویان مربوطه چاپ کند، همچنین معدل کل کلاس را محاسبه و چاپ کند.

۲. یک نمودار N-S برای مرتب سازی آرایه‌ای از اعداد صحیح به روشی غیر از روش حبابی رسم کنید.

۳. نمودار N-S برای الگوریتمی رسم کنید که تعداد N عدد را یکی یکی خوانده و در هنگام خواندن اعداد به گونه ای آن‌ها را در لیست A[N] قرار می‌دهد که در انتها آرایه به صورت نزولی مرتب باشد.

۴. نمودار N-S رسم کنید که مقدار x را دریافت کند و سپس e^x را با توجه به رابطهٔ زیر محاسبه کند.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \quad -\infty < x < \infty$$

۵. نمودار N-S برای الگوریتمی رسم کنید که یک عدد در مبنای M را خوانده و آن را به کمک یک زیر الگوریتم دیگر به مبنای N ببرد. M و N کوچکتر یا مساوی ۱۰ هستند.

۶. نمودار N-S برای الگوریتمی رسم کنید که تعداد n نمره بین صفر تا ۲۰ را از ورودی خوانده و تعداد هر یک از نمرات A تا E معادل نمرات خوانده شده را چاپ کند. بازهٔ نمرات را به صورت زیر در نظر بگیرید:

- 20~17
- 16.99~15
- 14.99~12
- 11.99~10
- 9.99~0

۷. نمودار N-S مربوط به تشخیص عدد اول را رسم کنید.

۸. نمودار N-S برای الگوریتمی رسم کنید که تجزیهٔ یک عدد را به عامل‌های اولش انجام دهد.

۹. نمودار N-S برای الگوریتمی رسم کنید که N عدد را بخواند و سپس دو عدد متوالی که بیشترین اختلاف را نسبت به هر جفت عدد متوالی دیگر دارند را چاپ کند. همین عمل را برای دو عدد با کمترین اختلاف نیز انجام دهید.

۱۰. نمودار N-S برای الگوریتمی رسم کنید که یک عدد را خوانده و بگوید متقارن است یا خیر.

۱۱. آیا می‌توانید کارنمای مربوط به یکی از الگوریتم‌های غیر ساخت یافته یا کارنماهایی که از دو مکان مختلف به End می‌رفتند را که قبلاً با آن‌ها آشنا شدید را بدون هیچ تغییری به نمودار N-S تبدیل کنید. اگر جواب شما منفی است مشکلات موجود را برشمرید و راه حلی ارائه کنید.

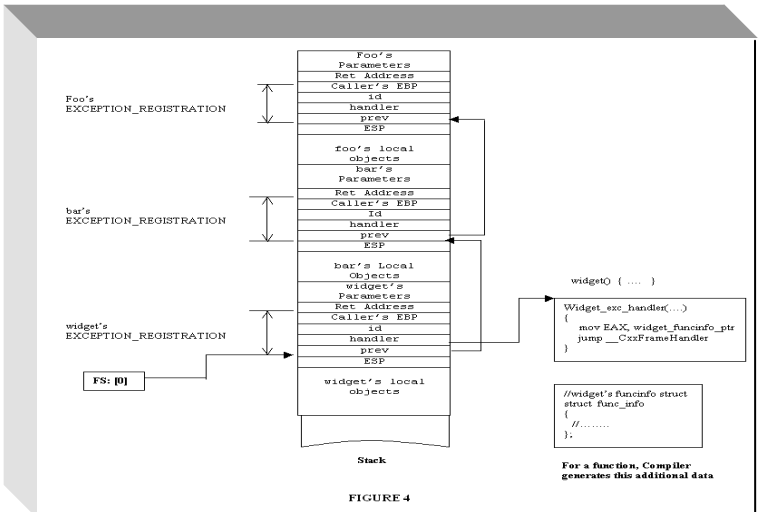


FIGURE 4

فصل نهم

توابع و کلاس‌های حافظه در C++

اهداف فصل و چکیده مطالب :

۱. آشنایی با نحوهٔ تعریف توابع در C++
۲. دسته‌بندی توابع بر حسب مقادیر ورودی و خروجی
۳. آشنایی با نحوهٔ ارسال آرایه‌ها به‌عنوان آرگومان توابع
۴. آشنایی با توابع ریاضی کتابخانه‌ای
۵. سربارگذاری توابع
۶. قالب‌های تابع و توابع کلی
۷. متغیرهای محلی و سراسری
۸. کلاس های حافظه و حوزهٔ متغیرها
۹. آرگومان‌های تابع main

۹-۱ :*تعریف* توابع در C++

مقدمه

برنامه هایی که تا اینجا نوشتیم تنها دارای یک تابع اصلی بوده‌اند. اما از آنجایی که تقسیم هر برنامه به تعدادی کار مشخص، یکی از اصول اساسی برنامه‌نویسی ساخت‌یافته است، لذا در زبان C++ امکان تعریف توابع دیگری جزء تابع اصلی نیز وجود دارد. در حقیقت یک تابع عبارت است از تعدادی از دستورهای برنامه که در یک واحد گرد آمده‌اند و یک کار مشخص را انجام می‌دهند. اصولاً در برنامه‌های طولانی و پیچیده که شامل چندین بخش منطقی و مستقل از هم هستند، برای هر قسمت منطقی، تابعی جداگانه نوشته می‌شود. تابع **main** در این بین نقش هماهنگ‌کننده و مدیریت‌کننده بین توابع مختلف برنامه را برعهده دارد. به عبارت دیگر تابع **main** را می‌توان همانند کارنمای کلی یک الگوریتم در نظر گرفت و توابع دیگر را همانند رویه‌های جزئی یک الگوریتم.

نوشتن توابع

برای نوشتن هر تابع ابتدا باید سه چیز را مشخص کنید:

۱. این تابع چه ورودی‌هایی دارد؟

۲. این تابع چه وظایفی را بر عهده دارد؟ (به زبان ساده‌تر، چه کاری را انجام می‌دهد؟)

۳. این تابع چه خروجی‌هایی یا نتایجی را دارد؟

با دانستن این سه مورد به‌راحتی می‌توان یک تابع را ایجاد کرد و آن را در برنامه استفاده کرد. هر تابع چهار جنبهٔ متفاوت دارد که عبارتند از:

۱. اعلان تابع یا **function declaration** (یا پیش نمونه تابع **function prototype**)

۲. تعریف تابع یا **function definition**

۳. مقدار برگشتی تابع یا **return value**

۴. فراخوانی یا احضار تابع یا **call function**

در ادامه به تشریح تک تک این جنبه‌ها می‌پردازیم:

۱. اعلان تابع : برای آنکه کامپایلر بداند که چه توابعی در یک برنامه وجود دارد و به نوعی با شناسایی این توابع امکان استفاده از این توابع در توابع دیگر برنامه - که قبل از تعریف تابع مذکور قرار دارند - از جمله داخل تابع **main** فراهم شود باید توابع را قبل از تابع **main** اعلان کرد.

آنچه که در اعلان یک تابع برای کامپایلر مشخص می‌گردد شامل موارد زیر است:

الف- نوع تابع : هر تابع باید با دریافت مقادیر ورودی خود، نتیجهٔ حاصل از عملیات برروی ورودی‌ها را به فراخوانندهٔ خود بازگرداند. هر تابع می‌تواند هیچ مقدار، یک مقدار و یا چند مقدار را بازگرداند. اما توابعی که هیچ مقداری را باز نمی‌گردانند از نوع **void** می‌باشند و توابعی که یک مقدار را توسط دستور **return** بازمی‌گردانند هم نوع با نوع بازگشتی خود، می‌باشند. به‌عنوان مثال اگر تابعی یک عدد **int** را باز گرداند، از نوع **int** می‌باشد،

اگر یک مقدار از نوع **char** بازگرداند تابع نیز از نوع **char** تعریف می‌شود و ... . اما توابعی که چند مقدار را باز می‌گردانند مربوط به بحث اشاره‌گرها و مرجع می‌شود که در فصل یازدهم مورد بررسی قرار می‌گیرند. فقط توجه شما را به این نکته جلب می‌کنیم که در این دسته از توابع نوع متغیرهایی که تابع بازمی‌گرداند تأثیری بر نوع تابع ندارد، جز آنکه یکی از مقادیر بازگشتی را با دستور **return** باز گرداند. لذا عموماً این توابع را از نوع **void** میگیرند و تمامی مقادیر را به واسطهٔ اشاره‌گر و یا مرجع باز می‌گردانند که دیگر نیازی به آوردن دستور **return** نباشد. بنابراین به‌عنوان یک قاعدهٔ کلی می‌توان چنین عنوان کرد که نوع یک تابع مشابه نوعی است که توسط دستور **return** باز می‌گردد، و اگر تابع هیچ دستور **return** ای نداشته باشد و یا آنکه توسط دستور **return** هیچ نوعی را باز نگرداند و تنها کنترل اجرای برنامه را به فراخوانندهٔ خود بازگرداند نوع تابع از نوع **void** معرفی میشود.

ب– نام تابع : هر تابع باید دارای یک نام باشد. نامگذاری توابع از قوانین نامگذاری متغیرها پیروی می‌کند. از آنجا که احضار (یا فراخوانی) توابع توسط نام توابع صورت می‌پذیرد، بهتر است نام تابع معرف عملیاتی باشد که آن تابع انجام می‌دهد، تا کد برنامه برای افراد دیگری که آن را می‌خوانند روشن و بدون ابهام باشد. به‌عنوان مثال اگر تابعی عملیات جمع را انجام می‌دهد بهتر است نام آن را **ADD** یا **JAM** انتخاب کنید.

ج– نوع آرگومان‌های تابع: چنانکه پیشتر نیز ذکر شد هر تابع در ریاضیات ممکن است یک یا چند ورودی داشته باشد. اما در C++ تابع حتی می‌تواند بدون هیچگونه مقدار ورودی باشد. چیزی که در اعلان تابع بسیار مهم است تعیین تعداد و نوع مقادیر ورودی (آرگومان‌های) تابع است. بنابراین چنانکه در مثال‌های زیر خواهید دید نوع آرگومان‌های تابع را به ترتیب در بین دو پرانتز ذکر می‌کنیم و آنها را با علامت کاما از یکدیگر جدا می‌کنیم. در C++ استاندارد تأکید شده که اگر تابعی هیچ مقدار ورودی ندارد، کلمهٔ **void** را به‌عنوان نوع آرگومان‌های آن تابع ذکر کنیم، تا عدم وجود آرگومان‌های تابع به نحو مطلوب‌تری مشخص گردد. شکل کلی اعلان یک تابع به صورت زیر است:

نوع تابع نام تابع (نوع اولین آرگومان , نوع دومین آرگومان , ... , نوع nامین آرگومان) ;

دوباره متذکر می‌شویم که اعلان توابع باید قبل از تابع **main** ذکر گردد.

مثال ۹–۱ : در زیر اعلان چند تابع مختلف را مشاهده می‌کنید.

```
void sum(int, int);
```

در این نمونه تابعی با نام **sum** اعلان شده که دو آرگومان ورودی از نوع **int** دارد و هیچ مقدار بازگشتی ندارد. چرا که تابع از نوع **void** اعلان شده است. (احتمالاً این تابع حاصل جمع را تنها در صفحه نمایش چاپ می‌کند لذا هیچ مقدار بازگشتی ندارد.)

```
float DIV(int, float);
```

در این نمونه تابعی با نام **DIV** اعلان شده که دو آرگومان ورودی دارد. اولین آرگومان از نوع **int** و دومین آرگومان از نوع **float** می‌باشد. همچنین مقداری که توسط دستور **return** در این تابع بازگردانده می‌شود، عددی از نوع **float** است.

```
void Print(void);
```

در این نمونه تابعی با نام **Print** اعلان شده، که نه تنها هیچگونه آرگومان ورودی ندارد، بلکه هیچ مقداری را هم باز نمی‌گرداند.

۲. تعریف تابع: مقصود از تعریف یک تابع پیاده‌سازی الگوریتم یک تابع به واسطهٔ دستورات زبان C++ است. اما باید توجه داشته باشید که تعریف هیچ تابعی در داخل تابع دیگری امکانپذیر نیست و محل تعریف توابع پس از تابع اصلی می‌باشد. شکل کلی تعریف توابع و به‌کارگیری آنها در شکل ۹-۱ به‌صورت نمادین نشان داده شده است:

```
#include <iostream>
using namespace std;
//function prototype or declaration
int sample (نوع آرگومان اول , نوع آرگومان دوم , ... , نوع آرگومان n- اُم);
int main()
{
     دستورات تابع اصلی
     return 0;
}
//function definition
int sample (نوع آرگومان اول نام اولین متغیر , نوع آرگومان دوم نام دومین متغیر , ... )
{
   دستورات بدنهٔ تابع
  return     int یک مقدار;
}
```

شکل ۹- ۱ : شیوهٔ به‌کارگیری توابع در یک برنامه

چنانکه در شکل فوق مشاهده می‌کنید تعریف تابع **sample** پس از تابع **main** قرار گرفته است.

جهت تعریف یک تابع ابتدا باید نوع تابع و نام تابع را ذکر کنید و سپس دستورات بدنهٔ تابع در داخل یک بلاک قرار دهید، و آخرین دستور تابع نیز یک دستور **return** است که باید مقداری مطابق نوع تابع به فراخواننده خود باز گرداند. هنگامی که در اعلان تابع نام آرگومان‌ها را به دنبال نوع آنها ذکر می‌کنیم، در حقیقت داریم یک متغیر تعریف می‌کنیم تا به شکل یک ظرف آمادگی ریخته شدن مقادیر ورودی تابع را در خود داشته باشند. به‌عنوان مثال تابع **gotoxy** را در نظر بگیرید، به‌عنوان مثال هنگامی‌که به‌واسطهٔ این تابع مکان‌نما را به ردیف دهم و ستون بیستم منتقل می‌کنیم باید بنویسیم:

```
gotoxy(20,10);
```

اما این مقادیر ۲۰ و ۱۰ که هنگام فراخوانی این تابع به‌عنوان ورودی به تابع ارسال شده در کجا باید قرار گیرند تا تابع بتواند از آنها استفاده کند و مکان‌نما را به نقطهٔ موردنظر هدایت کند؟ بنابراین باید در هنگام تعریف تابع حتماً نام متغیر متناظر با هر پارامتر را ذکر کنیم و در حقیقت آن متغیر را به‌عنوان بخشی از تابع تعریف در نظر بگیریم. به مقادیر ۲۰ و ۱۰ که در هنگام فراخوانی تابع ذکر می‌شوند، پارامترهای تابع و به متغیرهای **x** و **y** که مقادیر ورودی را در خود ذخیره می‌کنند آرگومان‌های تابع گفته می‌شود.

۳. فراخوانی تابع: با فراخوانی تابع تا حدود زیادی آشنا هستید، و با توابع زیادی از جمله توابع کتابخانه‌ای زبان C++ کارکرده‌اید. در حقیقت فراخوانی یک تابع چیزی جز ذکر نام تابع و ارسال مقادیر ورودی موردنظر به تابع نیست. توجه داشته باشید که هر تابعی قادر است تابع دیگری را فراخوانی کند، به شرط آنکه اعلان تابع فراخوانده شده قبل از تابع فراخواننده موجود باشد. لذا از نقطه‌ای که اعلان یک تابع قرار دارد این تابع برای تمامی توابع دیگر برنامه قابل شناسایی و فراخوانده شدن است. و نکتهٔ قابل توجه دیگر آنکه هر تابع فرعی را می‌توان از داخل تابع فرعی دیگری فراخوانی کرد، و لذا تنها تابع **main** است که توسط توابع برنامه قابل فراخوانی نیستند.

دسته‌بندی توابع

توابع را می‌توان از نظر پارامترهای ورودی و یا از نظر مقادیر خروجی دسته‌بندی کرد.

۱. انواع تابع از نظر پارامترهای ورودی : (انواع فراخوانی تابع)

از این منظر توابع بر دو دسته تقسیم می‌شوند:

الف– توابعی که با مقدار فراخوانی می شوند: در این روش مقداری برای هر آرگومان به تابع ارسال می‌شود، و پارامترهای ارسالی در آرگومان‌های تابع کپی می‌شود(عمل انتساب صورت می‌گیرد). لذا هرگونه تغییری در آرگومان-های تابع هیچ اثری در متغیرهایی که در هنگام فراخوانی به‌عنوان پارامتر به تابع ارسال شده‌اند، ندارد.

ب– توابعی که با آدرس فراخوانی می شوند: در این روش از فراخوانی تابع، به‌جای ارسال مقادیر پارامترهای ورودی، آدرس متغیرهایی که به‌عنوان پارامتر در هنگام فراخوانی تابع ذکر شده‌اند به تابع ارسال می‌شود، لذا در این حالت اعمال هر گونه تغییری برروی آرگومان‌های تابع برروی متغیرهای مذکور نیز تأثیر می‌گذارد. فراخوانی توابع به شیوهٔ اخیر به وسیلهٔ اشاره‌گرها یا مرجع امکان‌پذیراست، که در فصل یازدهم مورد بررسی قرار می‌گیرد. لذا در این فصل تنها به بررسی روش فراخوانی با مقدار می‌پردازیم.

البته می توان توابع را از نظر تعداد مقادیر ورودی نیز دسته بندی کرد اما از نظر برنامه‌نویسی هیچ ارزش آموزشی ندارد.

۲. انواع تابع از نظر تعداد مقادیر خروجی

توابع از نظر تعداد مقادیر خروجی حائز اهمیت بوده و به سه دستهٔ زیر تقسیم می‌شوند.

الف– توابعی که هیچ مقداری را بر نمی‌گردانند: این دسته از توابع از نوع **void** اعلان می‌شوند و کارهایی از قبیل چاپ اطلاعات در خروجی را انجام می‌دهند، که نیازی به برگرداندن مقادیر نداشته باشد.

ب– توابعی که یک مقدار را توسط دستور return بر می‌گردانند: این توابع اصولاً یک مقدار متناسب با نوع تابع به مجری(فراخوانندهٔ) خود توسط دستور **return** باز می‌گردانند.

ج– توابعی که چندین مقدار را برمی گردانند: برای پیاده سازی این دسته از توابع باید از اشاره‌گرها یا مرجع استفاده کنیم، و چنانکه پیشتر مطرح شد بحث اصلی در خصوص این توابع تا فصل یازدهم به تعویق می‌افتد. اما یک روش دیگر برای برگرداندن چندین مقدار، استفاده از آرایه‌ها است که در همین فصل مورد بررسی قرار می‌گیرد.

در ادامه به بررسی مثال‌هایی از توابع مختلف می‌پردازیم.

بررسی توابعی که هیچ مقداری را به فراخوانندهٔ خود برنمی‌گردانند (توابع نوع void)

مثال ۹-۲: برنامه‌ای که تابعی از نوع **void** را بدون هیچگونه آرگومان را نشان می‌دهد.

```
1.  //This is a sample for usage of void function
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  void Print(void);    // function prototype or declaration
6.  int main()
7.  {
8.      Print();        // Call function
9.    return 0;
10. }
11. //function definition
12. void Print(void)
13. {
14.    string name;
15.     cout<<"Enter your name : ";
16.      cin>>name;
17.        cout<<"Hello "<<name<<" !.\n";
18.    return;
19. }
```

توضیح مثال : در خط ۵ با اعلان تابع، در خط ۸ با احضار تابع و در خطوط ۱۲ الی ۱۹ با تعریف تابع روبرو هستیم. دستور **return** نوشته شده در خط ۱۸ اختیاری است و تنها جهت بازگرداندن کنترل اجرای برنامه به فراخوانندهٔ تابع است. هنگامی که کنترل به خط هشتم کد یعنی دستور فراخوانندهٔ تابع می‌رسد، کنترل اجرای برنامه به ابتدای خط ۱۲ منتقل شده و اجرای برنامه را از این نقطه از سر گرفته تا هنگامی که به انتهای بلوک دستورات تابع **Print** برسد، کنترل اجرای برنامه به بعد از فراخوانی تابع باز گشته و اجرای برنامه از خط نهم ادامه پیدا می‌کند.

مثال ۹-۳: برنامه‌ای که یک عدد را دریافت کرده آن را به عامل‌های اول تجزیه می‌کند.

```
1.  //This is a sample for usage of function
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  void Primes(unsigned int);
6.  int main()
7.  {
8.       int num;
9.        cout<<"Please enter an unsigned integer number : ";
10.        cin>>num;
11.         Primes(num);
12.    return 0;
13. }
```

```
14. //function definition
15. void Primes(unsigned int N)
16. {
17.       for(int k=2; k*k<=N; k++)
18.         while(N%k == 0)
19.            {
20.               cout<<k<<endl;
21.                N=N/k;
22.            }
23.       if(N!=1)
24.         cout<<N<<endl;
25. }
```

توضیح مثال : در این مثال تابع **primes** را با یک آرگومان از نوع **unsigned int** می‌بینید که در خط یازدهم مقدار **num** به‌عنوان پارامتر ورودی به تابع ارسال شده، و مقدار ارسال شده در متغیر **N** ریخته می‌شود. با الگوریتم این تابع در فصل قبل آشنا شدید لذا توضیح خاصی در مورد این برنامه نمی‌دهیم.

مثال ۹-۴ : برنامه‌ای که در فراخوانی با مقدار، چگونگی تغییر در آرگومان‌ها و عدم‌تأثیر آن را در پارامترهای ورودی نشان می‌دهد.

```
1.  //This program shows how to Call a function with value
2.  #include <iostream.h>
3.  void f1(int, int);
4.  int main()
5.  {
6.     int x, y;
7.      cout<<"Enter two integer numbers:";
8.       cin>>x>>y;
9.        cout<<"You entered : x="<<x<<", y="<<y;
10.     f1(x, y);
11.       cout<<"\nAfter return from f1 : x="<<x<<", y="<<y;
12.        cout<<endl;
13.   return 0;
14. }
15. //function definition
16. void f1(int x, int y)
17. {
18.    cout<<"\nf1 receives : x="<<x<<" , y="<<y;
19.     x++;
20.     y++;
21.    cout<<"\nNew values in f1 : x="<<x<<", y="<<y;
22. }
```

توضیح مثال : در این برنامه، دو مقدار **x** و **y** از ورودی خوانده شده، مقادیر فعلی آنها در برنامه اصلی چاپ می‌شود. سپس این مقادیر، به‌عنوان پارامتر به تابع **f1** ارسال می‌شوند و **f1** نیز آنها را چاپ می‌کند. سپس مقادیر آرگومان‌ها در تابع **f1** تغییر کرده، محتویات جدید آنها چاپ می‌شود. پس از برگشت از فراخوانی تابع، محتویات **x** و **y** مجدّداً چاپ

می‌شوند. چاپ این محتویات نشان می‌دهد که تغییراتی که در آرگومان‌ها ایجاد شد، تأثیری در پارامترهای ورودی ندارد. به عبارت دیگر متغیرهای x و y تعریف شده در خط ششم کد با متغیرهای تعریف شده در خط شانزدهم کد به کلی متفاوت هستند. و در خط دهم تنها مقادیر متغیرهای x و y داخل تابع main به متغیرهای x و y داخل تابع f 1 انتساب می‌یابد. خروجی این برنامه به‌صورت زیر خواهد بود.

```
Enter two integer numbers : 10  15
You entered: x=10, y=15
f1 receives: x=10 ,y=15
New values in f1: x=11, y=16
After return from f1: x=10, y=15
```

بررسی توابعی که یک مقدار را به فراخوانندهٔ خود برمی‌گردانند (غیر void)

در بسیاری از مسائل نیاز به نوشتن توابعی است که یک مقدار را برگردانند. مثل تابع sin() که سینوس یک زاویه را برمی‌گرداند. اینگونه توابع، کاربردهای فراوانی دارند که در ادامه مثال‌های متنوعی از آنها را خواهید دید. برای نوشتن اینگونه توابع، نوع آنها را باید در الگوی تابع در هنگام اعلان و عنوان تابع در هنگام تعریف مشخص کرد. برای برگرداندن مقداری توسط تابع، از دستور return استفاده می‌شود. مقداری که توسط این دستور برگشت داده می‌شود، به‌جای دستور فراخوانی تابع قرار می‌گیرد. لذا در تابع فراخواننده، می‌توان نام تابع را به متغیری نسبت داد و پس از اجرای تابع از محتویات متغیر استفاده کرد، زیرا در این حالت متغیر سمت چپ دستور انتساب حاوی مقدار برگشتی تابع است. به‌عنوان مثال، اگر f1 یک تابع از نوع int و x متغیری از نوع int باشد، دستور زیر، تابع f1 را فراخوانی کرده، مقداری را که توسط دستور return به‌جای نام تابع قرار می‌گیرد، در متغیر x قرار می‌دهد. در اینجا تابع f 1 فاقد آرگومان است:

```
int x = f1();
```

مثال ۹-۵: برنامه‌ای که به‌وسیلهٔ یک تابع مربع یک عدد را محاسبه کرده و نمایش می‌دهد.

```
1.  //This program uses a function that return a value.
2.  #include <iostream>
3.  using namespace std;
4.  unsigned int sq(int X);
5.  int main()
6.  {
7.     int x;
8.      cout<<"Enter an integer number:";
9.       cin>>x;
10.    int square = sq(x);
11.     cout<<"The square of "<<x<<" is "<<square<<endl;
12.    return 0;
13. }
14. //function definition
15. unsigned int sq(int X)
16. {
17.     return (unsigned int)X*X;
18. }
```

توضیح مثال : در این برنامه از یک تابع با نام **sq** استفاده شده، که با دریافت یک عدد به‌عنوان پارامتر ورودی مقدار مربع آن را برمی‌گرداند. مقدار برگشت شده توسط این تابع در متغیر **square** که در خط ۱۰ تعریف شده است، قرار می‌گیرد. نکتهٔ قابل‌توجه دیگر آنکه برای اولین بار در اعلان تابع در خط چهارم این کد نام آرگومان تابع آورده شده است. در حقیقت آوردن این نام صرفاً می‌تواند در هنگام نوشتن دستور احضار تابع ما را در تشخیص صحیح پارامترها کمککند و هیچ ارزش دیگری ندارد. لذا از ابتدا برای آنکه دانشجویان عزیز متوجه این مطلب باشند که تعریف آرگومان-های تابع تنها در عنوان تابع (در هنگام تعریف تابع) صورت می‌گیرد از قرار دادن نام متغیرهای آرگومان در اعلان تابع خودداری کردیم.

مثال ۹-۶ : برنامه‌ای که عددی را از ورودی خوانده به تابعی تحویل می‌دهد. تابع تشخیص می‌دهد که عدد موردنظر اول است یا خیر. اگر عدد موردنظر اول باشد، مقدار یک (ارزش درستی) و گرنه مقدار صفر (ارزش نادرستی) را برمی‌گرداند. سپس برنامه برای ادامه کار، از کاربر سؤال می‌کند. اگر پاسخ کاربر مثبت ('y') بود، برنامهٔ عدد بعدی را دریافت می‌نماید، و اگر پاسخ کاربر منفی بود برنامه خاتمه می‌یابد. نکتهٔ قابل‌توجه دیگر آنکه چون فضای نام استاندارد در تابع **main** معرفی شده در خارج از این تابع یعنی در داخل تابع **prime** شئ **cout** یا **cin** شناخته نمی‌شوند.

```
1.  //This program tells,the arriaval number is prime or not.
2.  #include "iostream"
3.  bool prime(int);
4.  int main(){
5.    using namespace std;
6.    int num;
7.     char ans;
8.      while(1)
9.       {
10.         cout<<"\n Enter a number:" ;
11.          cin>>num;
12.           if(prime(num))
13.              cout<<" Number "<<num<<" is prime.\n";
14.           else
15.              cout<<" Number "<<num<<" is not prime.\n";
16.         cout<<"\n Do you want to continue?(y/n):";
17.           cin>>ans;
18.         if(ans != 'y')
19.           break;
20.       }// end of while
21.    return 0;
22. }
19. // function definition
23. bool prime(int num)
24. {
25.     bool temp = 1;
26.       for(int i=2; (i <= num / 2) && temp ; i++)
27.          if(num % i == 0)
28.              temp = false;
29.     return temp;
30. }
```

مثال ۹-۷ : برنامه‌ای که عملکرد سوئیچ **Caps Lock** را به‌واسطهٔ یک تابع دریافت و یک تابع چاپ بازسازی می‌کند.

```
1. //This program simulates Caps Lock key.
2. #include <iostream>
3. #include <cstring>
4. #include <conio.h>
5. using namespace std;
6. char input();
7. char caps_lock(char);
8. void output(char);
9. int main()
10. {
11.     char ch=0;
12.     while((ch=input())!='$')
13.             output(ch);
14.    return 0;
15. }
16. //********************
17. char input()
18. {
19.     char ch = getch();
20.     return ch;
21. }
22. //********************
23. char caps_lock(char ch)
24. {
25.     if(islower(ch))
26.        ch=toupper(ch);
27.     return ch;
28. }
29. //********************
30. void output(char ch)
31. {
32.    cout.put(caps_lock(ch));
33. }
```

توضیح مثال : این برنامه اولین برنامه‌ای است که در آن از چندین تابع استفاده کرده‌ایم. تابع **input** یک کاراکتر را از کاربر دریافت کرده و کاراکتر دریافتی را برمی‌گرداند. لذا در یک حلقهٔ تکرار تا زمانی‌که کاربر کاراکتر '$' را وارد کند به دریافت کاراکترها ادامه می‌دهیم. تابع **output** کاراکتر دریافتی را چاپ می‌کند. اما در داخل این تابع قبل از چاپ کاراکتر دریافتی در صورتی‌که کاراکتر یک حرف کوچک باشد توسط تابع **caps_lock** آن کاراکتر را به حرف بزرگ تبدیل می‌کنیم. در داخل تابع **caps_lock** ابتدا چک می‌کنیم که آیا حرف دریافتی یک حرف کوچک است یا نه؟ اگر حرف دریافتی کوچک بود توسط تابع **toupper** آن را به حرف بزرگ معادلش تبدیل می‌کنیم. توجه کنید که متغیر **ch** تعریف شده در هر تابع مستقل از دیگر متغیرهای **ch** است.

آرایه‌ها به‌عنوان آرگومان توابع

همانگونه که متغیرها را به توابع ارسال می‌کنیم، می‌توان آرایه‌ها را نیز به توابع ارسال کرد. مزیت ارسال یک آرایه به تابع آن است که ارسال تعدادی مقدار به تابع راحت‌تر و بدون ذکر تعداد زیادی پارامتر امکانپذیر می‌شود. همچنین چنانکه پیشتر اشاره شد نام هر آرایه، اشاره‌گری است به اولین خانه از آرایه، لذا فراخوانی توابع به‌همراه آرگومان‌های آرایه‌ای، از نوع فراخوانی با آدرس بوده و هرگونه اعمال تغییرات برروی عناصر آرایهٔ داخل تابع برروی آرایه‌ای که در هنگام فراخوانی به‌عنوان پارامتر ورودی ذکر شده باشد تأثیر می‌گذارد. لذا به واسطهٔ آرایه‌ها می‌توان چندین مقدار هم نوع را توسط توابع به فراخوانندهٔ تابع بازگرداند.

۱) آرایه های یک بعدی به‌عنوان آرگومان تابع

در مورد توابعی که آرگومانی از نوع آرایه دارند باید به نکات زیر توجه کنید:

الف- هنگامی که می‌خواهیم آرایه‌ای یک بعدی را به تابعی ارسال کنیم باید علاوه بر نوع آرایه علامت [ ] را نیز به‌عنوان نماد آرایه پس از نوع آرایه ذکر کنیم، تا کامپایلر قادر به تشخیص آرگومانی که از نوع آرایه است بشود. به‌عنوان مثال در اعلان زیر یک آرایهٔ یک‌بعدی را به همراه یک متغیر **float** به تابعی با نام **f** ارسال کرده‌ایم:

```
int f(float, int[]);
```

ب- اما در هنگام تعریف تابع باید علاوه بر ذکر نام آرایه، علامت [ ] را نیز در عنوان تابع پس از نام آرایه بیاوریم. به‌عنوان مثال تعریف تابع **f** را ببینید:

```
int f(float fl, int a[])
{
        .
        .
        .
}
```

ذکر طول آرایه در بین دو علامت [ ] هیچ اهمیتی ندارد اما اگر در اعلان و یا در تعریف تابع طول آرایه را نیز بین این دو علامت ذکر کنیم هیچ مشکلی پیش نمی‌آید. اما اصولاً چون قرار است یک تابع مثل **f** برروی آرایه‌های متفاوتی عمل مشابهی را انجام دهد، طول آرایه را به‌عنوان یکی از پارامترهای تابع به آن ارسال می‌کنند تا عملیاتی چون پردازش آرایه برحسب متغیری که معرف طول آرایه است پی‌ریزی شود. به‌عنوان مثال در مثال ۹-۸ برنامه‌ای ارائه شده که میانگین اعداد دو آرایه با طول‌های متفاوت را توسط یک تابع محاسبه می‌کند.

ج- در هنگام فراخوانی تابع تنها آوردن نام تابع بدون هیچگونه علامت کروشه یا امثال آن کفایت می‌کند. به‌عنوان مثال فراخوانی تابع **f** می‌تواند به‌صورت زیر باشد:

```
int x[10];
  f(5.2, x);
```

مثال ۹-۸: برنامه‌ای که میانگین عناصر دو آرایه با طول‌های متفاوت را توسط یک تابع حساب می‌کند.

```
1. //This program uses array as argument.
2. #include <iostream>
3. #include <cstring>
4. #include <conio.h>
5. using namespace std;
6. float AVG(int [], int);
7. void show(float, int [], int);
8. int main()
9. {
10.     const int A_Size = 5;
11.     const int B_Size = 8;
12.        int A[A_Size]={1,3,78,43,20};
13.        int B[B_Size]={32,1,43,8,2,0,91,-5};
14.     float avg;
15.       avg=AVG(A,A_Size);
16.         show(avg,A,A_Size);
17.       avg=AVG(B,B_Size);
18.         show(avg,B,B_Size);
19.     return 0;
20. }
21. //********************
22. float AVG(int ar[],int len)
23. {
24.    int sum=0;
25.     for(int i=0; i<len; i++)
26.        sum+=ar[i];
27.    return ((float)sum / len);
28. }
29. //********************
30. void show(float avg, int ar[], int len)
31. {
32.     cout<<"The average of ";
33.      for(int i=0; i<len; i++)
34.           cout<<ar[i]<<" , ";
35.     cout<<"\nis = "<<avg<<endl;
36. }
```

توضیح مثال: در این برنامه مشاهده می‌کنید که با ارسال طول آرایه به تابع عملیات مشابهی را برروی دو تابع با طول‌های متفاوت انجام داده‌ایم. تابع **AVG** جهت محاسبهٔ میانگین عناصر آرایه و تابع **show** جهت نمایش خروجی طراحی شده است.

مثال ۹-۹: برنامه‌ای که با ارسال یک آرایه به یک تابع و انجام مرتب‌سازی حبابی برروی عناصر آرایهٔ نشان می‌دهد که عملیات برروی عناصر آرایهٔ داخل زیر برنامه بر آرایهٔ ارسال شده به‌عنوان پارامتر تأثیر می‌گذارد. زیرا آرایه‌ها به وسیلهٔ فراخوانی با آدرس به توابع ارسال می‌شوند.

```
1.  //Bubble sort program
2.  #include <iostream.h>
3.  void input(int [], int);
4.  void bubble(int [], int);
5.  void output(int [], int);
6.  int main()
7.  {
8.     const int k = 7 ;
9.      int temp[7];
10.      input(temp, k);
11.       bubble(temp, k);
12.        cout << "The sorted data are :\n" ;
13.         output(temp, k);
14.    return 0;
15. }
16. //**************************
17. void input(int temp[], int len)
18. {
19.    int i;
20.     for(i=0; i<len; i++){
21.        cout << "Enter number " << (i + 1) << " :";
22.         cin >> temp[i];
23.       }
24. }
25. //**************************
26. void bubble(int temp[], int len)
27. {
28.    int i, j, item;
29.     for(i=len-1; i>0; i --)
30.       for(j=0; j<i; j++)
31.         if(temp[j]>temp[j+1]){
32.             item = temp[j] ;
33.             temp[j] = temp[j+1];
34.             temp[j+1] = item ;
35.           }//end of if
36. }
37. //**************************
38. void output(int temp[], int len)
39. {
40.    int i;
41.     for(i=0; i<len; i++)
42.        cout<<temp[i]<<" ";
43. }
```

۲) آرایه‌های چند بعدی به‌عنوان آرگومان تابع

هنگامی می‌خواهیم آرایه‌ای چند بعدی را به تابعی ارسال کنیم باید به ازای هر بعد ماتریس یک علامت [ ] در اعلان تابع و متقابلاً در عنوان تابع قرار دهیم. ذکر ابعاد ماتریس در بین کروشه‌ها الزامی است و تنها می‌توان اولین بعد ماتریس را ذکر نکرد که عبارت است از تعداد ردیف‌های ماتریس. به‌عنوان مثال جهت ارسال آرایه‌ای با **n** ردیف و ۱۰۰ ستون می‌توان به‌صورت زیر عمل کرد.

```
void f(int [][100],int n);
```

از نظر فراخوانی تابع، در اینجا نیز تنها ذکر نام آرایه در دستور احضار کنندهٔ تابع کفایت می‌کند.

مثال ۹-۱۰: برنامه‌ای که دو آرایه (ماتریس) را به‌عنوان آرگومان ورودی دریافت کرده سپس حاصل‌ضرب آنها را در خروجی چاپ می‌کند.

```
1.  //This program calculates multiply of 2 matrix
2.  #include <iostream>
3.  using namespace std;
4.  void mul(int mat1[][3],int mat2[][4],int mat3[][4]);
5.  int main()
6.  {
7.     int mat1[2][3], mat2[3][4], mat3[2][4]={0} ;
8.     int i,j;
9.      //read mat1
10.      for(i=0; i<2; i++)
11.         for(j=0; j<3; j++)
12.         {
13.           cout<<"Enter mat1["<<i+1<<"]["<<j+1<<"]:";
14.           cin >> mat1[i][j] ;
15.         }
16.     //read mat2
17.      for(i=0; i<3; i++)
18.         for(j=0; j<4; j++)
19.        {
20.           cout<<"Enter mat2["<<i+1<<"]["<<j+1<<"]: ";
21.           cin >> mat2[i][j];
22.         }
23.    //multiply mat1 by mat2 and mat3 is for store result.
24.      mul(mat1,mat2,mat3);
25.    //show the result
26.    cout << "\nThe produc of mat1 & mat2 is : \n\n" ;
27.      for(i = 0 ;i < 2 ;i++)
28.      {
29.         for(j = 0 ; j < 4 ;j++)
30.            cout<<mat3[i][j]<<" ";
31.         cout << endl ;
32.      }
33.   return 0;
34. }
```

```
35. void mul(int mat1[2][3],int mat2[3][4],int mat3[2][4])
36. {
37.   int i,j,k;
38.    for(i=0; i<2; i++)
39.      for(j=0; j<4; j++)
40.      {
41.         mat3[i][j]=0 ;
42.         for(k=0 ;k<3 ; k++)
43.             mat3[i][j]=mat3[i][j]+mat1[i][k]*mat2[k][j];
44.      }
45. }
```

حذف اعلان توابع

اگر تعریف یک تابع قبل از تابع **main** قرار گیرد دیگر نیازی به ذکر اعلان آن تابع نیست. چرا که با آمدن تعریف تابع، کامپایلر به‌راحتی از نقطهٔ تعریف به بعد قادر است، آن تابع را شناسایی کند و دیگر نیازی به اعلان تابع نیست.

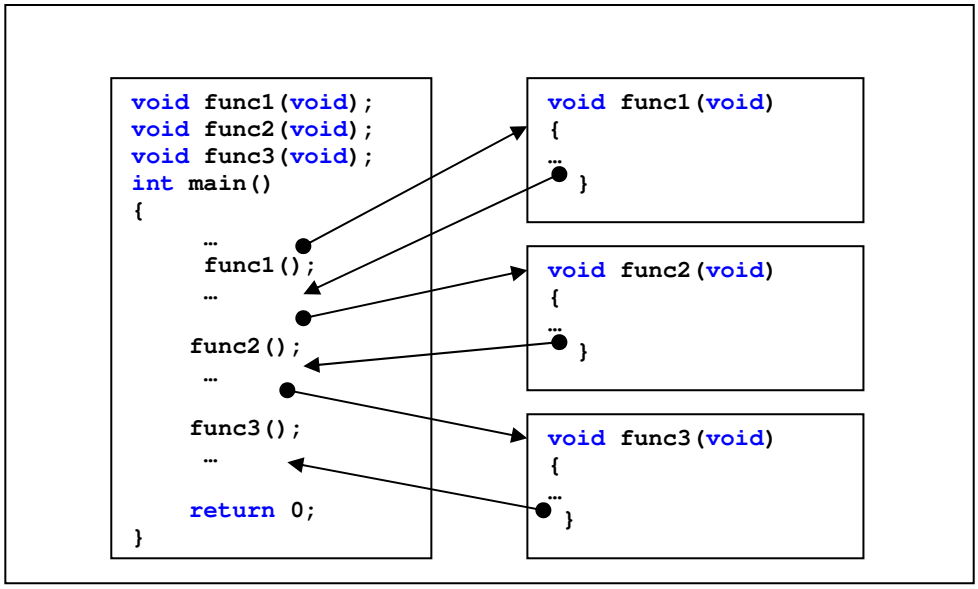
مثال ۹-۱۱: برنامه‌ای که نحوهٔ حذف اعلان توابع را نشان می‌دهد.

```
1.  //demonstrates function definition preceding function call
2.  #include <iostream>
3.  using namespace std;
4.  void starline()
5.  {
6.     for(int i=0; i<49; i++)
7.       cout<<'*';
8.     cout<<endl;
9.  }
10. int main()
11. {
12.   starline();
13.   cout<<"Data type    rang"<<endl;
14.   cout<<"char         -128 to 127\n"
15.       <<"short        -32,768 to 32,767\n"
16.       <<"int          System dependent\n"
17.       <<"long         -2,147,483,648 to 2,147,483,647\n" ;
18.   starline();
19.  return 0;
20. }
```

شاید این روش برای برنامه‌های کوچک مناسب باشد، اما برای برنامه‌های بزرگ پیشنهاد نمی‌شود. با اتخاذ این روش برنامه‌نویس باید دقت کند که توابع را به ترتیب احضار آنها در برنامه بنویسد که گاهی اوقات، این‌کار غیرممکن است. به همین‌خاطر بسیاری ازبرنامه‌نویسان C++ ترجیح‌می‌دهند تابع **main** را به‌عنوان اولین تابع در برنامه بنویسند و از اعلان توابع استفاده‌کنند. زیرا اجرای تمام برنامه‌های C++ ، با اجرای تابع **main** آغاز می‌شود.

توابع inline

هنگامی که کنترل اجرای برنامه به دستور فراخوانی یک تابع می‌رسد، کنترل اجرا به ابتدای مکانی از حافظه که دستورات تابع نوشته شده پرش می‌کند و اجرای برنامه از ابتدای بلوک دستورات تابع آغاز شده و تا زمانی‌که به دستور **return** یا انتهای بلوک دستورات برسد اجرای دستورات داخل تابع دنبال می‌شود. هرگاه که به دستور **return** یا علامت } انتهای بلوک تابع برسد، کنترل اجرای برنامه به پس از دستور فراخوانی تابع بازگشته و ادامه اجرای دستورات برنامه از آنجا ادامه می‌دهد. این روند در شکل نشان داده شده است.



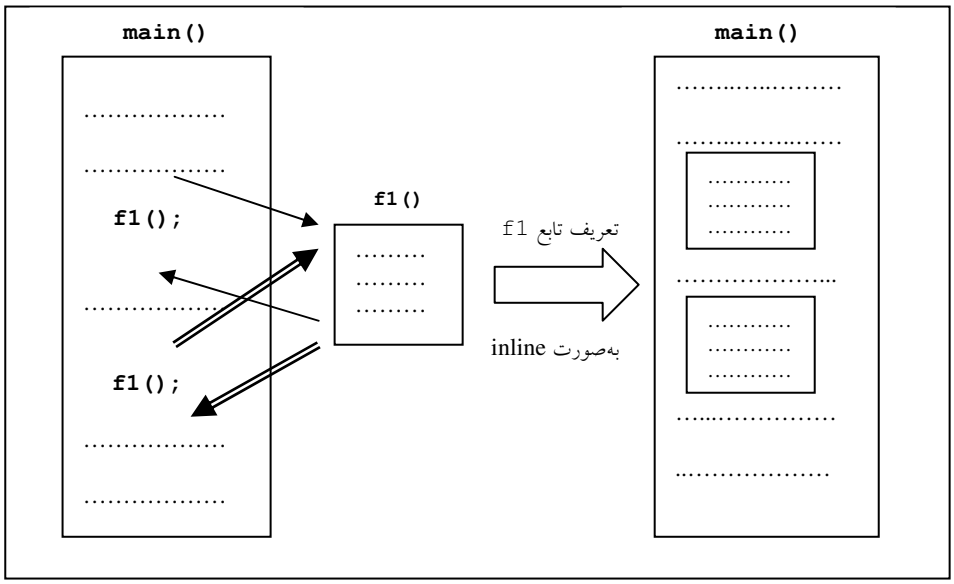
شکل ۹- ۲ : چگونگی عملکرد برنامه در هنگام فراخوانی توابع

اما با آنکه استفاده از توابع موجب می‌شود که حجم کد نویسی و دستورات برنامه کم شود و در نتیجه حجم فایل اجرایی حاصل از کامپایل برنامه نیز کاهش یابد و در میزان مصرف حافظه صرفه‌جویی گردد اما به‌کارگیری توابع و عمل احضار توابع به دلایل زیر موجب کاهش سرعت اجرای برنامه می‌شود. زیرا هنگامی که یک تابع فراخوانی می‌گردد اعمال زیر به ترتیب باید صورت بپذیرد که خود مستلزم صرف زمان می‌باشد. لذا احضار یک تابع شامل مراحل زیر است:

۱. قرار دادن آدرس فعلی که کنترل اجرای برنامه در آن قرار دارد.
۲. باید دستوری برای پرش به ابتدای دستورات تابع اجرا گردد. (مثل دستور **CALL** مربوط به زبان اسمبلی یا هر دستوری شبیه آن)
۳. باید هر آنچه در ثبات‌های **CPU** قرار دارد در مکانی از حافظه موسوم به پشته ذخیره گردد تا هنگام بازگشت از تابعِ فراخوانی شده، بتوان مقادیر فعلی ثبات‌ها را بازیابی کرد.
۴. باید دستوراتی جهت قرار دادن پارامترهای ارسال شده به تابع در پشتهٔ احضار برنامه اجرا گردد.

۵. حذف پارامترهای ارسالی به تابع از پشته و کپی کردن آنها در متغیرهایی که در تعریف تابع به‌عنوان آرگومان تابع در نظر گرفته شده است.
۶. قرار دادن مقدار بازگشتی تابع در پشتهٔ احضار برنامه.
۷. حذف مقدار بازگشتی از پشته و قرار دادن آن در متغیر مربوط به آن در حافظه.
۸. خواندن مقادیر قبلی ثبات‌ها از پشته و پرکردن ثبات‌های CPU با آن مقادیر و حذف این مقادیر از پشته.
۹. خواندن آدرس بازگشت از پشته و اجرای دستوراتی برای پرش به نقطهٔ بعد از دستور فراخوانی تابع.

با توجه به آنچه گفته شد برنامه‌نویس باید بین بهره‌گیری از مزایای نوشتن توابع و یا صرف زمان بیشتر برای اجرای برنامه یکی را انتخاب کند. اما در بسیاری از مواقع نه می‌توان بجای استفاده از توابع، مدام کدهای مربوط به یک تابع را تکرار کرد و نه می‌توان صرف چنین زمان زیادی را تحمل نمود، لذا اصولاً در توابع کوچک بهتر است با ذکر کلمهٔ **inline** تنها در موقع فراخوانی تابع (و نه اعلان آن) کامپایلر را وادار کنیم که هر جا تابع موردنظر فراخوانی شده است کدهای مربوط به بدنهٔ تابع را جایگزین دستور فراخوانی تابع بکند. بدین ترتیب هم از مزیت صرفه‌جویی در کدنویسی بهره‌گرفته‌ایم و هم سرعت اجرای برنامه را بالا برده‌ایم. اما متأسفانه حجم فایل اجرایی افزایش می‌یابد. چگونگی تأثیر استفاده از توابع **inline** در شکل ۹-۳ نشان داده شده است:



شکل ۹-۳ : توابع درون خط در مقایسه با توابع معمولی

در ادامه مثالی ارائه شده که چگونگی تعریف توابع **inline** را نشان می‌دهد.

مثال ۹-۱۲: برنامه‌ای که از یک تابع **inline** استفاده می‌کند.

```
1. //This program uses an inline function.
2. #include <iostream.h>
3. inline int max(int a, int b)
4. {
5.     return a > b ? a : b;
6. }
7. //**********************
8. int main()
9. {
10.    cout<<"Max of 10, 20 is : "<<max(10, 20)<<endl;
11.    cout<<"Max of 99, 88 is : "<<max(99, 88)<<endl;
12.   return 0;
13. }
```

توضیح مثال: شاید تنها نکتهٔ زیبا در این برنامه چگونگی عملکرد دستور **return** در خط پنجم است. چنانکه در این دستور می‌بینید یکی از مقادیر **a** یا **b** که توسط عملگر سه‌تایی برگشت داده می‌شود برگشت داده می‌شود.

۹-۲: توابع ریاضی کتابخانه ای

در زبان C++ در سر فایل **math.h** برخی توابع کتابخانه‌ای فراهم شده‌اند. متأسفانه تعداد این توابع بسیار زیاد است و امکان گنجاندن تمامی آنها در این کتاب وجود ندارد. لذا در پیوست چهار محتویات این سرفایل لیست شده است، که دانشجویان عزیز می‌توانند با مراجعه به این پیوست نام توابعی که در این قسمت توضیح داده نشده است را استخراج نموده و در اینترنت و یا مستندات **MSDN** نحوهٔ بکارگیری و کاربرد این توابع را مطالعه بکنند.

۱. تابع abs

این تابع برای محاسبهٔ قدرمطلق اعداد صحیح مورد استفاده قرار می‌گیرد. اگر آرگومان این تابع یک عدد منفی باشد نتیجهٔ حاصل از تابع یک عدد مثبت است و اگر آرگومان تابع، مثبت و یا صفر باشد نتیجه، یک عدد مثبت یا صفر خواهد بود. شکل کلی به‌کارگیری این تابع به‌صورت زیر است:

```
int abs(int num);
```

با توجه به اینکه با مفهوم تابع در بخش پیشین آشنا شده‌اید از این پس جهت نشان دادن شکل کلی به‌کارگیری توابع از اعلان آنها استفاده خواهیم کرد.

۲. تابع acos

این تابع برای محاسبهٔ **ArcCos** یک عدد مورد استفاده قرار می‌گیرد و الگوی آن به‌صورت زیر است:

```
double acos(double arg);
```

مقادیری راکه آرگومان این تابع می‌پذیرد در بازهٔ ۱- تا ۱ است و نتیجه به‌صورت رادیان و در بازهٔ صفر تا π است. در صورتی که پارامتر ورودی درمحدوده۱- تا ۱ نباشد مقدار **NULL** برگشت داده می‌شود.

۳. تابع asin

این تابع برای محاسبهٔ **ArcSin** یک عدد به‌کار رفته و الگوی آن به‌صورت زیر است:

```
double asin(double arg);
```

آرگومان این تابع در بازهٔ ۱- تا ۱ است و نتیجه حاصل به‌صورت رادیان و در باز هٔ **π/2-** و **π/2** است.

۴. تابع atan

این تابع برای محاسبهٔ **Arctan** یک عدد به‌کار می‌رود و الگوی آن به‌صورت زیر می‌باشد:

```
double atan(double arg);
```

مقادیری که توسط این تابع محاسبه می‌شوند به‌صورت رادیان و در بازهٔ **π/2-** و **π/2** است.

۵. تابع atan2

این تابع دو آرگومان را دریافت کرده، اولین آرگومان را بر دومین آرگومان تقسیم نموده و **Arctan** حاصل تقسیم را محاسبه می‌کند. الگوی این تابع به‌صورت زیر تعریف شده است:

```
double atan2(double y, double x);
```

همان‌طورکه می‌بینید این تابع دو آرگومان، به نام‌های **y** و **x** دارد که **Arctan (y/x)** را محاسبه می‌کند.

۶. تابع cos

این تابع برای‌محاسبهٔ کسینوس یک زاویه برحسب رادیان به‌کار می‌رود و الگوی آن به‌صورت زیراست:

```
double cos(double arg);
```

۷. تابع cosh

این تابع برای محاسبهٔ کسینوس هایپربولیک یک عدد به‌کار می‌رود و الگوی آن به‌صورت زیر است:

```
double cosh(double arg);
```

۸. تابع sin

این تابع برای محاسبه سینوس یک زاویه برحسب رادیان به‌کار می‌رود و دارای صورت کلی زیر است:

```
double sin(double arg);
```

۹. تابع sinh

این تابع برای محاسبهٔ سینوس هایپربولیک یک زاویه برحسب رادیان به‌کار رفته، و الگوی آن به‌صورت زیر است:

```
double sinh(double arg);
```

۱۰. تابع tan

این تابع برای محاسبهٔ تانژانت یک زاویه که برحسب رادیان می‌باشد به‌کار می‌رود و الگوی آن به‌صورت زیر است:

```
double tan(double arg);
```

۱۱. تابع tanh

این تابع برای محاسبهٔ تانژانت هایپربولیک یک زاویه برحسب رادیان به‌کار می‌رود و الگوی آن به‌صورت زیر است:

```
double tanh(double arg);
```

۱۲. تابع ceil

این تابع کوچکترین عدد صحیح بزرگتر یا مساوی با عددی را که به‌عنوان آرگومان آن می‌باشد، محاسبه می‌کند و الگوی آن به‌صورت زیر است:

```
double ceil(double num);
```

به عبارت دیگر این تابع سقف یک عدد را محاسبه می‌کند.

۱۳. تابع floor

این تابع بزرگترین مقدار صحیح کوچکتر یا مساوی با عددی را که به‌عنوان آرگومان آن می‌باشد، محاسبه می‌کند. الگوی آن نیز به‌صورت زیر است:

```
double floor(double num);
```

۱۴. تابع exp

این تابع برای محاسبهٔ توانی از e (پایه لگاریتم طبیعی) مورد استفاده قرار می‌گیرد و الگوی آن به‌صورت زیر است:

```
double exp(double arg);
```

۱۵. تابع log

این تابع لگاریتم طبیعی یک عدد مثبت را محاسبه می‌کند (پایه لگاریتم طبیعی عدد e = ۲ /۷۱۸۲۰۰۰ است) الگوی این تابع نیز به‌صورت زیر است:

```
double log(double num);
```

۱۶. تابع log10

این تابع لگاریتم مبنای ۱۰ اعداد مثبت را محاسبه می‌کند. الگوی این تابع به‌صورت زیر است:

```
double log10(double num);
```

۱۷. تابع pow

این تابع توان‌های یک مبنا (base) را محاسبه می‌کند و الگوی آن به‌صورت زیر می‌باشد:

```
double pow(double base, double exp);
```

نتیجهٔ حاصل از این تابع، عبارت $base^{exp}$ است. اگر مبنا صفر باشد و یا توان منفی یا صفر باشد این تابع عمل نخواهد کرد. همچنین اگر مبنا منفی باشد و توان (exp) از نوع صحیح نباشد نتیجه‌ای نخواهد داد.

۱۸. تابع sqrt

این تابع جذر یک عدد مثبت را محاسبه می‌کند و الگوی آن به‌صورت زیر است:

```
double sqrt(double num);
```

۱۹. تابع fabs

این تابع جهت محاسبهٔ قدرمطلق اعداد اعشاری به‌کار می‌رود و دارای الگوی کلی زیر است:

```
double fabs(double num);
```

۲۰. تابع fmod

این تابع دو آرگومان دارد که باقی‌ماندهٔ تقسیم اولین آرگومان را بر آرگومان دوم محاسبه می‌کند. به‌خاطر دارید که عملگر % تنها برروی اعداد صحیح عمل می‌کرد. الگوی این تابع به‌صورت کلی زیر است:

```
double fmod(double x, double y);
```

۲۱. تابع ldexp

این تابع دارای دو آرگومان به‌صورت زیر است و نتیجهٔ حاصل از این تابع عبارت $num * 2^{exp}$ است:

```
double ldexp(double num, int exp);
```

۲۲. تابع frexp

این تابع دارای دو آرگومان است. اولین آرگومان را به دو قسمت تبدیل می‌کند. که قسمت اول به‌صورت کسری و در بازهٔ [۰/۵ ، ۱) و قسمت دوم به‌صورت توانی از ۲ است. این تابع عدد **num** را به‌صورت $num = کسر * 2^{exp}$ در می‌آورد. قسمت کسری را به‌صورت یک مقدار **double** بازمی‌گرداند و قسمت توان را در متغیر **exp** قرار می‌دهد. این تابع دارای صورت کلی زیر است:

```
double frexp(double num, int* exp);
```

۲۳. تابع modf

این تابع عدد **num** را به دو قسمت اعشاری و صحیح تقسیم می‌کند. قسمت اعشاری را به‌صورت یک مقدار **double** برمی‌گرداند و قسمت صحیح را در متغیر **i** قرار می‌دهد. این تابع دارای الگوی کلی زیر است:

```
double modf(double num, int* i);
```

۲۴. تابع poly

این تابع جهت ارزیابی یک چند جمله‌ای به‌کار می رود. و دارای الگوی کلی زیر است:

```
double poly(double x, int n, double c[]);
```

در الگوی فوق، **n** معرف تعداد جملات، آرایهٔ **c** حاوی ضرایب چند جمله‌ای و **x** حاوی مقدار متغیر چند جمله‌ای است. به‌عنوان مثال اگر **n=3** باشد چند جمله‌ای که ارزیابی می‌شود به‌صورت زیر است:

c[3]*x + c[2]*x + c[1]*x + c[0]

۲۵. تابع hypot

این تابع با دریافت دو ضلع قائم یک مثلث قائم الزاویه، وتر آن را محاسبه می‌کند. الگوی کلی این تابع به‌صورت زیر است:

```
double hypot(double x, double y);
```

۹-۳ : سر بار گذاری توابع و قالب های تابع

سربارگذاری توابع (توابع هم نام)

در بسیاری از برنامه ها ممکن است بخواهیم اعمال مشابهی را بر روی داده های متفاوتی انجام دهیم. برای این منظور می‌توان توابع مورد نظر را به صورت هم نام در نظر گرفت و به اصطلاح توابع را سربارگذاری کرد. به‌عنوان مثال حالتی را در نظر بگیرید که می‌خواهیم مربع یک عدد را به خروجی ببریم، حال ممکن است پارامتر ورودی تابع از نوع **int** یا **float** باشد، برای این منظور دو تابع با نام مشابه **square** تعریف می‌کنیم که یکی مربع اعداد **int** و دیگری مربع اعداد **float** را محاسبه می‌کند. توابع سربارگذاری شده باید از نظر تعداد آرگومان‌های ورودی و یا نوع آرگومان‌های ورودی با یکدیگر متفاوت باشند. توجه داشته باشید که تفاوت در نوع برگشتی تابع نمی‌تواند مبنای سربارگذاری توابع باشد. به‌عنوان مثال دو تابع زیر به درستی سربارگذاری شده‌اند.

```
int square(int);
float square(float);
```

اما دو تابع زیر به شیوهٔ نادرستی سربارگذاری شده‌اند که موجب صدور پیغام خطایی از طرف کامپایلر در زمان ترجمهٔ می‌شود:

```
int square(int);
float square(int);
```

وقتی توابع سربارگذاری شده فراخوانی می‌شوند، کامپایلر C++ با توجه به تعداد پارامترهای ورودی یا نوع و ترتیب پارامترهای ورودی، تابع مناسب با پارامترهای ارسالی به تابع را تشخیص داده و اجرا می‌کند. باتوجه به آنچه گفته شد می‌توان چنین نتیجه‌گیری کرد که وقتی توابعی کارهای مشابهی را با انواع مختلف داده‌ها انجام می‌دهند، این روش تعریف توابع می‌تواند مفید واقع شود.

مثال ۹-۱۳ : برنامه‌ای که نحوهٔ سربارگذاری توابع را نشان می‌دهد.

```
1.  //This program is an example for function overloading.
2.  double max( double d1, double d2 )
3.  {
4.     return ( d1 > d2 ) ? d1 : d2;
5.  }
6.  int max( int i1, int i2 )
7.  {
8.     return ( i1 > i2 ) ? i1 : i2;
9.  }
10. void main()
11. {
12.    int    i = max( 12, 8 );
13.    double d = max( 32.9, 17.4 );
14. }
```

توضیح مثال : خط ۱۲ موجب فراخوانی دومین تابع **max** و خط ۱۳ موجب فراخوانی اولین تابع **max** می‌شود.

آرگومان های دارای مقدار پیش‌فرض

برای هر آرگومانی که در عنوان تابع تعریف می شود، هنگام فراخوانی باید مقداری به تابع ارسال شود به این نوع آرگومان‌ها که تاکنون با آنها سروکار داشتید آرگومان‌های الزام آور یا **mandatory argument** گفته می‌شود. اما اگر برای این آرگومان مقدار پیش‌فرضی را در نظر بگیریم، در صورتی که هنگام فراخوانی تابع، آرگومان آن را ارسال نکنیم، کامپایلر از مقدار پیش‌فرض استفاده می‌کند، به این نوع آرگومان‌ها نیز **default argument** گفته می‌شود. به‌عنوان مثال فرض کنید الگوی تابع **Func** را به‌صورت زیر اعلان کنیم:

```
long Func(int x=50);
```

این الگو به کامپایلر می‌گوید که تابع **Func** یک مقدار **int** را به‌عنوان پارامتر می‌پذیرد و یک مقدار از نوع **long** را برمی‌گرداند. اگر هنگام فراخوانی این تابع، آرگومانی به آن ارسال نشود، از مقدار ۵۰ استفاده می‌شود. چون آرگومان‌ها در الگوی تابع می‌توانند فاقد نام باشند، الگوی این تابع را به‌صورت زیر نیز می‌توان نوشت:

```
long Func(int =50);
```

تعریف تابع درحالتی‌که الگوی آن دارای آرگومان پیش‌فرض باشد، با حالت عادی هیچ تفاوتی ندارد. بنابراین، تابع **Func** را می‌توان به‌صورت زیر تعریف کرد:

```
int Func(int x)
{
  .
  .
  .
}
```

اگر تابعی دارای چند آرگومان باشد، هر آرگومان می‌تواند دارای مقدار پیش‌فرض باشد، ولی یک محدودیت وجود دارد به‌طوری که، آرگومان‌های پیش‌فرض باید سمت راست‌ترین آرگومان‌ها باشند. این محدودیت به این دلیل است که وقتی کنترل اجرای برنامه به دستور فراخوانندهٔ تابع می‌رسد، ابتدا مقادیری را که به‌عنوان پارامتر ورودی ذکر شده‌اند به ترتیب از چپ به راست در متغیرهایی که در تعریف تابع به‌عنوان آرگومان ذکر شده‌اند کپی می کند و اگر آرگومانی بدون پارامتر ورودی باقی ماند، از مقدار پیش‌فرض برای آن استفاده می‌کند. حال تابعی را در نظر بگیرید که دارای دو آرگومان است و اولین آرگومان آن مطابق الگوی زیر دارای مقدار پیش‌فرض می باشد:

```
int Func(int =64,int );
```

حال اگر بخواهیم این تابع را با ارسال تنها یک پارامتر فراخوانی کنیم به‌طوری که آرگومان اول از مقدار پیش‌فرض و آرگومان دوم از پارامتر ورودی استفاده کنند با مشکل مواجه خواهیم شد. چرا که پارامتر ورودی در هنگام فراخوانی تابع برای آرگومان اول منظور شده و آرگومان دوم بدون مقدار ورودی می‌ماند. لذا اگر چنین اعلانی را برای یک تابع در برنامهٔ خود بنویسید و کد برنامه را کامپایل کنید با خطای زمان کامپایل به‌صورت زیر برخورد خواهید کرد:

error C2548: 'Func' : missing default parameter for parameter 2

دوباره متذکر می‌شویم که مقادیر پیش‌فرض تنها باید در اعلان تابع ظاهر شوند و نه در هیچ کجای دیگر.

مثال ۹-۱۴: برنامه‌ای که چگونگی کاربرد آرگومان‌های فرضی را نشان می‌دهد. این برنامه حجم یک مکعب مستطیل را توسط تابعی محاسبه می‌کند.

```
1. //This program uses default arguments
2. #include <iostream>
3. using namespace std;
4. int BoxVolume(int length, int width=10, int height=1);
5. int main()
6. {
7.   int area, length = 100, width = 50, height = 2;
8.     area = BoxVolume(length, width, height);
9.       cout<<"First time area equals: "<<area<<endl;
10.    area = BoxVolume(length, width);
11.      cout<<"Second time area equals: "<<area<<endl;
12.    area = BoxVolume(length);
13.       cout<<"Third time area equals: "<<area<<endl;
14. return 0;
15. }
16. int BoxVolume(int length, int width, int height)
17. {
18.    return (length * width * height);
19. }
```

توضیح مثال: فراخوانی اول با سه آرگومان انجام شد که مقادیر ۱۰۰، ۵۰ و ۲ ارسال می‌شوند. در فراخوانی دوم دو آرگومان با مقادیر ۱۰۰ و ۵۰ ارسال می‌شوند و به‌جای آرگومان سوم از مقدار پیش‌فرض ۱ استفاده می‌شود. در فراخوانی سوم، یک آرگومان با مقدار ۱۰۰ ارسال می‌شود و به‌جای دو آرگومان دیگر از مقادیر فرضی ۱۰ و ۱ استفاده می‌شود. خروجی حاصل از اجرای این برنامه به‌صورت زیر خواهد بود:

```
First time area equals: 10000
Second time area equals: 5000
Third time area equals: 1000
```

توجه: اگر اعلان تابعی را که دارای آرگومان‌های پیش‌فرض است بخواهید حذف کنید در آن صورت می‌توانید با انتقال تابع به قبل از مکان اولین فراخوانی خود (چنانکه دیدید عموماً قبل از تابع اصلی به شرط آنکه در دیگر توابع فرعی فراخوانی نشده باشد) در آن صورت می‌توانید مقادیر پیش‌فرض را در تعریف تابع ذکر کنید.

تعریف توابع به صورت قالب[1]

معمولاً از توابع سربارگذاری شده برای انجام عملیات مشابهی استفاده می‌شود که روی انواع مختلفی از داده‌ها با منطقی متفاوت در برنامه به کار می‌روند. اگر منطق و عملیات برنامه برای هر نوع داده‌ای یکسان باشد این‌کار می‌تواند با استفاده

۱. در اکثر مثال‌های این مبحث از مرجع استفاده شده است. لذا خوانندگان عزیز می‌توانند مطالعهٔ این بخش را تا هنگام مطالعه مبحث مرجع در فصل یازدهم به تعویق بیندازند، اما ما به جهت کامل شدن بحث توابع، این مطالب را در این قسمت آورده ایم.

از قالب توابع، بسیار فشرده‌تر و آسان‌تر انجام‌شود. به عبارت دیگر، در توابع کلی، یک عمل مشابه برروی انواع مختلفی از داده انجام می‌شود. برنامه‌نویس تنها یک قالب از تابع را می‌نویسد و کامپایلر C++ در هنگام ترجمهٔ برنامه به زبان ماشین، برحسب انواع پارامترهایی که به‌عنوان ورودی در فراخوانی تابع به‌کار رفته است، تعریف جدیدی از تابع متناسب با پارامترهای ورودی ایجاد می‌کند و با توجه به این تعریف برنامه به زبان ماشین ترجمه می‌شود. به‌عنوان مثال، می‌توان قالب تابعی برای تابع مرتب‌سازی آرایه نوشت و کامپایلر C++ به‌طور خودکار توابع قالب دیگری را تولید کند که آرایه‌ای از نوع صحیح، آرایه‌ای از نوع اعشاری و غیره را مرتب کند. در زبان C، این‌کار با استفاده از ماکروها انجام می‌شود، اما ماکروها اثرات جانبی زیادی را ایجاد می‌کنند و کنترل نوع را نیز انجام نمی‌دهند. اما قالب‌های تابع، کنترل نوع را به شکل دقیقی انجام می‌دهند.

با استفاده از قالب‌ها می‌توان توابع کلی[1] و چنانکه بعداً خواهید دید کلاس‌های کلی را ایجاد کرد. در یک تابع کلی، نوعِ داده‌ای که این توابع برروی آن عمل می‌کنند به‌صورت پارامتر مشخص می‌شود. بنابراین، می‌توانید با یک تابع کلی برای چند نوع مختلف از داده‌ها کار کنید، بدون اینکه نیاز به دستورالعمل‌های خاص آن انواع باشد. در این فصل، توابع کلی را مورد بررسی قرار می‌دهیم. و در جلد دوم پس از مطرح شدن بحث کلاس‌ها، کلاس‌های کلی را مورد مطالعه قرار خواهیم داد.

توابع کلی

تابع کلی، مجموعه‌ای کلی از اعمال را تعریف می‌کند که برروی انواع مختلفی از داده‌ها امکان‌پذیر است. نوعِ داده‌ای که تابع باید برروی آن عمل کند، به‌عنوان آرگومان به آن ارسال می‌شود. با تابع کلی، یک رویه کلی را می‌توان برروی انواع مختلفی از داده‌ها انجام داد. با ایجاد یک تابع کلی، می‌توان ماهیت الگوریتم را مستقل از هر نوع داده‌ای تعریف کرد. به این ترتیب، کامپایلر کد مناسبی را برای داده‌ای که در زمان اجرای تابع مشخص می‌شود تولید می‌کند. به‌طور کلی، وقتی یک تابع کلی ایجاد می‌کنید، مثل این است که تابعی ایجاد می‌کنید که خودش را قادر است مجدداً تعریف کند.

توابع کلی با کلمه کلیدی **template** ایجاد می‌شوند. این کلمه کلیدی، قالبی را ایجاد می‌کند که عمل تابع را مشخص می‌کند و جزئیات امر را به کامپایلر واگذار می‌نماید. شکل کلی تعریف قالب تابع به‌صورت زیر است:

```
template <class  T>
return value  function name ( arguments list )
{
   body function
}
```

در اینجا، **T** نامی است که مشخص‌می‌کند چه نوع داده‌ای توسط تابع مورد استفاده قرارمی‌گیرد، و برای نامگذاری آن می‌توان از قواعد نامگذاری متغیرها استفاده کرد. این نوع در هنگام اجرای برنامه مشخص می‌شود. این نام ممکن است در تعریف تابع به کار گرفته شود. لذا هنگامی که کامپایلر نسخه خاصی از تابع را ایجاد نمود، نوع واقعی را به‌جای این نام قرار می‌دهد. همچنین ***return value*** مقدار بازگشتی تابع، ***function name*** نام تابع و ***argument list*** لیست پارامترهای تابع و ***body function*** بدنهٔ تابع می‌باشد.

---

1. total function

مثال ۹-۱۵ : برنامه‌ای که تابع قدرمطلق را با استفاده از قالب‌ها برای انواع داده پیاده‌سازی می‌کند.

```
1.  //This program uses template for absolute value function
2.  #include <iostream>
3.  using namespace std;
4.  //********************
5.  template <class T>
6.  T abs(T n)
7.  {
8.      return (n < 0) ? (-n) : (n) ;
9.  }
10. //********************
11. int main()
12. {
13.    int int1=5;
14.     int int2=-6;
15.      long lon1=70000L;
16.       long lon2=-80000L;
17.        double dub1=9.95;
18.         double dub2=-10.6;
19.    //calls instantiate functions
20.    cout<<"\nabs("<<int1<<")="<<abs(int1); //abs(int)
21.     cout<<"\nabs("<<int2<<")="<<abs(int2); //abs(int)
22.      cout<<"\nabs("<<lon1<<")="<<abs(lon1); //abs(long)
23.       cout<<"\nabs("<<lon2<<")="<<abs(lon2); //abs(long)
24.        cout<<"\nabs("<<dub1<<")="<<abs(dub1); //abs(double)
25.         cout<<"\nabs("<<dub2<<")="<<abs(dub2); //abs(double)
26.          cout<<endl;
27.    return 0;
28. }
```

توضیح مثال : چون شیوه تعیین مقدار قدرمطلق برای انواع مقادیر به نوع آنها بستگی ندارد، به راحتی می‌توان با یک تابع کلی تابع قدرمطلق را پیاده‌سازی کرد (به تحلیل مثال توجه کنید). خروجی این برنامه به‌صورت زیر خواهد بود:

```
abs(5)=5
abs(-6)=6
abs(70000)=70000
abs(-80000)=80000
abs(9.95)=9.95
abs(-10.6)=10.6
```

تحلیل مثال : سطر زیر را در نظر بگیرید:

```
template <class T>  T  abs(T  n)
```

این سطر دو چیز را به کامپایلر می‌گوید:

۱. قالبی در حال ایجاد شدن است.

۲. یک تعریف کلی آغاز شده است.

در اینجا، **T** یک نوع فرضی(کلی) است. پس از بخش **template**، تابع **abs** اعلان شده که از **T** به‌عنوان نوع داده‌ای که باید دریافت شود استفاده شده است. در تابع **main**، تابع **abs** با استفاده از سه نوع دادهٔ مختلف فراخوانی

شد: **int**، **long** و **double**. چون تابع **abs** یک تابع کلی است، کامپایلر سه نسخه از این تابع را ایجاد می‌کند: یکی از آنها قدرمطلق یک مقدار **int** را برمی‌گرداند، دومی، قدرمطلق مقادیر **long** و سومی قدرمطلق مقادیر **double** را محاسبه می‌نماید.

کامپایلر چه کاری انجام می‌دهد؟

یک سؤال مهم که همواره در ذهن افراد تیزبین نقش می‌بندد این است که وقتی کامپایلر کلمه کلیدی **template** و تعریف تابعی را که بعد از آن است می‌بیند چه کاری انجام می‌دهد؟ هیچ کار، واقعاً هیچ کاری انجام نمی‌دهد. قالب تابع باعث نمی‌شود کامپایلر کدی تولید کند. از آنجا که قالب تابع هنوز چیزی در مورد نوع داده‌ای که تابع با آن کار می‌کند بیان نمی‌کند، از این رو هیچ کدی تولید نمی‌شود. فقط به‌خاطر می‌سپارد که برای کاربرد احتمالی بعدی، این تابع دارای یک قالب است.

تولید کد تنها وقتی صورت می‌گیرد که تابع واقعاً توسط دستوری در داخل برنامه احضار شود. در برنامهٔ مثال ۹-۱۵ تولید کد در عبارت‌های نظیر دستور زیر انجام می‌شود:

```
cout<<"\nabs("<<int1<<")="<<abs(int1);
```

هنگامی که کامپایلر چنین احضار تابعی را می‌بیند در می‌یابد که نوع مورد استفاده **int** است. زیرا این نوع داده همان نوع پارامتر **int1** است. از این رو تعریف خاصی از تابع **abs** را برای نوع **int** تولید می‌کند و در هر جایی که با نام **T** در قالب تابع روبرو می‌شود نوع **int** را جایگزین می‌کند. این عمل *نمونه‌سازی* یا *ارائه یک نمونه* از *قالب تابع* نام دارد و هر نسخهٔ نمونه‌سازی شده از *قالب تابع* یک *تابع قالب* نامیده می‌شود.

علاوه بر این، کامپایلر فراخوانی یک تابع نمونه‌سازی شده را تولید می‌کند و آن را به کدی اضافه می‌کند که در آن **abs(int1)** قرار دارد. به همین ترتیب عبارت **abs(lon1)** باعث می‌شود که کامپایلر نسخه‌ای از **abs** را تولید کند که روی نوع **long** و نیز احضار این نسخه از تابع عمل می‌کند. حال آن که احضار **abs(dub1)** تابعی را تولید می‌کند که روی نوع **double** عمل می‌کند. البته کامپایلر آنقدر هوشمند است که تنها یک نسخه از **abs** را برای هر نوع داده‌ای تولید کند. به این ترتیب حتی اگر دو احضار برای نسخهٔ **int** این تابع وجود داشته باشد کد مربوط به این نسخه تنها یکبار در کد اجرایی ظاهر می‌شود.

توجه دارید که مقدار حافظهٔ اصلی (**RAM**) مورد استفاده در برنامه، بستگی به این دارد که آیا از روش قالب استفاده می‌کنید یا سه تابع مستقل می‌نویسید. آنچه که باعث صرفه‌جویی شده است آن است که از تایپ سه تابع مستقل در فایل اصلی خودداری کنیم. این کار لیست برنامه را کوتاه‌تر و درک آن را ساده‌تر می‌کند، اما در حجم فایل اجرایی تأثیر چندانی ندارد. همچنین با استفاده از قالب توابع اگر بخواهیم روش کار تابع را تغییر دهیم، تنها لازم است برنامه را در یک مکان به‌جای سه مکان تغییر دهیم.

کامپایلر براساس نوع داده‌های به‌کار گرفته شده در پارامترهای احضار تابع تصمیم می‌گیرد چگونه تابع را کامپایل کند. نوع برگشتی تابع در این تصمیم‌گیری دخالتی ندارد. این عمل شبیه روشی است که کامپایلر تصمیم می‌گیرد کدام یک از چند تابع سربارگذاری شده را احضار کند.

قالب یک تابع واقعاً یک تابع نیست. زیرا حقیقتاً باعث نمی‌شود دستورهای تابع در حافظه قرار گیرند.(به عبارت دیگر در فایل اجرایی برنامه چیزی به‌عنوان قالب تابع وجود ندارد، بلکه کدهای توابع قالب ساخته شده هستند که به فایل اجرایی اضافه می‌گردند. یک تابع یک طرح[1] یا نقشه یا یک نسخهٔ اصلی برای ساختن توابع قالب است. استفاده از قالب‌های تابع یک گام بزرگ در جهت برنامه‌نویسی شئ‌گرا است که دلیل این مدعا را در جلد دوم و در مباحث شئ‌گرا به دفعات خواهید دید.

تابعی با دو نوع کلی

در دستور **template** می‌توان بیش از یک نوع داده را تعریف کرد. در این صورت باید آنها را با کاما از هم جدا نمود. به مثال زیر توجه کنید.

مثال ۹-۱۶ : برنامه‌ای که کاربرد تابعی با دو نوع کلی را نشان می‌دهد.

```
1.  //Uses two template types for a function.
2.  #include <iostream.h>
3.  //***************************
4.  template <class t1, class t2>
5.  void Func(t1 x, t2 y)
6.  {
7.     cout<<"x = "<<x<<" , y = "<<y<<endl;
8.  }
9.  //***************************
10. int main()
11. {
12.    Func(10, "I like C++");
13.    Func(89.25, 19L);
14.  return 0;
15. }
```

توضیح مثال : در این مثال، **t1** و **t2** دو نوع کلی هستند که در فراخوانی اول تابع **Func**، به ترتیب، انواع **int** و ***char** به‌جای آنها قرار می‌گیرند و در فراخوانی دوم، به ترتیب، انواع **float** و **long** به‌جای آنها قرار خواهند گرفت. خروجی حاصل از این برنامه به‌صورت زیر خواهد بود:

```
x = 10 , y = I like C++
x = 89.25 , y = 19
```

1. pattern

توابع قالب و انواع استاندارد در C++

در قالب تابع، می‌توان انواع استاندارد موجود در C++ را با انواع کلی تعریف کرد. به مثال زیر توجه کنید.

۹-۱۷: برنامه‌ای که ترکیب انواع استاندارد و انواع کلی را در قالب تابع نشان می‌دهد.

```
1.  //uses template for function that finds number in array
2.  #include <iostream>
3.  using namespace std;
4.  //**************************
5.  template <class ar_type>
6.  int find(ar_type  array[], ar_type  value, int size)
7.  {
8.     for(int i=0; i<size; i++)
9.        if(array[i]==value)
10.           return (i+1);
11.    return 0;
12. }
13. //**************************
14. int main()
15. {
16.    char chrarr[]={'H','e','l','l','o'};  //array
17.    char ch='o';                          //value to find
18.    int  intarr[]={1,3,5,9,11,13};
19.    int  in=6;
20.    long lonarr[]={1L,3L,5L,9L,11L,13L,16L};
21.    long lo=11L;
22.    double dubarr[]={1.0,3.2,5.1,9.6,12.4,24.4};
23.    double du=4.0;
24. cout<<"\n o in char Array : index="<<find(chrarr, ch, 5);
25. cout<<"\n 6 in int  Array : index="<<find(intarr, in, 6);
26. cout<<"\n11 in long Array : index="<<find(lonarr, lo, 7);
27. cout<<"\n 4 in double Array : index="<<find(dubarr,du,6);
28. cout<<endl;
29.   return 0;
30. }
```

توضیح مثال: در این برنامه، تابع **find** دومین آرگومان را در آرایهٔ **array** جستجو می‌کند. چون آرگومان اول یک نوع کلی است، تابع **find** می‌تواند برای جستجو در هر نوع آرایه‌ای به‌کار رود. آرگومان **size**، یک متغیر از نوع استاندارد یعنی **int** است و مقدار آن به تابع ارسال می‌شود. خروجی این برنامه به‌صورت زیر است.

```
 o in char   Array : index=5
 6 in int    Array : index=0
11 in long   Array : index=2
 4 in double Array : index=0
```

تعریف مجدّد تابع کلی

گرچه قالب تابع در صورت نیاز، خودش را دوباره تعریف می‌کند، ولی به‌طور صریح نیز قابل تعریف مجدد است. در این صورت، کامپایلر تابع کلی را نسبت به آن تابع مجدّد تعریف شده، نادیده می‌گیرد، و از آن نسخهٔ خاص استفاده می‌کند. به مثال زیر توجه کنید.

مثال ۹-۱۸ : برنامه‌ای که داده‌هایی با انواع مختلف را جابه‌جا می‌کند (به تحلیل مثال توجه کنید).

```
1. //This program overloads a total function.
2. #include <iostream>
3. using namespace std;
4. template <class T> void swapArgs(T& a, T& b)
5. {
6.      T  temp;
7.      temp = a;
8.      a = b;
9.      b = temp;
10.     cout<<"Inside template swapArgs\n";
11. }
12. //this overrides the generic version of swapArgs()
13. void swapArgs(int& a, int& b)
14. {
15.    int temp;
16.    temp = a;
17.    a = b;
18.    b = temp;
19.    cout<<"Inside swapArgs specialization\n";
20. }
21. //-----------------------------------------------------
22. int main()
23. {
24.    int i = 10, j = 20;
25.    double x = 10.1,  y = 23.3;
26.    char a = 'x', b = 'z';
27.    cout<<"Original i, j : "<<i<<" , "<<j<<"\n";
28.     swapArgs(i, j);//calls explicitly overloaded swapArgs
29.      cout<<"Swapped i, j : "<<i<<" , "<<j<<"\n";
30.    cout<<"Original x, y : "<<x<<" , "<<y<<"\n";
31.     swapArgs(x, y);//calls generic swapArgs
32.      cout<<"Swapped x, y : "<<x<<" , "<<y<<"\n";
33.    cout<<"Original a, b : "<<a<<" , "<<b<<"\n";
34.     swapArgs(a, b);//calls generic swapArgs
35.      cout<<"Swapped a, b : " << a << " , " << b << "\n";
36.   return 0;
37. }
```

توضیح مثال : خروجی حاصل از این برنامه به‌صورت زیر است:

```
Original i, j : 10 , 20
 Inside swapArgs specialization
Swapped i, j : 20 , 10
Original x, y : 10.1 , 23.3
Inside template swapArgs
Swapped x, y : 23.3 , 10.1
Original a, b : x , z
Inside template swapArgs
Swapped a, b : z , x
```

تحلیل مثال : وقتی **swapArgs(i, j)** فراخوانی می‌شود، نسخه‌ای از این تابع که صریحاً در برنامه مجدداً تعریف شده است، فراخوانی می‌گردد، بنابراین، کامپایلر این نسخه از تابع کلی **swapArgs** را تولید نمی‌کند (زیرا تابع کلی مجدداً تعریف گردیده است). تابع **swapArgs** تعریف شده در خطوط ۱۳ الی ۲۰ را به‌صورت زیر نیز می‌توان تعریف کرد:

```
template < > void swapArgs <int>(int& a, int& b)
{
    int  temp;
    temp = a;
    a = b;
    b = temp;
    cout<<"Inside swapArgs int specialization.\n";
}
```

در این روش، قالب به‌صورت **template < >** ذکر می‌شود. نوعی که نسخه خاصی از تابع باید برروی آن عمل کند، در جلوی نام تابع و در بین علائم < > قرار می‌گیرد (مثل **<int>**).

چرا ماکروها نه؟

برنامه‌نویس‌های قدیمی زبان **C** ممکن است تعجب کنند چرا ما برای ایجاد نسخه‌های مختلف یک تابع برای انواع داده‌ای مختلف از ماکرو استفاده نمی‌کنیم. برای مثال تابع **abs** را می‌توانیم به‌صورت زیر تعریف کنیم:

```
#define abs(n) ( (n<0) ? (-n) : (n) )
```

این کار شبیه اثر قالب کلاس در مثال ۹-۱۵ است زیرا جایگزینی متن ساده‌ای را انجام می‌دهد و به این ترتیب می‌تواند با هر نوع داده‌ای کار کند. با وجود این همانگونه که قبلاً متذکر شدیم، در **C++** از ماکروها زیاد استفاده نمی‌شود زیرا هنگام کار با ماکروها با مشکلات متعددی روبرو می‌شویم. ماکروها هیچ کنترلِ نوعی در برنامه ندارد. در ماکروها ممکن است چند آرگومان وجود داشته باشد که باید از یک نوع باشند اما کامپایلر کنترل نمی‌کند که آیا آنها همنوع هستند یا خیر. علاوه بر این، مقدار بازگشتی توسط ماکرو اصلا مشخص نمی‌شود از این‌رو کامپایلر نمی‌تواند بگوید که آیا شما آن را در یک متغیر ناسازگار و غیر همنوعش قرار داده‌اید یا خیر. در هر حالت، ماکروها به تابع‌هایی محدود می‌شوند که می‌توان آنهارا به‌صورت دستورهای یک خطی بیان کرد. در رابطه با ماکرو مشکلات حاد دیگری هم وجود دارد که در مجموع به شما توصیه می‌کنیم از ماکروها استفاده نکنید، اما در انتهای همین فصل یک کاربرد مهم ماکروها را در فراخوانی با نام خواهید دید.

تعریف مجدّد قالب تابع

علاوه بر اینکه تابع کلی را می‌توان مجدداً تعریف کرد، خود قالب تابع را نیز می‌توان مجدّداً تعریف نمود. برای این کار، نسخه دیگری از قالب تابع ایجاد می‌کنیم که لیست آرگومان‌های آن با لیست آرگومان‌های قالب قبلی متفاوت باشد.

مثال ۹-۱۹ : برنامه‌ای که کاربرد تعریف مجدد قالب تابع را نشان می‌دهد در این برنامه، قالب تابع **Func** مجدّداً تعریف شده است تا بتواند یک یا دو پارامتر را بپذیرد.

```
1. //This program overloads a template.
2. #include <iostream>
3. using namespace std;
4. //-------------------------------------------------------
5. //first version of Func() template
6. template <class T>
7. void Func(T a)
8. {
9.    cout<<"Inside Func(T a), a is "<<a<<endl;
10. }
11. //***************
12. //second version of Func() template
13. template <class X, class Y>
14. void Func(X a, Y b)
15. {
16.     cout<< "Inside Func(X a, Y b) , a is "<<a
17.         <<", b is "<<b<<endl;
18. }
19. //-------------------------------------------------------
20. int main()
21. {
22.    Func(100);        //calls Func(T a)
23.    Func(100, 200);    //calls Func(X a, Y b)
24.   return 0;
25. }
```

توضیح مثال : خروجی حاصل از اجرای این برنامه به‌صورت زیر است:

```
Inside Func(T a) , a is 100
Inside Func(X a , Y b) , a is 100 , b is 200
```

۹-۴ : کلاس های حافظه و حوزهٔ متغیرها

متغیرهای محلی و سراسری

در برنامه‌هایی که تا اینجا دیدید، متغیرهای مورد نیاز هر تابع، در داخل آن تابع تعریف شدند. متغیرهایی که در داخل یک تابع تعریف می‌شوند، *متغیرهای محلی*[1] نامیده می‌شوند. متغیرهای محلی فقط در همان تابعی که تعریف می‌شوند قابل دسترس هستند. یعنی حوز[2] اینگونه متغیرها فقط در همان تابعی است که در آن تعریف می‌شوند. به‌عنوان مثال هر متغیری که در تابع **main** تعریف شود فقط در همین تابع قابل استفاده است، و هر متغیری که در یک تابع فرعی مثل **Func** تعریف شود نیز فقط در همین تابع **Func** قابل دسترس و استفاده است. و چنانکه پیشتر دیدید با توجه به همین تفاوت در حوزهٔ متغیرها است که می‌توان در دو تابع مستقل دو متغیر با نام‌های مشابه مثلاً با نام **x** تعریف کرد. و لذا متغیر **x** تعریف شده در یک تابع با متغیر **x** تعریف شده در تابع دیگر به‌کلی متفاوت است. چنانکه در مثال ۹-۴ به وضوح دیدید که متغیرهای تعریف شده در تابع **main** با متغیرهای تعریف شده در تابع **f 1** به کلی متفاوت بودند.

اما در **C++** این امکان وجود دارد که متغیرها را در خارج از توابع تعریف کرد. در این صورت به اینگونه متغیرها، متغیرهای سراسری[3] گفته می‌شود. متغیرهای سراسری در دسترس تمامی توابعی که تعریف آنها بعد از تعریف متغیر سراسری قرار دارد، هستند و لذا از یک متغیر سراسری، توابع مختلفی می‌توانند استفاده کنند. به‌عنوان مثال اگر متغیری در خارج از توابع و بالای تابع **main** تعریف شود، در تمام توابع موجود در برنامه که پس از تعریف این متغیر، تعریف شده باشند قابل دسترس است. یکی از کاربردهای متغیرهای سراسری می‌تواند برگشت مقادیر از یک تابع به تابع فراخواننده باشد. اما با توجه به آنکه از طریق فراخوانی با آدرس می‌توان مقادیر مختلف را به مجری یک تابع بازگرداند لذا توصیه می‌شود از متغیرهای سراسری برای این منظور استفاده نشود. چرا که متغیرهای سراسری درک و نگهداری برنامه را مشکل می‌کنند. از طرفی چنانکه بعداً خواهید دید استفاده از متغیرهای سراسری با اصول طراحی شئ‌گرا و مقولهٔ بسته‌بندی[4] و حفاظت داده‌ها مغایر است. بنابراین توصیهٔ ما به شما این است که حتی‌الامکان از متغیرهای سراسری در برنامه‌های خود استفاده نکنید، مگر آنکه در برنامه‌ای اغلب توابع بخواهند از یک متغیر به‌طور مشترک استفاده کنند. یکی از مهمترین تفاوت‌های متغیرهای محلی و سراسری در این است که متغیرهای محلی تا زمانی‌که مقدار اولیه نگرفته‌اند مقدار آنها معتبر نیست ولی متغیرهای سراسری دارای مقدار اولیهٔ صفر هستند. اگر در تابعی متغیر محلی هم‌نام با یک متغیر سراسری تعریف شود، در آن تابع، آن متغیر سراسری قابل دسترس نیست، مگر با استفاده از عملگر حوزهٔ تفکیک[5]، یعنی عملگر::.

اما ثوابت را نیز می‌توان به‌صورت سراسری تعریف کرد، و به جهت این که امکان تغییر در ثوابت سراسری توسط توابع وجود ندارد، تأثیرات منفی چندانی برروی برنامه نمی‌تواند بگذارد و لذا به کار بردن ثوابت سراسری بسیار معمول تر از متغیرهای سراسری است.

---

1. local variables 2. scope 3. global variables 4. encapsulation 5. scope resolution

مثال ۹-۲۰: برنامه‌ای که چگونگی استفاده از متغیرهای محلی و سراسری را نشان می‌دهد.

```
1.  //This program uses global and local variables.
2.  #include <iostream>
3.  using namespace std;
4.  int x; //define global variable
5.  void func1(void);
6.  void func2(void);
7.  int main()
8.  {
9.    x = 100; //access global variable
10.   func1();
11.   func2();
12.   cout<<"\n In main x is : "<<x; //access global variable
13.    cout<<endl;
14.  return 0;
15. }
16. //*********************
17. void func1(void)
18. {
19.   cout<<"\n In func1 x is: "<<x; //access global variable
20. }
21. //*********************
22. void func2(void)
23. {
24.    int x ; //define local variable
25.     cout << "\n In func2 x changes:";
26.      for(x = 1; x < 6; x ++)
27.           cout<<"  x="<<x;//access local variable
28. // access global variable by scope resolution operator
29.     cout<<"\n Global x in func2 is : "<< ::x ;
30. }
31.
```

توضیح مثال: در خط۴ این برنامه متغیر x به‌عنوان متغیر سراسری تعریف شده است. این متغیر در توابع **main** و **func1** قابل دسترس است. اما چون در تابع **func2** متغیری محلی به نام x تعریف شده است، متغیر سراسری در این تابع قابل استفاده نیست. با این وجود چنانکه در خط ۲۹ این کد می‌بینید به واسطهٔ عملگر :: به متغیر سراسری تعریف شده در خط ۴ دسترسی پیدا کرده‌ایم. خروجی حاصل از اجرای این برنامه به‌صورت زیر است:

```
In func1 x is: 100
In func2 x changes:   x=1  x=2  x=3  x=4  x=5
Global x in func2 is : 100
In main x is : 100
```

کلاس‌های حافظه

اکنون می‌خواهیم ویژگی‌هایی از C++ را که به روابط بین متغیرها و توابع مربوط می‌شود را یعنی کلاس‌های حافظه را مورد بررسی قرار دهیم. مقصود از کلاس حافظهٔ[1] هر متغیر، دو ویژگی حوزه[2] و طول عمر[3] آن متغییر است. منظور از حوزهٔ متغیر این است که یک متغیر در چه جاهایی از برنامه قابل دستیابی است. به‌عبارت دیگر، کلاس حافظهٔ متغیری مثل x ، مشخص می‌کند که این متغیر در چه جاهایی از برنامه قابل استفاده است و در چه جاهایی نمی‌توان به آن دسترسی داشت و از آن استفاده کرد. منظور از طول عمر متغیر، مدت زمانی است که متغیر در حافظه وجود دارد. به عبارت دیگر طول عمر متغیر به ما می‌گوید، یک متغیر چه زمانی به‌وجود می‌آید و چه زمانی از بین می‌رود. بر این اساس چهار نوع کلاس حافظه در C++ قابل تعریف است که عبارت‌اند از:

۱. کلاس حافظهٔ اتوماتیک[4]
۲. کلاس حافظهٔ ثبات[5]
۳. کلاس حافظهٔ استاتیک[6]
۴. کلاس حافظهٔ خارجی[7]

برای تعیین کلاس حافظه برای متغیرها، به‌صورت زیر عمل می‌شود:

کلاس حافظه نوع متغیر نام متغیر ;

به‌عنوان مثال، دستورات زیر، کلاس حافظهٔ متغیر x را static و کلاس حافظهٔ y را register اعلام می‌کنند.

```
static int x;
register char y;
```

حال به بررسی انواع کلاس حافظه می‌پردازیم.

کلاس حافظهٔ اتوماتیک

تقریباً تمامی متغیرهایی که تا اینجا در توابع مختلف تعریف کردیم دارای کلاس حافظهٔ اتواماتیک بودند. به عبارت دیگر کلیهٔ متغیرهای محلی، دارای کلاس حافظهٔ اتوماتیک هستند. زیرا هنگام ورود به تابع به‌طور اتوماتیک ایجاد می‌شوند و هنگام خروج از تابع نیز به‌طور اتوماتیک از بین می‌روند. برای تعیین کلاس حافظه اتوماتیک، از کلمهٔ کلیدی auto استفاده می‌شود. اما کامپایلر به‌طور پیش‌فرض برای متغیرهایی که کلاس حافظه برای آنها ذکر نگردد نیز این نوع از کلاس حافظه را در نظر می‌گیرد، لذا الزامی در به‌کارگیری این کلمه در تعیین کلاس حافظهٔ auto نیست. به‌عنوان مثال، دستور زیر، متغیر m را با کلاس حافظهٔ اتوماتیک تعریف می‌کند:

```
auto float m;
```

بنابراین با توجه به مطالب بالا متغیرهایی که با کلاس حافظهٔ اتوماتیک تعریف می‌شوند دارای دو ویژگی عمده هستند. اولاً این متغیرها از نظر حوزهٔ دسترسی محلی می‌باشند، و ثانیاً با فراخوانی تابعی که در آن تعریف شده‌اند مقدار حافظهٔ لازم به آنها اختصاص می‌یابد و با خاتمه اجرای تابع از بین می‌روند و لذا طول عمر متغیرهای اتوماتیک محدود به بازهٔ زمانی است که تابع مربوطه در حال اجرا می‌باشد.

1.storage class 2. scop 3.life time 4. automatic 5. register 6. static 7. extern

کلاس حافظهٔ ثبات

کلاس حافظه ثبات به کامپایلر پیشنهاد می‌کند که متغیر اتوماتیک را در ثبات پردازنده قرار دهد. بنابراین، حوزه و طول عمر متغیرهای کلاس حافظهٔ ثبات مثل کلاس حافظهٔ اتوماتیک است. همان‌طور که می‌دانید، ثبات ها حافظه‌هایی در داخل پردازنده هستند. کامپیوتر برای انجام محاسبات برروی متغیرها، آنها را از حافظهٔ **RAM** به حافظهٔ **cache** و سپس به ثبات‌ها ارسال می‌کند و پس از انجام محاسبات، به حافظهٔ **RAM** برمی‌گرداند. اگر کامپایلر بتواند، متغیرهایی را در ثبات نگه دارد، سرعت انجام محاسبات با آن متغیرها افزایش می‌یابد.

اما تعداد ثبات‌های پردازنده محدود بوده، و پردازنده برای انجام محاسبات خود از آنها استفاده می‌کند. در صورتی که ثبات های خالی وجود داشته باشد، می توان از آنها برای ذخیره متغیرها استفاده کرد، و از صرف زمان جهت تبادل داده‌ها بین **RAM** و ثبات‌های پردازنده خودداری کرد. به همین دلیل، تعیین کلاس حافظه ثبات، حالت پیشنهاد را برای کامپایلر دارد. اگر کلاس حافظهٔ متغیری را ثبات تعیین کنید ولی پردازنده نتواند ثبات خالی را در اختیار آن متغیر قرار دهد، کلاس حافظهٔ ثبات برای آن متغیر حذف می‌شود و کلاس حافظهٔ آن به کلاس حافظهٔ اتوماتیک تغییر می یابد. لذا پیشنهاد می‌شود فقط متغیرهای مهم برنامه را که نیاز به انجام سریع محاسبات با آنها است، با این کلاس حافظه مشخص کنید، مثل اندیس حلقهٔ تکرار. برای تعیین کلاس حافظهٔ ثبات از کلمهٔ کلیدی **register** استفاده می‌گردد. به‌عنوان مثال، دستور زیر متغیر **p** را با کلاس حافظهٔ ثبات تعریف می‌کند:

```
register int p;
```

کلاس حافظهٔ ثبات محدودیت‌هایی دارد که عبارت است از:

۱. فقط برای متغیرهای محلی قابل استفاده است.
۲. انواع کاراکتر، صحیح و اشاره گر می توانند با کلاس حافظهٔ ثبات تعریف شوند. در مورد سایر انواع باید به مستندات کامپایلری که از آن استفاده می‌کنید، مراجعه نمایید.
۳. دستیابی به آدرس متغیرهایی با کلاس حافظهٔ ثبات، معنی ندارد، چون ثبات‌ها پردازنده فاقد آدرس می‌باشند. با نحوهٔ دسترسی به آدرس متغیرها در فصل اشاره‌گرها آشنا خواهید شد.

کلاس حافظهٔ استاتیک

بر عکس دو کلاس حافظهٔ قبلی متغیرهای استاتیک را می‌توان هم به‌صورت محلی و هم به‌صورت سراسری تعریف کرد. به‌عبارت دیگر متغیرهای استاتیک محلی در داخل تابع و متغیرهای استاتیک سراسری در خارج از تابع تعریف می‌شوند. مقدار اولیه متغیرهای استاتیک محلی و استاتیک سراسری هر دو برابر صفر است. از طرفی متغیرهای استاتیک سراسری تفاوت چندانی با متغیرهای اتوماتیک سراسری ندارند و لذا تنها به بررسی متغیرهای استاتیک محلی می‌پردازیم. متغیرهای استاتیک محلی دارای این ویژگی هستند که در اولین باری که تابع فراخوانی می‌شود ایجاد می‌گردند و تنها یکبار مقدار اولیه می‌گیرند. همچنین هنگام خروج از تابع، آخرین مقدار خودشان را حفظ می‌کنند. لذا در فراخوانی های بعدی تابع، دستوری که موجب تعریف و مقداردهی اولیه به این متغیرها می‌گردد اجرا نمی‌گردد، تا مقدار قبلی متغیر استاتیک حفظ گردد.

مثال ۹-۲۱ : برنامه‌ای که با تولید خروجی ساده‌ای، تفاوت بین متغیرهای اتوماتیک و استاتیک محلی را نشان می‌دهد. این برنامه را خود اجرا کنید و خروجی آن را مشاهده کنید.

```
1.  //This program demonstrates static variables.
2.  #include <iostream>
3.  using namespace std;
4.  float AVG(float);
5.  int main()
6.  {
7.     float score=1, avg;
8.     while(score)
9.     {
10.      cout<<"Enter a score: ";
11.       cin>>score;
12.     avg = AVG(score);
13.      cout<<"New average is : "<<avg<<endl;
14.    }
15.   return 0;
16. }
17. float AVG(float new_score)
18. {
19.   static float total;  //static variables are initialized
20.   static int count=0;  //only once per program
21.   //total=0;
22.   //The line 21 is unnecessary, becuase static variable
23.   //initialized to ziro atomaticly.
24.   count++;                //increment count
25.   total +=new_score;      //add new data to total
26.  return total / count ;   //return the new average
27. }
```

توضیح مثال : در این برنامه با دریافت نمرات یک دانشجو از کاربر، مقدار معدل را به‌صورت لحظه به لحظه با توجه به آخرین نمرهٔ ورودی محاسبه کرده و نمایش می‌دهیم. در تابع **AVG** دو متغیر محلی **total** و **count** به‌صورت استاتیک تعریف شده‌اند. این دو متغیر در اولین باری که تابع **AVG** اجرا می‌شود با مقدار صفر مقدار اولیه‌دهی می‌شوند. اما در دفعات بعدی که تابع اجرا می‌گردد عمل مقدار اولیه دهی انجام نمی‌شود. همچنین متغیر **count** آخرین تعداد نمرات وارد شده و متغیر **total** حاصل جمع نمرات را در خود نگه می‌دارند. لذا هر بار که تابع اجرا می‌گردد نمرهٔ جدید به حاصل جمع نمرات قبلی افزوده شده و معدل جدید محاسبه می‌گردد. حال به‌عنوان یک آزمایش علامت // را از جلوی خط ۲۱ بردارید و برنامه را دوباره اجرا کنید. چه نتیجه‌ای حاصل می‌شود؟ آیا تمایز بین مفاهیم مقداردهی کردن و مقدار اولیه دادن روشن‌تر نشد!؟

کلاس حافظهٔ خارجی

متغیرهایی که درخارج از توابع تعریف می‌شوند (متغیرهای سراسری) دارای کلاس حافظهٔ خارجی هستند. ویژگی‌های این متغیرها عبارت است از:

۱. با شروع اجرای برنامه ایجاد می‌شوند و تا پایان اجرای برنامه حضور دارند (طول عمر).

۲. در توابعی که پس از تعریف این متغیرها قرار داشته باشند قابل استفاده‌اند (حوزه متغیر).

تفاوت متغیرهای اتوماتیک سراسری و متغیرهای استاتیک سراسری

اجازه دهید برای روشن‌تر شدن مبحث ابتدا سؤالی را مطرح کنیم. *اگر متغیری به‌صورت استاتیک سراسری، قبل از تابع اصلی تعریف شود، این متغیر با متغیرهای اتوماتیک سراسری چه تفاوت‌هایی دارد؟* به‌عنوان مثال در دستورات زیر، **x** و **y** متغیرهای استاتیک سراسری و **m** و **n** متغیرهای اتوماتیک سراسری هستند.

```
static int x, y;
int m, n;
int main()
{
  ...
}
```

اگر کل برنامه C++ تنها در یک فایل وجود داشته باشد، در واقع، هیچ تفاوتی بین متغیرهای استاتیک سراسری و متغیرهای اتوماتیک سراسری وجود ندارد. اما گاهی ممکن است برنامه‌های شما بسیار طولانی باشند و مجبور شوید که بخش‌هایی از برنامه را در چند فایل قرار دهید. در این صورت، متغیرهای استاتیک سراسری فقط در همان فایلی که تعریف شده‌اند و متغیرهای اتوماتیک سراسری در تمام فایل‌ها قابل دستیابی‌اند و برای استفاده از آنها در فایل‌های دیگر، باید آنها را با دستور **extern** در فایل‌های دیگر به کامپایلر اعلان کرد. دستور **extern** به کامپایلر می‌گوید برای این متغیرها حافظه جدیدی در نظر نگیرد، بلکه از همان حافظه‌ای که در فایل دیگر به آنها اختصاص داده شد استفاده کند. لذا، در شکل ۹-۴ متغیرهای **x** و **y** فقط در فایل **T1.cpp** ولی متغیرهای **m** و **n** در هر دو فایل **T1.cpp** و **T2.cpp** قابل دسترس هستند.

فایل T1.cpp

```
static int x,y;
int m,n;
void f1(void)
int main()
{
…
}
void f2(void)
{
   m=5; x=6;
}
```

فایل T2.cpp

```
extern int m,n;
void f2(void)
{
   m=n/m;
}
void f3(void)
{
   m=50;
}
```

شکل ۹- ۴ : تفاوت متغیرهای استاتیک سراسری و متغیرهای اتوماتیک سراسری

۹–۵ : برنامه‌های چند فایلی

هنگامی که یک پروژهٔ نرم‌افزاری بزرگ به نتیجه می‌رسد، در واقع کار گروهی تعدادی متخصص نرم‌افزار است که به بار می‌نشیند. یکی از اولین دلایل به‌وجود آمدن ایدهٔ پیمانه‌ای کردن[1] نرم‌افزارها دقیقاً همین مسئله است که بتوان یک مسئله را به چندین زیر مسئلهٔ کوچکتر (به چندین پیمانه) تقسیم کرد و هر بخش یا پیمانه را به یک گروه خاص از نرم‌افزار نویسان سپرد تا برروی آن کار کنند، و سپس این بخش‌ها یا پیمانه‌ها در کنار یکدیگر قرارمی‌گیرند تاحلی برای مسئلهٔ بزرگتر پدید آید. به‌عنوان مثال فرض کنید در یک تیم ۲ نفری می‌خواهند برنامه‌ای بنویسند که اعمال جمع و تفریق و ضرب دو ماتریس خیلی بزرگ با ابعاد نامحدود m×n را انجام دهد. در این حالت ممکن است مسئله را به دو بخش مجزا مثلاً قسمت محاسبه کنندهٔ نتایج و قسمت ورودی و خروجی داده‌ها تقسیم کنند. حال سؤال این است که چگونه می‌توان به شیوه‌ای درست و با کمترین زحمت نتایج و کدهای به‌دست آمده از کار این دو نفر را با یکدیگر ترکیب کرد تا به جواب نهایی برای این مسئله برسیم!؟

احتمالاً شما پیشنهاد می‌کنید که کدهای هر یک از این افراد را در داخل یک فایل سوم کپی کرده و در نهایت به جواب مسئله دست می‌یابیم. اما این ایدهٔ چندان جالبی نیست چون ممکن است هر یک از این افراد بارها و بارها به دلایل مختلف مجبور به اعمال اصلاحات برروی کد خود باشد و نمی‌توان هر بار که اصلاحات اعمال شد دوباره کدهای تغییر یافته را کپی کرد، لذا بهتر است راهی بیابیم که کد تولید شده توسط هریک از این افراد به‌صورت یک موجودیت مستقل عمل کند تا تغییر در متن کد یک گروه، نیازمند تغییرات چندانی در کل پروژه نباشد. از طرف دیگر آیا بهتر نیست راهی پیدا کنیم تا بتوانیم از کدهای تولید شده در این پروژه، در پروژه‌های آتی خود نیز استفاده کنیم!؟ اینجا است که ایدهٔ برنامه‌های چند فایلی مشکل ما را حل می‌کند.

در واقع هنگامی که سرفـایلی را به برنامهٔ خود الحاق می‌کنید، دارید از تکنیک برنامه‌های چند فایلی بهره می‌گیرید. چنانکه در ادامه خواهید دید به‌آسانی می‌توان برنامه‌ای را در چندین فایل توزیع کرد تا به‌راحتی بتوان از ماجول‌های (توابع یا همان پیمانه‌های) نوشته شده برای یک برنامه در هر برنامه دیگری استفاده کرد. مثلاً در سرفایل math.h بسیاری از توابع ریاضی تعریف شده‌اند و شما در برنامه‌های مختلفی از آنها بهره می‌گیرید بدون اینکه کوچکترین زحمتی در جهت کپی کردن آن توابع در کد منبع خود متحمل شوید و تنها با الحاق سر فایل math.h به کد منبع خود که حاوی اعلان این توابع ریاضی است قادر به استفاده از این توابع ریاضی هستید، و چنانکه پیشتر نیز در انتهای فصل سوم مطرح کردیم عمل کپی کردن تعاریف این توابع به کد منبع توسط کامپایلر و پیونددهنده صورت می‌گیرد. در گذشته در زبان‌های برنامه‌نویسی غیر شئ‌گرا کتابخانهٔ توابع وجود داشت که شامل تعریف تعداد زیادی از توابع کاربردی بودند. مثل stdlib.h یا stdio.h در زبان C. اما پس از پیدایش زبان‌های شئ‌گرا علاوه بر کتابخانهٔ توابع، کتابخانهٔ کلاس‌ها نیز به این زبان‌ها اضافه شد. مثل iostream و vector در C++. هدف ما نیز در این قسمت تنها فراگیری روشی برای ساخت کتابخانه‌ای از توابع است.

---

1. madularity

برای ایجاد برنامه‌های چند فایلی ابتدا پروژه‌ای بر مبنای **Win32 Console Application** مطابق آنچه که در فصل سوم بیان شد ایجاد کنید و نام آن را به‌عنوان مثال **test** قرار دهید. همچنین در پنجرهٔ **Step 1 of 1** گزینهٔ **A Simple Application** را انتخاب کنید و دکمهٔ **Finish** را بزنید تا مراحل ساخت پروژه تکمیل گردد. سپس در پنجرهٔ **File View** فایل **test.cpp** را انتخاب کنید. مشاهده می‌کنید که با یک تابع **main** عادی روبرو هستید[1]. در این قسمت برنامه‌ای با دو هدف خواهیم نوشت، اول اینکه بیاموزیم چگونه توابع مورد نیاز یک برنامه را در فایل‌های مختلفی توزیع کنیم، سپس آنکه نکتهٔ قابل توجه‌ای را در خصوص آرگومان‌های دارای مقدار پیش‌فرض و اعلان دوبارهٔ توابع می‌آموزیم.

برنامه‌ای که می‌خواهیم در این قسمت بنویسیم شامل یک تابع است با دو آرگومان از نوع **int** که در این تابع اعداد صحیح بین این دو آرگومان به نمایش در می‌آیند. تا اینجا باید متوجه شده باشید که تابع **main** در کدام فایل باید قرار داشته باشد. اما چنانکه پیش‌تر قرار گذاشتیم برای تعریف هر تابعی غیر از تابع **main** به‌طور استاندارد نیازمند یک اعلان و یک تعریف هستیم. فرض کنید تابعی که می‌خواهیم در این برنامه تعریف کنیم دارای اعلان زیر باشد:

```
void display(int, int);
```

چنانکه قبلاً نیز بحث شد اعلان توابع باید در داخل یک فایل با پسوند **.h** قرارگیرد. برای این منظور در پنجرهٔ **File View** برروی پوشهٔ **Header Files** راست کلیک کرده و گزینهٔ **Add Files to Folder...** را انتخاب کنید سپس پنجره‌ای باز می‌شود که شما می‌توانید فایل موردنظر خود را به پروژهٔ خود اضافه کنید. در قسمت **File name** نام سرفایل را به‌صورت **header.h** وارد کرده و سپس دکمهٔ **Ok** را بفشارید، پیغامی ظاهر می‌شود، دکمهٔ **Yes** را بفشارید. سپس در پنجرهٔ **File View** برروی فایل **header.h** دو بار کلیک کنید پیغامی ظاهر می‌شود با این مضمون که، این فایل موجود نیست آیا مایل به ساخت آن هستید؟ دکمهٔ **Yes** را بزنید و به فایل **header.h** وارد شوید. اعلان تابع مذکور را در این تابع وارد کنید و سپس آن را ذخیره نمایید. پس از آن فایل دیگری با نام **header.cpp** در پوشهٔ **Source Files** مشابه روش فوق ایجاد کنید. در این فایل در ابتدا باید فایل **StdAfx.h** را الحاق کنید و سپس توابع موردنظر خود را بنویسید. لذا این فایل را به‌صورت زیر اصلاح کنید:

```
#include <StdAfx.h>
//function definition, You can define more than one function
void display(int x,int y)
{
	for(int i=x; i<=y; i++)
		cout<<i<<endl;
}
```

در مرحلهٔ بعد باید سرفایل **iostream** در داخل تابع **StdAfx.h** اضافه کنیم تا بتوان در کلیهٔ فایل‌های برنامه از اشیائی مثل **cin** و **cout** استفاده کنیم لذا دو خط زیر را قبل از آخرین خط فایل **StdAfx.h** بیفزایید:

```
#include <iostream>
using namespace std;
```

---

۱. فعلاً به آرگومان‌هایی که برای تابع **main** ذکر شده توجهی نکنید این موضوع به تفصیل در قسمت بعدی بررسی می‌شود.

حال فایل **test.cpp** را نیز مطابق دستورات زیر اصلاح کنید:

```
#include <stdafx.h>
#include "header.h"
int main()
{
      display(10,20); //ok; display from 10 to 20
       return 0;
}
```

حال اگر برنامهٔ خود را کامپایل و اجرا کنید خواهید دید که اعداد ۱۰ تا ۲۰ برروی صفحهٔ کنسول نمایش داده می‌شود. اما ذکر چند نکته خالی از لطف نیست:

۱. برای الحاق کردن سر فایل **header.h** نمی‌توان از < > استفاده کرد، بلکه حتماً باید نام این سرفایل در داخل یک جفت کوتیشن قرارگیرد. این بدان دلیل است که این سرفایل را ما خودمان تعریف کردیم، ولی برای سرفایل‌های مربوط به خود کامپایلر **C++** هم‌می‌توان از علامت<> استفاده‌کرد و هم می‌توان از علامت " " استفاده کرد.

۲. فایل **test.cpp** را به‌صورت زیر تغییر دهید و یک بار دیگر برنامهٔ خود را کامپایل و اجرا کنید، چه نتیجه‌ای مشاهده می‌کنید!؟

```
#include <stdafx.h>
#include "header.h"
void display(int, int =200); //redeclaration function
int main()
{
      display(10,20); //ok; display from 10 to 20
       cin.get();
      display(30);    //ok; display from 30 to 200
       return 0 ;
}
```

۳. حال اگر تابع فایل **test.cpp** را به‌صورت زیر تغییر دهید، چه نتیجه‌ای مشاهده می‌کنید؟

```
#include <stdafx.h>
#include "header.h"
void display(int, int y=200); //redeclaration function
void display(int x=100, int); //redeclaration function
int main()
{
      display(10,20); //ok; display from 10 to 20
       cin.get();
      display(30);    //ok; display from 30 to 200
       cin.get();
      display();      //ok; display from 100 to 200
       return 0 ;
}
```

۴. بسیاری از شرکت‌ها هستند که ماجول‌هایی را برای استفادهٔ دیگر برنامه‌نویسان تهیه می‌کنند و به فروش می‌رسانند. در این حالت آنچه در اختیار مشتریان خود قرار می‌دهند یک فایل با پسوند **lib.** و یک فایل با پسوند **h.** می‌باشد. مسلماً بدین‌وسیله آنچه که به پیاده‌سازی توابع و الگوریتم‌ها مربوط می‌شود و در فایل **cpp.** قرار دارد در اختیار دیگران قرار نمی‌گیرد. اما ببینیم چگونه می‌توان فایلی با پسوند **lib.** ساخت و به چه شیوه‌ای می‌توان از آن در برنامه‌ها استفاده کرد.

فرض کنید می‌خواهید چگونگی پیاده‌سازی تابع **display** که در بالا بحث شد را از دسترس دیگر کاربران پنهان نگاه دارید، برای این منظور ابتدا باید پروژهٔ جدیدی در محیط **VC6** بر مبنای **Win32 Static Library** ایجاد کنید. یک نام دلخواه برای پروژهٔ خود انتخاب کنید مثلاً **DIS** و کلید **OK** را بفشارید. در پنجرهٔ **Step 1 of 1** گزینهٔ **Pre–Compiled header** را انتخاب و دکمهٔ **Finished** را بفشارید تا پروژه ساخته شود. در منوی **Project** گزینهٔ **Files** را از قسمت **Add to Project** انتخاب کنید. سپس سر فایل **header.h** و فایل **header.cpp** را از این طریق به پروژهٔ خود اضافه نمایید. همچنین سرفایل‌ها و فضاهای نام موردنیاز خود که در توابع فایل **header.cpp** مورد استفاده قرار داده‌اید به سرفایل **StdAfx.h** اضافه کنید. مثلاً در اینجا به دلیل استفاده از شئ **cout** در تابع **display** باید تغییرات لازم را در فایل **StdAfx.h** اِعمال کنید تا سرفایل **iostream** در داخل **StdAfx.h** قرار گیرد. لذا خطوط زیر را در خط یکی مانده به آخر فایل **StdAfx.h** و دقیقاً قبل از **#endif** قرار دهید:

```
#include "iostream"
using namespace std;
```

پس از انجام مراحل فوق گزینهٔ **Set Active Configuration...** را از منوی **Build** انتخاب کنید. پنجره‌ای باز می‌شود گزینهٔ **DIS – Win32 Release** را انتخاب و دکمهٔ **OK** را بفشارید. سپس با زدن کلید **F7** پروژهٔ خود را **Build** کنید تا فایلی با نام **DIS.lib** در پوشهٔ **Release** در داخل محلی که پروژه را ذخیره کرده‌اید ایجاد گردد. حال با قرار دادن دو فایل **header.h** و **DIS.lib** در پوشهٔ هر پروژهٔ جدیدی و اضافه کردن آن به پروژه از طریق گزینهٔ **Add Files** و امثال آن قادر هستید از توابعی که در داخل این سرفایل تعریف شده‌اند به آسانی استفاده کنید. به‌خاطر داشته باشید که فایل **DIS.lib** حتماً باید در پوشهٔ **Debug** مربوط به پروژه‌های جدید قرار گیرد تا قابل شناسایی باشد.[۱]

۱. با مراجعه به لوح فشردهٔ کتاب تمامی پروژه‌های بحث شده در این قسمت را می‌توانید در پوشهٔ **TEST** بیابید.

۹– ۶: آرگومان های تابع main

چنانکه پیشتر نیز مطرح شده اولین تابعی که از یک برنامه اجرا می‌شود، تابع **main** آن برنامه است. این تابع همانند توابع دیگر می‌تواند دارای آرگومان باشد. شاید از خود بپرسید اگر تابع اصلی دارای آرگومان باشد، پارامترهای ورودی تابع چگونه می‌تواند به آن ارسال گردد؟ پاسخ این سؤال بسیار ساده است. آیا تا به حال با خط فرمان **Dos** یا **Linux** کار کرده‌اید. به‌عنوان مثال دستور سادهٔ **dir** را در نظر بگیرید. با استفاده از این دستور قادر هستیم محتویات شاخهٔ جاری را ببینیم. مثلاً با اجرای دستور زیر محتویات درایو **C** به نمایش در می‌آید:

```
C:\>dir
```

حال ممکن است بخواهیم اطلاعات را به‌صورت صفحه به صفحه ببینیم. در این صورت پس از دستور **dir** یک عبارت به اصطلاح سوئیچ یعنی **p/** را تایپ می‌کنیم. در این صورت با نوشتن دستور زیر در خط فرمان اطلاعات درایو **C** به‌صورت صفحه به صفحه به نمایش در می‌آید.

```
C:\>dir /p
```

اما در واقع **dir** نام یک برنامهٔ وابسته به سیستم‌عامل است که با قرار دادن آن در خط فرمان و زدن کلید **Enter** موجب اجرای آن می‌شویم. همچنین اگر بخواهیم هرگونه فایل اجرایی که دارای پسوند **exe** است را اجرا کنیم کافی است نام آن را در خط فرمان تایپ کنیم. به‌عنوان مثال فایل اجرایی یکی از برنامه‌هایی را که تاکنون نوشته‌اید، در شاخهٔ جاری قرار دهید و نام آن را تایپ کنید و کلید **Enter** را بزنید. خواهید دید که برنامهٔ موردنظر از طریق خط فرمان اجرا می‌گردد. اما به عبارت **p/** توجه کنید. این عبارت نام هیچ برنامهٔ نیست بلکه یک پارامتر ورودی برای برنامه‌ای است که به‌واسطهٔ دستور **dir** اجرا می‌گردد. در واقع نحوهٔ ارسال پارامتر ورودی به تابع **main** یک برنامه نیز به همین نحو است که در ادامه توضیح آن را خواهیم داد. نکته‌ای که از این بحث حاصل می‌شود این که اگر تابع **main** یک برنامه دارای آرگومان باشد، باید حتماً آن را از طریق خط فرمان اجرا کرد تا بتوان پارامترهای ورودی را برای آرگومان‌های تابع **main**، آن برنامه ارسال کرد.

تابع **main** نمی‌تواند هرگونه آرگومانی داشته باشد، بنابراین اگر آرگومان داشته باشد آرگومان‌های آن از الگوی خاصی پیروی می‌کنند. این تابع می‌تواند دو آرگومان به نام‌های **argc** و **argv** داشته باشد. آرگومان **argc** از نوع **int** بوده، و مشخص کنندهٔ تعداد پارامترهای ورودی در خط فرمان است. چون نام برنامه به‌عنوان یک پارامتر محسوب می‌شود، حداقل مقدار **argc** برابر با ۱ است. بنابراین اگر برنامه‌ای مانند **test** دارای دو پارامتر ورودی باشد عددی که در آرگومان **argc** قرار می‌گیرد برابر با ۳ خواهد بود، زیرا علاوه بر آن دو پارامتر، نام برنامه نیز یک پارامتر محسوب می‌شود. آرگومان **argv** آرایه‌ای از رشته‌ها است که عناصر آن پارامترهای ورودی تابع می‌باشند. لذا کلیه پارامترهای ارسال شده از طریق خط فرمان به تابع اصلی به‌صورت رشته‌ای، در آرایه‌ای از رشته‌ها به نام **argv** ذخیره می‌شوند. بنابراین اگر پارامترهای ورودی تابع **main** شامل تعدادی عدد باشد و بخواهیم از اعدادی که به‌عنوان پارامتر به تابع اصلی ارسال شده‌اند استفاده کنیم، باید ابتدا آنها را از فرم رشته‌ای، به فرم عددی تبدیل کنیم. اگر یک تابع **main** دارای آرگومان باشد باید عنوان این تابع به‌صورت زیر تغییر یابد:

```
int main(int argc , char* argv[])
```

نحوهٔ ارسال پارامتر به تابع main

فرض کنید برنامه‌ای به نام **test.cpp** توسط کامپایلر **C++** ترجمه کرده و برنامه‌ای به نام **test.exe** از آن ساخته شد. برای اجرای این برنامه در سطح سیستم‌عامل کافی است به‌صورت زیر عمل کنیم (با فرض اینکه این برنامه در درایو **C** موجود است):

```
C:\>test
```

چنانکه مشاهده می‌کنید برای اجرای فایل اجرایی یک برنامه نیازی به ذکر پسوند **exe** نیست. حال فرض کنید که این برنامه دارای دو پارامتر باشد، برای اجرای آن در سطح سیستم عامل، باید مقادیر پارامترها را با یک فاصله به‌صورت زیر تایپ کنیم:

```
C:\>test par1 par2
```

par1 و **par2** پارامترهایی هستند که به تابع اصلی ارسال می‌شوند. البته ممکن است تعداد این پارامترها به هر تعداد دلخواهی باشد و یا ممکن است برخی از پارامترها اعداد و ارقام نیز باشند. اما چنانکه پیشتر نیز مطرح شد تمامی این پارامترها به‌صورت رشته در آرایهٔ **argv** ذخیره می‌شوند. در چنین مثالی **argc** حاوی تعداد پارامترهای ورودی و برابر ۳ می‌باشد. و در مورد ترتیب دسترسی به پارامترهای ارسالی به تابع **main** نیز باید دقت داشته باشید که: **argv[0]** به نام برنامه، **argv[1]** به اولین آرگومان، **argv[2]** به دومین آرگومان و **argv[n]** به **n**– اُمین آرگومان اشاره می‌کنند.

مثال ۹–۲۲: برنامه‌ای که یک عدد را به‌عنوان پارامتر ورودی تابع **main** پذیرفته، فاکتوریل آن عدد را به روش بازگشتی محاسبه می‌کند. این برنامه می‌تواند پارامتر دوم نیز داشته باشد. اگر پارامتر دوم برابر با **"display"** باشد، عدد تولید شده در صفحه نمایش چاپ خواهد شد.

```
1.  //This program uses command line arguments.
2.  #include <iostream>
3.  #include <string>
4.  #include <cstdlib>
5.  using namespace std;
6.  unsigned long double fact(int);
7.  int main(int argc, char *argv[])
8.  {
9.     int number;
10.    bool display;
11.    if(argc < 2 || argc>3)
12.      {
13.          cerr<<"Number of parameter is wrong."
14.              <<"usage: number display\n";
15.          exit(0);
16.      }
17.     if(argc == 3 && !strcmp(argv[2], "display"))
18.        display = true;
19.     else
20.       display = false;
```

```
21.      number = atoi(argv[1]);
22.      unsigned long double Factorial = fact(number);
23.        if(display)
24.          cout << Factorial << endl ;
25.    return 0;
26.  }
27.  //**********************
28.  unsigned long double fact(int x)
29.  {
30.      if(x != 0)
31.         return((unsigned long double)x * fact(x - 1)) ;
32.      return 1 ;
33.  }
```

توضیح مثال : در این مثال برای محاسبهٔ فاکتوریل عدد از یک تابع بازگشتی استفاده کرده‌ایم. توابع بازگشتی توابعی هستند که یک تابع، خودش را فراخوانی می‌کند. به‌طوری که در خط ۳۱ این برنامه می‌بینید تابع **fact** خودش را فراخوانی کرده است. با مبحث توابع بازگشتی در فصل آتی به‌طور مفصل آشنا خواهید شد و لذا صرفاً به جهت ایجاد یک پیش‌زمینه در ذهن شما این مثال را در اینجا مطرح کرده‌ایم. در خط ۲۱ این کد با استفاده از تابع **atoi** که یکی از توابع کتابخانه‌ای است رشتهٔ عددی موجود در **argv[1]** را به مقدار عددی صحیح تبدیل کرده‌ایم. به‌کار گرفتن این تابع بدین دلیل است که عدد وارد شده به‌عنوان پارامتر دوم تابع **main**، که فاکتوریل آن باید محاسبه شود، به‌صورت رشته ای به تابع اصلی منتقل خواهد شد. و لذا برای استفاده از آن، باید به‌صورت عددی تبدیل شود. این تابع در هدرفایل‌های **stdlib.h** یا **cstdlib** قرار دارد. از طرفی در ساختار **if** موجود در خط ۱۱ تعداد پارامترهای ارسال شده به تابع **main** را چک می‌کنیم، اگر کاربر تنها نام برنامه را وارد کرده باشد و یا تعداد پارامترهای ورودی بیش از سه عدد باشد ابتدا یک پیغام خطا توسط شئ **cerr** در صفحه چاپ کرده و سپس به وسیلهٔ تابع **exit(0)** که الگوی آن نیز در هدرفایل‌های **stdlib.h** یا **cstdlib** قرار دارد برنامه را خاتمه می‌دهیم. برای اجرای برنامه کافی است فایل اجرایی برنامه را در درایو **C** قرار داده و سپس در خط فرمان چنین عبارتی را تایپ کنید.

```
C:\>9-22 5 display
```

خروجی حاصل از اجرای این فرمان مقدار فاکتوریل عدد ۵ می‌باشد که برابر ۱۲۰ است.

کار در کلاس ۹-۱ :

سعی کنید نحوهٔ عملکرد تابع **fact** *را در مثال ۹-۲۲ با دنبال کردن برنامه کشف کنید. البته در خصوص تکنیک به‌کار رفته در این تابع در فصل آتی به تفصیل سخن خواهیم گفت.*

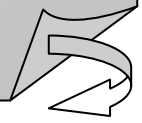
۹-۷ : ماکروها و فراخوانی توابع با نام

پیشتر فراخوانی با مقدار را آموختیم، همچنین گرچه بحث و بررسی در خصوص فراخوانی با آدرس را به فصل یازدهم موکول کردیم ولی دیدید که در هنگام ارسال آرایه‌ها به‌عنوان آرگومان به توابع در حقیقت فراخوانی با آدرس صورت می‌گیرد و متوجه شدید که در فراخوانی با آدرس هرگونه تغییری که در داخل تابع بر آرگومان‌های تابع اعمال شود برروی مقادیر پارامترهای ارسالی به تابع تأثیر مستقیم دارد و مقادیر آنها نیز تغییر می‌کند. حال می‌خواهیم نوع دیگری از فراخوانی را بررسی کنیم که چندان در C یا C++ مرسوم نیست و به‌دلیل اینکه این نوع فراخوانی توسط ماکروها پیاده‌سازی می‌شوند و با توجه به مشکلات عدیده‌ای که ماکروها در برنامه‌نویسی ایجاد می‌کنند در اکثر کتب برنامه نویسی C و C++ هیچ سخنی از این نوع فراخوانی به میان نیامده است. ولی ما در اینجا به دلیل کاربردهایی که این نوع فراخوانی می‌تواند داشته باشد آن را در حد بسیار مختصر بررسی می‌کنیم. قبل از آنکه بحث خود را ادامه دهیم توجه شما را به‌کار در کلاس زیر جلب می‌کنیم:

کار در کلاس ۹-۲ :

به نظر شما اگر در شبه کد زیر فراخوانی تابع **Exchange** *از نوع فراخوانی با آدرس باشد چه مقادیری در خروجی به نمایش در می آید.*

```
1.  A[4]←{8,6,4,2};
2.  I←3;
3.  Exchange(x,y)
4.  {
5.      temp ← x;
6.      x ← y;
7.      y ← temp;
8.  }
9.  main()
10. {
11.     Exchange(I,A[I]);
12.     Output(I,A[0],A[1],A[2],A[3]);
13. }
```



اگر فراخوانی با آدرس صورت می‌گیرد پس هرگونه تغییری که در آرگومان‌های ورودی تابع **Exchange** رخ دهد برروی مقادیر پارامترهای ارسالی تأثیر می‌گذارد و لذا خروجی به‌صورت زیر خواهد بود:

2,8,6,4,3

اما اگر فراخوانی بالا به‌صورت فراخوانی با نام بود، پس از اجرای دستور خط ۱۱ در کد قبل انتساب‌های زیر صورت می‌گرفت:

```
x ← I , y ← A[I]
```

لذا با اجرای خطوط ۵ و ۶ در متغیر **temp** مقدار ۳ و در متغیر **x** مقدار ۲ قرار می‌گرید زیرا :

```
temp ← 3;
x ← A[3];
```

تا اینجا همه چیز مطابق انتظار پیشرفت رفت اما یک اشتباه کوچک کردیم، و آن اینکه چون فراخوانی با نام صورت گرفته به‌جای نام متغیر **x**، نام متغیر ورودی یعنی **I** در دستورات تابع جایگزین می‌شود و به‌جای **y** نام **A[I]** جایگزین می‌شود، بنابراین خطوط ۵ و ۶ در حقیقت به‌صورت زیر اجرا می‌شوند:

```
temp ← I;
I ← A[I];
```

لذا ابتدا در متغیر **temp** مقدار موجود در متغیر **I** قرار گرفته و سپس در متغیر **I** مقدار **A[I]** یا به‌عبارت دیگر **A[3]** قرار می‌گیرد. بنابراین متغیر **I** از این پس دارای مقدار ۲ خواهد بود. حال به‌نظر شما دستور موجود در خط هفتم چگونه باید ارزیابی شود!؟

دستور موجود در خط هفتم نیز با قرار گرفتن نام پارامترهای ارسالی به‌جای آرگومان‌های تابع باید ارزیابی شود، لذا خواهیم داشت:

```
A[I] ← temp;
```

و لذا چون در خط قبلی مقدار متغیر **I** به مقدار ۲ تغییر یافته است لذا دستور فوق به‌صورت زیر اجرا خواهد شد:

```
A[2] ←  3;
```

بنابراین خروجی حاصل از یک فراخوانی با نام به‌صورت زیر در خواهد آمد:

```
2,8,6,3,2
```

اما چه نتیجه‌ای از مثال فوق می‌توان گرفت. وقتی پارامترها از طریق نام ارسال می‌شوند، پارمترهای واقعی از نظر متنی، به‌جای آرگومان‌ها در داخل تابع قرار می‌گیرند. این روش تنها در زبان **ALGOL** به‌طور جدی قابل پشتیبانی و پیاده‌سازی بود، و دیگر زبان‌ها به ندرت از آن پشتیبانی می‌کنند. اما در زبان‌های **C** و **C++** نیز می‌توان به‌واسطهٔ ماکروها کاری کرد که کامپایلر در زمان ترجمه رفتاری از خود نشان دهد تا این جایگزینی متنی صورت گیرد، و لذا متذکر می‌شویم که این روش فراخوانی توابع، جزء روش‌های استاندارد **C++** نیست و توصیه هم می‌کنیم که هرچه کمتر از این روش و به‌طور کلی از ماکروها استفاده کنید. بنابراین قبل از آنکه مثالی در این زمینه ارائه کنیم اجازه دهید به بررسی دقیق‌تری از ماکروها در زبان **C** و نحوهٔ عملکرد کامپایلر در هنگام برخورد به یک ماکرو بپردازیم.

مسلماً از فصل سوم به‌خاطر دارید که برای تعریف ثوابت می‌توانستیم از ماکروها به شیوهٔ زیر استفاده کنیم:

```
#define    PI    3.14
```

ماکروها جزء دستورات پیش‌پردازنده محسوب می‌شوند و لذا باید در ابتدای برنامه نوشته شوند و به سمیکالن نیز ختم نمی‌گردند. اما واقعاً کامپایلر با برخورد به دستور فوق چه می‌کند!؟ در واقع هنگامی که کامپایلر به دستور فوق می‌رسد قبل از هرگونه کامپایلی، هر کجای برنامه که لیترال **PI** وجود داشته باشد مقدار **3.14** را به‌جای آن قرار می‌دهد و سپس به کامپایل برنامه می‌پردازد. حتی می‌توان از این هم فراتر رفت و برخی نمادهای موجود در زبان **C++** را نیز به واسطهٔ ماکروها تغییر داد. مثلاً اگر دو ماکروی زیر را در ابتدای برنامهٔ خود قرار دهید می‌توانید در هر کجا که نیازمند قرار دادن { هستید از عبارت **Begin** و هر کجا که نیازمند } هستید از عبارت **End** استفاده کنید.

```
#define  Begin  {
#define  End    }
```

اما مسلماً این تنها کاربرد ماکروها نیست چنانکه پیشتر در همین فصل مشاهده کردید توابع تک خطی و کوچک را می‌توان به واسطهٔ ماکروها پیاده‌سازی کرد. شکل کلی به‌کارگیری ماکروها برای تعریف توابع به‌صورت زیر است:

```
#define  نام ماکرو (اسامی آرگومان ها)  تعریف تابع
```

نام ماکرو بهتر است با حروف تمام بزرگ انتخاب شود و حداقل باید یک فاصله با دستور **define** داشته باشد. پس از آن بدون قرار دادن فضای خالی، در داخل یک جفت پرانتز اسامی آرگومان‌های تابع موردنظر خود را ذکر کنید، و توجه داشته باشید که اگر تعداد آنها بیش از یکی است آنها را با کاما از یکدیگر جدا کنید. پس از آن تعریف تابع باید ذکر گردد.

هشدار : در لیست آرگومان‌ها نباید نوع آرگومان‌ها ذکر گردد. چنانکه پیشتر نیز مطرح شد در ماکروها هیچ کنترل نوعی صورت نمی‌گیرد.

به‌عنوان مثال در زیر ماکرویی تعریف شده که قرینهٔ یک عدد را بر می‌گرداند:

```
#define  NEGATIVE(x) -x
```

هنگامی می‌خواهیم ماکروی فوق را برای متغیری مثل **Y** فراخوانی کنیم فارق از نوع این متغیر که **int** یا **double** یا هر نوع دیگری که باشد کافی است بنویسیم:

```
Y = NEGATIVE(Y);
```

هنگامی که کد خود را کامپایل می‌کنید، قبل از انجام هرگونه عمل کامپایل، یک سری جایگزینی برای دستورات پیش‌پردازنده صورت می‌گیرد و لذا دستور فوق به‌صورت زیر جایگزین می‌شود، البته این جایگزینی به‌صورتی نیست که کد منبع را تغییر دهد، بلکه در فایل‌هایی که خود کامپایلر می‌سازد اعمال می‌شود:

```
Y = -Y;
```

یک ماکرو حتی می‌تواند یک عبارت منطقی باشد، مثلاً ماکروی زیر بخشپذیری بر ۳ را معین می‌کند:

```
#define MOD3(n)   n%3==0
```

حال می‌توان این ماکرو را به‌صورت زیر در یک دستور شرطی به‌کار برد:

```
if(MOD3(x))
```

در نوشتن و فراخوانی ماکروها باید دقت کافی به خرج داد تا از اجرای آنها نتیجه موردنظر به‌دست آید. مثلاً ماکروی زیر را که عمل مجذور کردن را انجام می دهد، در نظر بگیرید:

```
#define SQUARE(x) x*x
```

حال فراخوانی زیر را در نظر بگیرید:

```
Sq = SQUARE(n);
```

با این فراخوانی مقدار عبارت **n*n** در داخل متغیر **Sq** به درستی قرار می‌گیرد. اما اگر فراخوانی زیر صورت گیرد چطور؟

```
Sq = SQUARE(n+1);
```

با این فراخوانی مقدار عبارت **n+1*n+1** در متغیر **Sq** جایگزین می‌شود که مسلماً مقدار موردنظر ما را برآورده نمی‌کند. برای رفع این مشکل دو راه‌حل وجود دارد. اول آنکه می‌توان فراخوانی فوق را اصلاح کرد، مثلاً می‌توان ابتدا **n+1** را در یک متغیر دیگر قرار داد و سپس فراخوانی را به واسطهٔ آن متغیر صورت داد، و یا آنکه می‌توان **n+1** را در داخل یک جفت پرانتز قرار داد که در آن صورت فراخوانی فوق به‌صورت زیر انجام می‌شود:

```
Sq = SQUARE((n+1));
```

اما روش دوم اصلاح تعریف خود ماکرو به‌صورت زیر است تا از ایجاد چنین اشتباهی جلوگیری شود:

```
#define SQUARE(x) ((x)*(x))
```

آخرین نکته‌ای که در خصوص ماکروها باید بیان کنیم این است که اگر تعریف تابعی که ماکرو می خواهد آن را پیاده‌سازی کند به‌گونه‌ای باشد که نتوان آن را در یک خط جای داد می‌توان تعریف تابع را در چندین خط ادامه داد به شرط آنکه در انتهای هر خط ناتمام (غیر از خط آخر) از علامت \ استفاده کرد. که نمونهٔ آن را در زیر می‌بینید:

```
#define EXCHANGE(x,y) x=x+y;\
                      y=x-y;\
                      x=x-y
```

کار در کلاس ۹-۳ :

برنامه‌ای به زبان **C++** *بنویسید که با استفاده از ماکروها مساحت دایره‌هایی به شعاع* **r** *را محاسبه کند. نشان دهید که* **r** *می تواند هر نوعی باشد. مثلاً* **int** *یا* **float** *و ...*

و اما تمام این مطالب در خصوص ماکروها بیان شد تا نحوهٔ فراخوانی با نام را بیاموزیم. در زیر مثالی ارائه شده که مبین فراخوانی با نام در C++ است.

مثال ۹-۲۳: به پارامتری که در خطوط ۱۱ و ۱۳ و ۱۶ برای آرگومان **a** ارسال شده و خروجی که در نتیجهٔ آن حاصل گردیده خوب توجه کنید.

```
1.  //This program show Calling function by name.
2.  #include <iostream>
3.  using namespace std;
4.  //**********************
5.  #define SIGMA(s,a,i,n) for(i=1,s=0.0;i<=(n);s+=(a),i++)
6.  //**********************
7.  void main()
8.  {
9.     int i;
10.    float sum;
11.     SIGMA(sum, i, i, 10);
12.       cout << sum <<endl;       // display 55
13.     SIGMA(sum, i*i, i, 10);
14.       cout << sum <<endl;       // display 385
15.    double H10;
16.     SIGMA(H10, 1.0/i, i, 10);
17.       cout << H10 <<endl;       // display 2.92897
18. }
```

کار در کلاس ۹-۴ :

یکی از مشکلاتی که در زمینهٔ کار کردن با ماکروها با آن روبرو هستیم این است که در ماکروها نمی توان هیچ متغیر جدیدی تعریف کرد. و لذا اگر بخواهیم تابع **Exchange** *مطرح شده در کار در کلاس ۹-۲ را در زبان* **C++** *پیاده سازی کنیم باید متغیر* **temp** *در هنگام فراخوانی به تابع ارسال شود. و لذا ممکن است فراخوانی به صورت* **Exchange(x,y,temp)** *باشد که* **temp** *باید توسط کاربر و هم نوع* **x** *و* **y** *تعریف و به ماکرو ارسال شود. ماکروی لازم برای پیاده سازی این تابع را در زبان* **C++** *بنویسید و آن را آزمایش کنید. به نظر شما آیا برای ارسال آرایه به ماکرو باید شیوهٔ خاصی را به‌کار ببرید؟*

۹-۸ : تمرین

۱. مفاهیم آرگومان و پارامتر چه تفاوتی با یکدیگر دارند؟

۲. هر تابع دارای چه جنبه‌هایی است؟ هر یک را توضیح دهید.

۳. به چه روش‌هایی می‌توان یک تابع را فراخوانی کرد؟ تفاوت هر یک را بیان کنید.

۴. از یک تابع چند مقدار می‌تواند برگردانده شود؟

۵. چرا در هنگام تعریف تابع باید حتماً نام آرگومان‌های تابع را ذکر کنیم اما در اعلان تابع قرارداد کردیم که نام آرگومان‌ها ذکر نگردد؟ آیا ذکر نام آرگومان‌ها در اعلان تابع برنامه را با مشکل مواجه می‌سازد؟

۶. چگونه توابعی را می‌توان در یک دستور انتساب فراخوانی کرد؟

۷. منظور از سربارگذاری توابع چیست؟

۸. تعریف توابع کلی چه مزیتی نسبت به سربارگذاری توابع‌دارد؟ نحوهٔ ایجاد توابع‌کلی را نیز شرح‌دهید.

۹. اعلان کردن دوبارهٔ توابع **(redeclaration)** چه نتیجه‌ای در بر دارد؟ مثالی در این زمینه بیاورید.

۱۰.چرا فراخوانی توابع عملی زمان بر برای کامپیوتر محسوب می‌شود؟ چه راه کاری برای گریز از این مشکل پیشنهاد می‌کنید.

۱۱.منظور از کلاس‌های حافظه چیست؟ انواع آن را نیز با ذکر مثالی شرح دهید.

۱۲.تفاوت متغیرهای استاتیک خارجی را با متغیرهای اتوماتیک خارجی به‌طور کامل شرح دهید.

۱۳.نحوهٔ استفاده از آرگومان‌های تابع **main** را شرح دهید.

۱۴.تابعی بنویسید که با دریافت دو عدد صحیح **num** و **exp** ، عدد **num** را به وسیلهٔ ضرب به توان **exp** برساند.

۱۵.برنامه‌ای بنویسید که با دریافت دو عدد ب.م.م آن دو عدد را در تابعی محاسبه و چاپ کند. سپس با ارسال مقدار ب.م.م به تابع دیگری، مقدار ک.م.م دو عدد را نیز محاسبه و به خروجی ببرد.

۱۶.تابع تمرین ۱۴ را به‌گونه‌ای بازنویسی کنید که برای هر نوع دادهٔ ورودی کار کند.

۱۷.تابعی بنویسید که هنگام احضار آن، با چاپ پیغامی اطلاع دهد که چندبار تا کنون آن‌را احضار کرده‌اید. سپس برنامه‌ای بنویسید که دست کم این تابع را ده بار احضار کند.

۱۸.پس از اجرای هر یک از دستورهای زیر، مقدار **x** را نشان دهید.

الف- `x = fabs(7.5);`

ب- `x = floor(-3.4);`

ج- `x = ceil(-5.2);`

د- `x = ceil(-fabs( -8 + floor(5.8)));`

۱۹.برنامه‌ای به زبان C++ بنویسید که با توجه به مطالب مطرح شده در خصوص نگاره‌سازی در فصل قبل توابع ریاضی مثل $y = x^2$ یا $y = \cosh x$ را در صفحه نمایش رسم کند.

۲۰.تفاوت تابع قالب با قالب تابع چیست؟

۲۱.برنامه‌ای به زبان C++ بنویسید که با دریافت یک عدد اعشاری و یک عدد صحیح به‌عنوان دقت موردنظر کاربر، عدد اعشاری را تا تعداد ارقام اعشاری خواسته شده مطابق مطالب مطرح شده در فصل دوم به‌واسطهٔ تابع **floor** گرد کند.

۲۲.برنامه‌ای به زبان C++ بنویسید که نقش آموزش جدول ضرب را ایفا کند. بدین صورت که ابتدا جدول ضرب را برای کاربر نمایش دهد تا مطالعه کند. سپس با استفاده از مولد اعداد تصادفی هر بار دو عدد بین یک تا ده تولید کند و حاصل ضرب آن دو را از کاربر بپرسد اگر کاربر پاسخ درست داد برنامه ادامه پیدا کند در غیر این صورت دوباره جدول ضرب را برای مطالعه نمایش دهد. برنامه باید با کلید **Esc** خاتمه یابد. به‌عنوان مثال خروجی برنامه می‌تواند چنین باشد.

```
How Much is 6 times 7 ?
```

۲۳.برنامهٔ تمرین ۲۱ را طوری تغییر دهید که با استفاده از متغیرهای **static** درصد پاسخ‌های درست، تعداد سؤال‌های صورت گرفته و آماری از این دست را نیز در خروجی چاپ کند.

۲۴.تابعی به نام **distance** بنویسید که فاصلهٔ بین دو نقطه دریافتی از کاربر را محاسبه کند. این دو نقطه ممکن است برروی یک محور، یا در داخل صفحه و یا در داخل فضا باشند. این تابع را برای اعداد صحیح و اعداد اعشاری سربارگذاری کنید. و با استفاده از آرگومان‌های دارای پیش‌فرض توابع را طوری طراحی کنید که هم برای نقطه‌های روی محور و هم برای نقطه‌های روی صفحه و یا نقطه‌های داخل فضا جواب دهد. یعنی تمامی این موارد توسط یک تابع پاسخ داده شود نه سه تابع متفاوت.

۲۵.برنامه‌ای بنویسید که با دریافت ضرایب یک معادلهٔ درجهٔ دوم، ضرایب را به تابعی ارسال کند، سپس تابع تعداد جواب‌ها را به‌عنوان نوع برگشتی خود باز گرداند. و جواب‌های معادله را نیز در یک آرایه به نام **result** قرار دهد. و در نهایت در داخل تابع فراخواننده جواب‌های معادله چاپ شود.

۲۶.برنامه‌ای به زبان C++ بنویسید که محتویات آرایه‌هایی با انواع متفاوت را به واسطهٔ قالب تابع چاپ کند. سپس تابع کلی را برای نوع **int** مجدّداً تعریف کنید.

۲۷.برنامه‌ای بنویسید که توابعی با انواع متفاوت را به دو روش مرتب‌سازی حبابی و مرتب‌سازی درجی مرتب کند. سپس سعی کنید این قالب را برای وکتورها نیز پیاده‌سازی کنید.

۲۸.یک تابع کلی برای جستجوی دودویی بنویسید. سپس برنامه‌ای بنویسید که چند مقدار با انواع متفاوت را به‌عنوان پارامتر ورودی تابع **main** دریافت کند و آنها را در آرایه‌های متفاوتی جستجو کند.

۲۹.هدف از دستور زیر چیست؟

```
template <class A> array <A> :: array (int s)
```

۳۰.آیا قطعه برنامهٔ زیر دارای خطا است؟

```
double cube(int);
int cube(int);
template <A> A func(int b, A c)
{return b*c;}
```

۹-۹: موارد مطالعاتی

۱. برنامه‌نویسان C ممکن است تعجب کنند که با آنکه تابع **realloc** جهت تغییر اندازهٔ حافظهٔ گرفته شده در زبان C وجود دارد، تابع معادلی مثل **renew** در زبان C++ وجود ندارد. در مورد نحوهٔ به‌کارگیری تابع **realloc** در زبان C++ تحقیق کنید.

۲. تابع **main** دارای یک آرگومان سوم نیز هست. در مورد این آرگومان تحقیق کنید. احتمالاً برنامهٔ زیر و توضیحات مندرج در آن می‌تواند شما را در این راه کمک کند. این برنامه را در کامپایلر خود کامپایل کرده و فایل اجرایی آن را در درایو C قرار دهید و از طریق فرمان زیر از خط فرمان آن را اجرا کنید.

```
C:\>Test /n
```

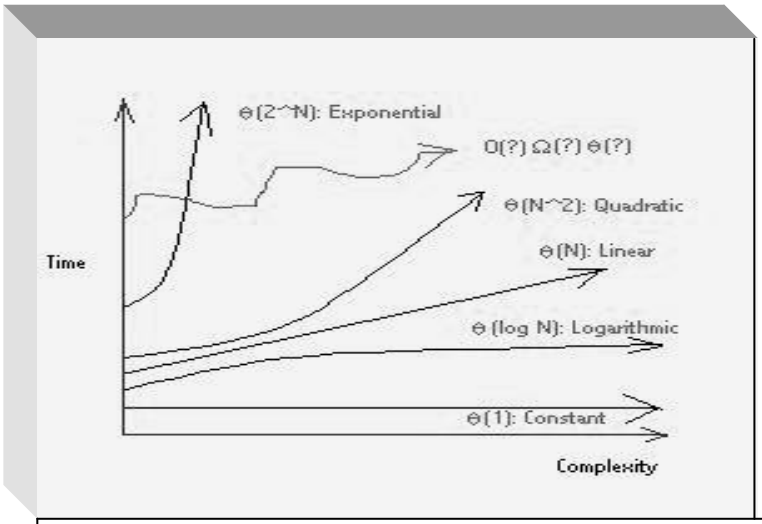
```
1.  //This program uses third argument of main function.
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  void main( int argc, char* argv[], char* envp[] )
6.  {
7.     int iNumberLines = 0;  // Default is no line numbers.
8.  // If more than .EXE filename supplied, and if the
9.  // /n command-line option is specified, the listing
10. // of environment variables is line-numbered.
11.     if( argc == 2 && stricmp( argv[1], "/n" ) == 0 )
12.           iNumberLines = 1;
13. //Walk through list of strings until a NULL is
14. //encountered.
15.     for( int i = 0; envp[i] != NULL; ++i )
16.     {
17.         if( iNumberLines )
18.           cout << i << ": " << envp[i] << "\n";
19.     }
20. }
```

۳. (پروژه برنامه‌نویسی) : به خط فرمان سیستم عامل خود بروید و عبارت **edit** را تایپ کرده و کلید **Enter** را بزنید. برنامه‌ای موسوم به ویرایش‌گر **Dos** باز می‌شود. کمی با این ویرایش‌گر کار کنید تا نحوهٔ عملکرد و امکانات آن را ببینید. سپس توابعی را برای برخی از عملیاتی که در این برنامه امکان‌پذیر است بنویسید. البته برای نوشتن این پروژه نیازمند مطالب مطرح شده در فصل یازدهم هستید، لذا فعلاً فقط برروی ابعاد پروژهٔ خود فکر کنید و زیر روال های لازم را به زبان C++ تهیه و آزمایش کنید. پروژه بر این قرار است که محیطی همانند محیط یک ویرایشگر تجاری و قابل قبول بنویسید. لذا باید برای تک تک کلیدهای صفحه کلید کد بنویسید. حتی باید کلیدهای جهتی (بالا و پایین و ...) و یا کلیدهایی مثل **Insert** و **Delete** و **Back Space** را نیز مدیریت کنید. باید برای برنامه به‌واسطهٔ کلیدهای توابع یعنی کلیدهای **F1** تا **F12** منو طراحی کنید. حتی از شما می‌خواهیم اعمالی چون **copy** و **past** را نیز شبیه‌سازی کنید، گر چه نیازی به طراحی عملکرد ماوس در این برنامه نیست. از جمله اعمال دیگری که نیاز به پیاده‌سازی آنها نیست اعمالی چون **save** یا **open** و امثال آن است که مربوط به هارددیسک

می‌شود و فعلاً از بحث ما خارج است. لذا فعلاً اعمالی را طراحی کنید که از عهدهٔ آن بر می‌آیید. بد نیست از وکتورها به‌طور گسترده در این پروژه استفاده کنید تا بسیاری از مشکلات سر راه شما حل گردد.

۴. برای ایجاد یک رابط خوب در پروژهٔ قبلی نیازمند استفاده از توابع گرافیکی هستید. تحقیق کنید در زبان **C** و **C++** و کامپایلرهای مختلف آنها چه توابع گرافیکی فراهم شده است.

۵. بد نیست بدانید متغیرهای اتوماتیک در **stack** و متغیرهای خارجی و استاتیک در **Heap** سیستم ذخیره می‌شوند. در مورد **stack** و **heap** سیستم عامل تحقیق کنید.



فصل دهم

آشنایی مقدماتی با روش‌های طراحی و تحلیل الگوریتم‌ها

اهداف فصل و چکیده مطالب :

۱. آشنایی با مقدمات تحلیل الگوریتم‌ها
۲. الگوریتم‌های بازگشتی
۳. کاربرد قضیه اصلی در تحلیل الگوریتم‌ها
۴. حل رابطه‌های بازگشتی خطی
۵. آشنایی با حل مسائل به شیوهٔ تکرار
۶. آشنایی با حل مسائل به شیوهٔ تقسیم و حل
۷. آشنایی با برنامه‌نویسی پویا
۸. آشنایی مقدماتی با شبه کد

۱۰-۱ : آشنایی با مقدمات تحلیل الگوریتم ها

پیشتر در فصل ششم سعی کردیم با مطرح کردن برخی الگوریتم‌های مرتب‌سازی و جستجو و ارائهٔ تحلیل بسیار ساده‌ای برروی آنها پیش‌زمینه‌ای را در خصوص تحلیل الگوریتم‌ها در ذهن شما ایجاد کنیم. اما در این فصل قصد داریم مباحث مهمتری را در خصوص طراحی الگوریتم‌ها مطرح سازیم. مباحثی چون الگوریتم‌های بازگشتی، برنامه‌نویسی پویا، تقسیم و حل و امثال آن. البته هدف نهایی در این فصل یافتن مهارت برای تحلیل الگوریتم‌ها است. تا به‌راحتی بتوانید الگوریتم های کارآمد را از الگوریتم‌های ناکارآمد باز بشناسید.

تحلیل الگوریتم‌ها

منظور از تحلیل الگوریتم‌ها مشخص کردن مدت زمان اجرای برنامه‌ها است. می‌دانیم که یک پردازنده در هر لحظه بیش از یک دستور را نمی‌تواند اجرا کند، پس هر برنامه در کامپیوترهای سریال باید دستور به دستور انجام شود، تعداد دستورات برنامه در حقیقت مدت زمان اجرای برنامه را نشان می‌دهد. دراینجا ذکر این نکته لازم است که در عمل بعضی از دستورها مثل جمع به مراتب سریع‌تر از دستوراتی مثل ضرب هستند ولی ما در اینجا مدت زمان اجرای همه دستورات را برابر فرض می‌کنیم، و لذا چنین تصور می‌کنیم که یک خط دستور C++ که به علامت سمی‌کالن ختم می‌شود تنها یک سیکل زمانی از CPU وقت می‌گیرد. برای روشن شدن مطلب ابتدا چند مثال می‌آوریم و سپس به معرفی نمادهای که در زمینهٔ تحلیل الگوریتم‌ها به‌کار می‌رود می‌پردازیم.

مثال ۱۰-۱ : قطعه برنامه روبرو را در نظر بگیرید:

```
1.  a++;
2.  b=a+2;
3.  cin>>a;
```

برنامه روبرو از سه دستور تشکیل شده و هر دستور به یک واحد زمان احتیاج دارد پس زمان اجرای کل برنامه سه واحد است.

مثال ۱۰-۲ : قطعه برنامه روبرو را در نظر بگیرید:

```
1.  a=g-d;
2.    for(i=1; i<=n; i++)
3.        a++;
4.  g=a-d;
```

در برنامه فوق خط اول یک واحد زمان می‌خواهد و در خط دوم و سوم حلقهٔ for ، n مرحله اجرا می‌شود پس n واحد زمان می‌خواهد، خط چهارم هم یک سیکل زمانی احتیاج دارد پس زمان اجرای کل برنامه n+2 واحد زمان است. حال که با زمان اجرای برنامه‌ها تاحدی آشنا شدیم به معرفی چند نماد که در تحلیل الگوریتم‌ها به‌کار می‌رود می‌پردازیم. از این به بعد زمان تخمینی اجرای برنامه را با T(n) نمایش می دهیم، لذا می‌توان در خصوص کد فوق نوشت: $T(n) = n+2$.

نماد O

نماد **O** برای حد بالای زمان اجرای برنامه(یعنی زمان اجرای الگوریتم در بدترین حالت) به‌کار می‌رود، اگر فرض کنیم زمان اجرای برنامه برحسب **n** به‌صورت یک تابع **f (n)** باشد، می‌گوییم یک الگوریتم دارای زمان اجرای **O(g(n))** در بدترین حالت است، اگر یک عدد مثبت و ثابت **c** و یک عدد طبیعی n_0 وجود داشته باشد به‌طوری که برای $n > n_0$ داشته باشیم:

$$0 \le f(n) \le c \times g(n)$$

اجازه دهید این مطلب را با ذکر چند مثال مطلب را روشن‌تر کنیم.

مثال ۱۰-۳ : قطعه برنامه زیر را در نظر بگیرید:

```
1.   a++;
2.    for(i=1; i<=n; i++)
3.         b[i]=c[i+1];
4.   a=b+c;
5.    for(i=1; i<=n; i++)
6.         c[i]=a+b[i+1];
```

همان‌طور که می‌بینید خط اول این برنامه یک واحد زمانی، خط دوم و سوم **n** واحد زمانی، خط چهارم یک واحد زمانی و خط پنجم و ششم هم **n** واحد زمان می‌خواهد، پس برای **T(n)** رابطه زیر را داریم:

$$T(n) = 1 + n + 1 + n = 2n + 2$$

حال اگر $n_0=1$ و $g(n) = n$ و $C=10$ انتخاب کنیم خواهیم داشت:

$$0 \le 2n + 2 \le 10n$$

در نتیجه با توجه به تعریف **O** می‌توان گفت این الگوریتم در بدترین حالت دارای زمان اجرای **n** است و می‌نویسیم:

$$T(n) = O(n)$$

مثال ۱۰-۴ : قطعه برنامه روبرو را در نظر بگیرید:

```
1.  a++;
2.   a=b+c;
3.    if(a==b)
4.       a++;
5.    else
6.       b++;
7.   a=b+c;
```

در این مثال خط اول یک واحد زمانی، خط دوم هم یک واحد زمانی و خط سوم و چهارم و پنجم و ششم هم یک واحد زمانی و در نهایت خط آخر هم یک واحد زمانی می‌خواهد:

$$T(n) = 1 + 1 + 1 + 1 = 4$$

حال اگر $n_0=0$ و $g(n) = 1$ و $c = 5$ انتخاب کنیم خواهیم داشت:

$$0 \le 4 \le 5$$

در نتیجه برای زمان اجرای این الگوریتم در بدترین حالت داریم:

$$T(n) = O(1)$$

مثال ۱۰–۵ : قطعه برنامه روبرو را در نظر بگیرید:

```
1. a++;
2.  a=b+c;
3.    for(i=1; i<=n; i++)
4.      for(j=1; j<=n; j++)
5.         f[i][j]=f[j][i];
6.   a++;
7.      for(i=1; i<=n; i++)
8.         c[i]=a+b[i+1];
```

خطوط اول و دوم این کد هر کدام به یک سیکل زمانی نیازدارند. از آنجا که خط پنجم کد به واسطهٔ دو حلقهٔ تودرتو n^2 بار اجرا می‌گردد، می‌توان گفت خطوط سوم و چهارم و پنجم مجموعاً به n^2 واحد زمانی نیاز دارند. خطوط ششم تا هشتم را هم که قبلاً نمونهٔ تحلیل آنها را دیده اید. بنابراین به‌طور کلی در خصوص زمان اجرای این کد داریم:

$$T(n) = 1 + 1 + n2 + 1 + n = n2 + n + 3$$

حال اگر $n_0 = 10$ و $g(n) = n^2$ و $c = 5$ انتخاب کنیم خواهیم داشت:

$$0 < n^2 + n + 3 < 5n^2$$

و لذا طبق تعریف نماد O در بدترین حالت این الگوریتم دارای زمان اجرای:

$$T(n) = O(n^2)$$

می‌باشد.

تذکر : شاید این سؤال در ذهن شما پیش آمده باشد که، چرا با آنکه مقدار $T(n)$ را در مثال‌های فوق بزرگتر از مقدار O به‌دست آوردیم، اما زمان اجرای بدترین حالت را برابر مقدار O فرض کرده‌ایم و از ثابت c و درجات کوچکتر n چشم‌پوشی کرده‌ایم.!؟ در جواب این سؤال باید گفت؛ در تحلیل الگوریتم‌ها وقتی با تابعی مثل $f(n)$ روبرو هستیم بزرگترین درجه $f(n)$ را برای زمان اجرای بدترین حالت در نظر می‌گیریم زیرا به‌عنوان مثال تأثیری که زمان اجرای دستوری با پیچیدگی n^2 بر روی سرعت برنامه می‌گذارد به مراتب بیشتر از تأثیر دستوری با پیچیدگی زمانی n است. لذا به‌راحتی می‌توان از درجات کوچکتر n در مقابل درجات بزرگتر n چشم‌پوشی کرد. از طرفی از ثابت c نیز چشم‌پوشی می‌کنیم، زیرا آهنگ رشد زمان اجرای الگوریتم‌ها نسبت به n مدنظر ماست و جزئیات زمان اجرا چندان اهمیت ندارد، چنانکه پیشتر نیز دیدید زمان اجرای دستوراتی مثل جمع و ضرب را برابر فرض کردیم. لذا چنین فرض می‌کنیم که زمان اجرای الگوریتمی با پیچیدگی $T(n) = 5n^2$ تقریباً مشابه زمان اجرای الگوریتمی با پیچیدگی $T(n) = n^2$ است. برای روشن شدن موضوع به نمونه‌های زیر توجه کنید:

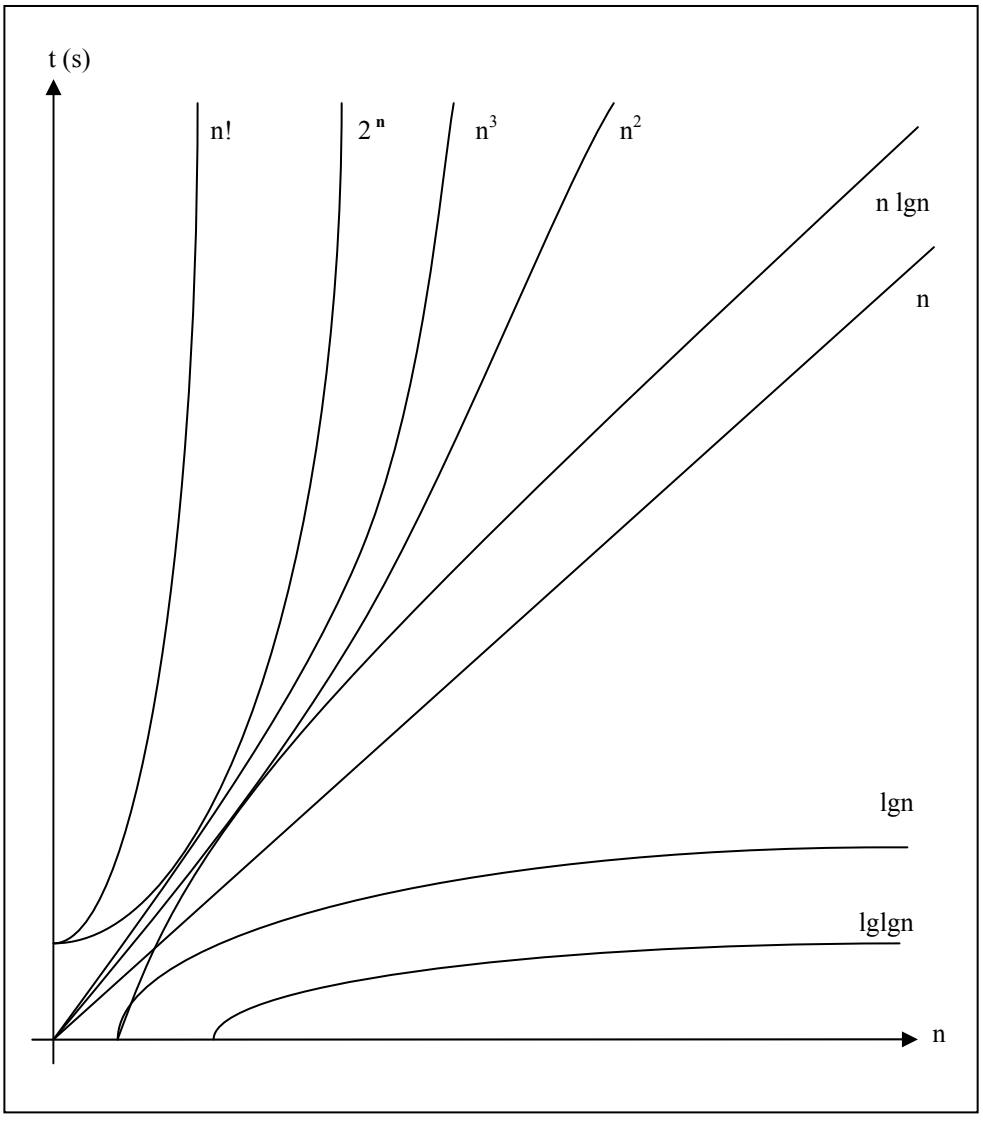
$$T(n) = 9n^5 + 5n^3\lg n + 5 \quad \Rightarrow \quad T(n) = O(n^5)$$

$$T(n) = n + 1825\lg n + 965 \quad \Rightarrow \quad T(n) = O(n)$$

$$T(n) = 2n^3\lg n + n^3 + n\lg n \quad \Rightarrow \quad T(n) = O(n^3\lg n)$$

$$T(n) = n + \lg n + 1 \quad \Rightarrow \quad T(n) = O(n)$$

اگرچه در تمام موارد فوق، همان‌طور که می‌بینید برای تحلیل، بزرگترین درجه را در نظر می‌گیریم، اما نکته دیگر این است که سرعت رشد تابع $n^k \lg n$ به مراتب بیشتر از سرعت رشد تابع n^k است. پس اگر $T(n)$ شامل $n^k \lg n$ و n^k بود برای تحلیل زمان اجرای الگوریتم، $n^k \lg n$ را درنظر می‌گیریم. به‌طور کلی تابعی که بیشترین سرعت رشد را دارد در نظر گرفته می‌شود. برای آنکه سرعت اجرای توابع مختلف را راحت‌تر به خاطر بسپارید به نمودار زیر توجه کنید.



شکل ۱۰- ۱ : سرعت برخی توابع متداول در محاسبهٔ پیچیدگی

نماد Θ

نماد Θ برای نشان دادن حد متوسط زمان اجرای برنامه‌ها (یعنی زمان اجرای الگوریتم در حالت متوسط) به‌کار می‌رود. اگر فرض کنیم زمان اجرای برنامه برحسب n به‌صورت یک تابع f (n) باشد می‌گوییم یک الگوریتم دارای زمان اجرای Θ(g(n)) است اگر دو عدد مثبت و ثابت c و d و یک عدد طبیعی n_0 وجود داشته باشد، به‌طوری که برای $n > n_0$ داشته باشیم:

$$0 \le c \times g(n) \le f(n) \le d \times g(n)$$

به‌عنوان مثال به عبارات زیر توجه کنید:

$$T(n) = n^4 + n + \lg n + 65 \Rightarrow T(n) = \Theta(n^4)$$

$$T(n) = n \lg n + \lg n + 10 \Rightarrow T(n) = \Theta(n \lg n)$$

مثال ۱۰–۶: می‌خواهیم ثابت کنیم $\frac{1}{2}n^2 - 3n = \Theta(n^2)$.

برای این منظور باید یک c و یک d مثبت و یک n_0 پیدا کنیم که برای $n > n_0$ نامساوی زیر برقرار باشد:

$$cn^2 \le \frac{1}{2}n^2 - 3n \le dn^2$$

ابتدا نامساوی فوق را بر n^2 تقسیم می‌کنیم:

$$c \le \frac{1}{2} - \frac{3}{n} \le d$$

این نامساوی برای $n_0=7$ و $c=1/14$ و $d=1/2$ برقرار است. دقت کنید این نامساوی برای مقادیر دیگر این متغیرها هم ممکن است برقرار باشد. اما در اینجا فقط یافتن یک عدد که در نامساوی صدق کند کافی است به همین دلیل ممکن است هر شخص مقادیر متفاوتی را برای n_0 و c و d انتخاب کند.

حال به یک مثال جامع توجه کنید. تابع درجه دوم $f(n) = an^2 + bn + c$ که در آن a و b و c اعداد ثابتی هستند و $a > 0$ است را در نظربگیرید. می‌خواهیم ثابت کنیم $f(n) = \Theta(n^2)$ کافی است یک n_0 و c_1 و d پیدا کنیم که در شرایط زیر صدق کند:

$$0 \le c_1 n^2 \le an^2 + bn + c \le dn^2 \qquad n_0 \le n$$

اگر مقادیر زیر را انتخاب کنیم این نامساوی برقرار خواهد شد. به‌عنوان تمرین مقادیر دیگری به جزء مقادیر زیر پیدا کنید که در نامساوی صدق کند.

$$c_1 = a/4 \quad d = 7a/4 \quad n_0 = 2 \times \max\{(|b|/a), \sqrt{(|c|/a)}\}$$

به‌طور کلی در یک چند جمله‌ای $p(n) = \sum_{i=0}^{d} a_i n^i$ وقتی a_i ها ثابت هستند و $a^d > 0$ رابطهٔ زیر را داریم $p(n) = \Theta(n^d)$. حال اگر $f(n) = 1$ باشد آنگاه طبق رابطهٔ فوق داریم:

$$p(n) = \Theta(n^d) = \Theta(n^0) = \Theta(1)$$

پس هر تابع ثابت دارای پیچیدگی زمانی Θ(1) در حالت معمول است.

نماد Ω

نماد Ω برای نشان دادن حد پایین زمان اجرای برنامه‌ها (یعنی زمان اجرای الگوریتم در بهترین حالت) به‌کار می‌رود. اگر فرض‌کنیم زمان اجرای برنامه برحسب n به‌صورت یک تابع $f(n)$ باشد، گوییم یک الگوریتم دارای زمان اجرای $\Omega(g(n))$ است اگریک عدد مثبت و ثابت c و یک عدد طبیعی n_0 وجود داشته باشد. به‌طوری که برای $n > n_0$ داشته باشیم:

$$0 \le c \times g(n) \le f(n)$$

اگر در تعریف این سه نماد دقت کنید متوجه می‌شوید که اگر تابع $T(n)$ از مقادیر O و Ω یک تابع برابر باشد، تابع $T(n)$ از Θ همان تابع خواهد بود. یعنی:

$$T(n) = O(n) = \Omega(n) \implies T(n) = \Theta(n)$$

کار در کلاس ۱۰-۱ :

توابع زیر را برحسب سرعت رشد آنها مرتب کنید.

$\lg n$
n
$n!$
n^{200}
$n \lg n$
$\lg \lg n$
$n \lg n^3$
1
$n^2 + n + 6$
$\sqrt{n}$
$n^{\frac{1}{5}}$
$(\lg n)!$
2^{2^n}
$\sqrt{\lg n}$
$\ln \ln(\sqrt{n})$
$2^{\lg n}$
$\lg((n)^{\lg n^2})$

۱۰-۲: تمرین

۱. حد بالا و پایین توابع زیر را بیابید.

a. $T(n) = n + \lg n + 1$
b. $T(n) = 9n^5 + 80n^3 + 10n + 1000$
c. $T(n) = n\lg n + n + \lg n + 4$
d. $T(n) = n^{10} + n^9 \lg n + 45$
e. $T(n) = n\lg n^2 + \lg n^5$
f. $T(n) = n \lg \lg n + 1000$
g. $T(n) = n^3 \lg n^2 + n^2 \lg n^{10} + 1$

سپس با استفاده ازحد بالا و پایین، حد متوسط توابع را نیز بیابید.

۲. الگوریتم‌های زیر را تحلیل کنید.

I.
```
1. void function(int n)
2. {
3.    for(i=0; i<n; i++)
4.     for(j=0; j<n; j++)
5.      for(k=0; k<n; k++)
6.       for(p=0; p<n; p++)
7.        a++;
8. }
```

II.
```
1. void function(int n)
2. {
3.    int a=0;
4.     while(a<100)
5.     {
6.       for(i=0; i<n; i++)
7.        for(j=0; j<n; j++)
8.         for(k=0; k<n; k++)
9.          for(p=0; p<n; p++)
10.             a++;
11.    }
12.    a++;
13. }
```

III.
```
1. void function(int n)
2. {
3.    int r=0;
4.      for(i=1; i<=n; i++)
5.       for(j=1; j<=n; j++)
6.        for(k=j; k<=i+j; k++)
7.             r=r+1;
8. }
```

۳. ثابت کنید هر تابع درجه سه از $\Omega(n^3)$ است.

۱۰-۳: الگوریتم های بازگشتی

شیوهٔ بازگشتی یکی از روش‌های بسیار ساده و در عین‌حال خوب برای حل مسأله‌ها است. ایدهٔ حل مسائل از طریق روش بازگشتی از وجود برخی توابع ریاضی که تعریفی بازگشتی داشتند، نشئت گرفته‌است. در ریاضیات برخی توابع وجود دارند که ضابطه آنها ذاتاً به‌صورت بازگشتی تعریف می‌شود، و لذا برای یافتن پاسخ این توابع اصولی وضع شد، که منجر به پیدایش الگوریتم‌های بازگشتی گردید. به‌عنوان مثال به تعریف ریاضی دنبالهٔ فیبوناچی توجه کنید.

$$\text{Fibonacci}(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } n = 1, \\ \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2) & n > 1 \end{cases}$$

در این ضابطه اگر مقدار **n** برابر صفر یا یک باشد مقدار یک به‌عنوان مقدار **Fibonacci(n)** بازگردانده‌می‌شود، و اگر مقدار **n** هر عدد دیگری غیر از مقادیر صفر یا یک باشد، مقدار جملهٔ **n**- اُم این دنباله برابر حاصل جمع دو جملهٔ قبل از خود خواهد بود. مسلماً با توجه به این ضابطه به درستی می‌توانید حدس بزنید که یک تابع بازگشتی چگونه تابعی است. تابع بازگشتی تابعی است که برای محاسبهٔ جملهٔ **n**- اُم آن نیازمند بازگشت به جملات قبلی‌تر و یا به عبارت بهتر ملزم به یافتن مقادیر جملات قبلی باشیم. از طرفی در هر تابع بازگشتی یک یا چند حالت پایه (حالت توقف) وجود دارد. به‌عنوان مثال در ضابطهٔ مربوط به دنبالهٔ فیبوناچی دو جملهٔ اول حالت‌های پایه را تشکیل می‌دهند. زیرا مقدار آنها مستقل از رابطهٔ بازگشتی به‌دست می‌آید. اگر حالت توقف وجود نداشته باشد، رابطهٔ بازگشتی هیچگاه منجر به تولید پاسخ نمی‌شود. زیرا مدام هر جمله جملات قبل از خود را طلب‌کرده و این کار بدون توقف ادامه پیدا می‌کند. اگر تابعی با شرایط فوق داشته‌باشیم (یعنی ضابطهٔ تابع، شامل یک یا چند حالت پایه و یک رابطهٔ بازگشتی باشد)، قادر هستیم به‌راحتی یک الگوریتم بازگشتی برای آن ایجاد کنیم. از طرفی در ادامه خواهید دید که بسیاری از مسائل دیگر که ذاتاً بازگشتی نیستند را نیز با تغییراتی جزئی می‌توان به یک تابع بازگشتی تبدیل کرد و در نهایت به یک الگوریتم بازگشتی برسیم. به‌عنوان مثال می‌توان توابعی چون محاسبهٔ توان یا فاکتوریل را به‌صورت بازگشتی پیاده سازی کرد، در حالی که تعریف ریاضی این توابع بازگشتی‌نیست. همچنین مسائلی را خواهیددید که اصلاً جنبهٔ ریاضی ندارند، امّا برای حل آنها نیازمند ایجاد یک الگوریتم بازگشتی هستیم. بنابراین در ادامه یک روش کلی را برای پیاده‌سازی الگوریتم‌های بازگشتی مطرح می‌کنیم و سپس چندین مثال از الگوریتم‌های بازگشتی را حل خواهیم کرد.

در روش بازگشتی معمولاً مسأله‌ها را بر اساس زیر مسأله‌های قبلی خودش حل می‌کنیم، برای این منظور یک تابع بازگشتی باید خودش را فراخوانی کند. بر فرض اگر ورودی برنامه**n** باشد ما برنامه را طوری طراحی می‌کنیم که اگر **n=1** بود جواب را به‌طور مستقیم تولید کند (حالت پایه). و اگر **n** در شرایط حالت پایه صدق نکند مسأله را به مسأله‌ای برای یافتن جواب **(n-1)** و یا کمتر تبدیل کند و با فراخوانی مجدّد خودش آن را حل کند. پس در حقیقت بایدیک رابطه بین ورودی‌ها پیدا کنیم‌تا با استفاده از آن رابطه و استفاده از فراخوانی تابع برای حـالت‌های پیشین جواب نهایی را بیابیم. به

این رابطه، رابطهٔ بازگشتی و به الگوریتمی که از این رابطه استفاده می‌کند تابع بازگشتی می‌گویند. توابع بازگشتی در C++ اصولاً دارای صورت کلی زیر هستند:

```
return_value   function_name( arguments )
{
      if (به حالت توقف رسیدی)
          {
                    مسئله را برای حالت توقف حل کن
          }
      else
          {
                    تابع را بار دیگر برای مقادیر کوچک‌تر فراخوانی کن
          }
}
```

چنانکه پیشتر مطرح شد یکی از ساده‌ترین و در عین‌حال معروف‌ترین مثال‌ها در روش بازگشتی محاسبهٔ جملهٔ n- اُم فیبوناچی است. دنبالهٔ فیبوناچی به‌صورت زیر است:

1, 1, 2, 3, 5, 8, …

و چنانکه در ضابطهٔ مربوط به این دنباله دیدید، هر عدد برابر جمع دو عدد قبل از آن در دنباله است، یعنی:

```
Fibonacci(n)=Fibonacci(n-1)+ Fibonacci(n-2);
```
رابطه ۱۰-۱

```
Fibonacci(0)=1;
```
رابطه ۱۰-۲

```
Fibonacci(1)=1;
```
رابطه ۱۰-۳

با توجه به رابطه‌های ۱۰-۱ و ۱۰-۲ و ۱۰-۳ به‌راحتی می‌توان تابع زیر را که جملهٔ n- اُم فیبوناچی را محاسبه می‌کند، نوشت. توجه کنید رابطهٔ ۱۰-۱ همان رابطهٔ بازگشتی مورد نظر است که بر اساس آن الگوریتم بازگشتی را می نویسیم.

```
1.  int Fibonacci(int n)
2.  {
3.     if(n==1||n==0)
4.         return 1;
5.     else
6.         return (Fibonacci(n-1)+ Fibonacci(n-2));
7.  }
```

همان‌طورکه در بالا اشاره شد تابع بالا برای ورودی یک وصفر مقدار جواب را به‌صورت مستقیم تولید می‌کند، و گرنه با استفاده از رابطهٔ ۱۰-۱ تابع **Fibonacci** خود را برای ورودی‌های کمتر فراخوانی می‌کند. این فراخوانی تا جایی ادامه پیدا می‌کند که به یک حالت پایه برسیم، در این زمان جواب نهایی محاسبه می‌شود.

تابع فوق را برای ورودی n=3 پیگیری می‌کنیم تا به نحوهٔ عملکرد توابع بازگشتی پی‌ببریم. هنگامی که این تابع برای مقدار n=3 به واسطهٔ دستور زیر فراخوانی می‌شود:

```
Fibonacci(3);
```

چون مقدار ورودی برابر صفر یا یک نیست، پس حالت پایه اجرا نمی‌گردد و به‌جای آن به قسمت **else** رفته و دستور زیر اجرا می‌گردد:

```
return (Fibonacci(2)+ Fibonacci(1));
```
رابطهٔ ١٠-٤

حال با دو فراخوانی مجزا روبه رو هستیم که در زیر نشان داده شده است:

```
Fibonacci(2)
```

و

```
Fibonacci(1)
```

ابتدا دستور سمت چپ علامت + اجرا می‌شود. بنابراین تابع مذکور برای ورودی **n=2** فراخوانی می‌شود. که باز چون **n=2** برابر حالت توقف نیست دستور زیر اجرا می‌گردد.

```
return (Fibonacci(1)+ Fibonacci(0));
```
رابطهٔ ١٠-٥

چون هم **n=1** و هم **n=0** جزء شرایط حالت توقف هستند آنچه که پس از فراخوانی تابع **Fibonacci** برای این مقادیر در دستور فوق جایگزین می‌شود در خط زیر نشان داده شده است:

```
return (1 + 1);
```

بنابراین مقدار ٢ برای فراخوانی **n=2** در رابطهٔ ١٠-٤ جایگزین می شود. و لذا این رابطه به‌صورت زیر در می آید:

```
return (2 + Fibonacci(1));
```

حال نوبت اجرای دستور **Fibonacci(1)** می‌باشد. با اجرای این دستور نیز چون **n=1** است و جزء حالات پایانی است مقدار ١ بازگردانده می‌شود و در نهایت رابطهٔ١٠-٤ به‌صورت نهایی زیر در می‌آید:

```
return (2 + 1);
```

با اجر شدن این دستور مقدار ٣ به‌عنوان مقدار **Fibonacci(3)** بازگردانده می شود. در این مثال دیدید که تابع بازگشتی **Fibonacci** تا جایی که به حالت توقف برسد خود را صدا می زند، هنگامی که به حالت توقف رسید مقدار حالت پایه را برمی‌گرداند و مراحل طی شده را، به عکس می پیماید و با جایگزین کردن مقادیر به دست آمده برای حالت‌های کمتر، در نهایت مقدار نهایی را برای دستور **Fibonacci(n)** محاسبه کرده و مقدار به‌دست آمده را باز می‌گرداند.

با توجه به این مثال می‌توان گفت کلید نوشتن یک برنامهٔ بازگشتی برای یک مسئله پیداکردن یک رابطهٔ بازگشتی برای آن مسأله است. البته چنانکه پیشتر نیز مطرح شد ممکن است یک مسئله ذاتاً بازگشتی نباشد مثل محاسبهٔ فاکتوریل یک عدد. اما می‌توان آن را به‌صورت بازگشتی نیز پیاده‌سازی کرد. و لذا نکتهٔ مهم در جهت ساخت یک الگوریتم بازگشتی برای این مسئله یافتن یک رابطهٔ بازگشتی است که بتوان به‌وسیلهٔ آن فاکتوریل یک عدد را محاسبه نمود.

مثال ۱۰–۷: یک الگوریتم بازگشتی برای محاسبه فاکتوریل اعداد بنویسید.

می‌دانیم که فاکتوریل یک عدد در ریاضیات به‌صورت زیر تعریف می‌شود:

$$0! = 1$$
$$1! = 1$$
$$2! = 2 \times 1$$
$$3! = 3 \times 2 \times 1$$
$$n! = n \times \underbrace{(n-1) \times (n-2) \times \ldots \times (3) \times (2) \times (1)}_{(n-1)\,!}$$

رابطهٔ ۱۰–٦

اما اگر به رابطهٔ ۱۰–٦ دقت کنید این رابطه را می توان به‌صورت زیر به شکل یک رابطهٔ بازگشتی درآورد:

$$n! = n \times (n-1)!$$

رابطهٔ ۱۰–۷

با توجه به رابطهٔ۱۰–۷ که یک رابطهٔ بازگشتی است به‌راحتی می‌توان تابعی که فاکتوریل n را محاسبه می‌کند نوشت:

```
1. int Factorial(int n)
2. {
3.     if(n==0)
4.        return 1;
5.     else
6.        return (n*Factorial(n-1));
7. }
```

حال به تحلیل این الگوریتم می‌پردازیم. هنگامی که این تابع برای مقدار n اجرا می‌شود پس از بررسی شرط if تابع را برای مقدار n–1 اجرا می‌کند، پس در حقیقت مدت زمان اجرای این برنامه برای ورودی n برابر مدت زمان اجرای این برنامه برای ورودی (n–1) به علاوهٔ یک واحد زمان برای اجرای بدنهٔ الگوریتم برای مقدار n است. پس برای مقدار T(n) رابطهٔ زیر را داریم:

$$T(n) = 1 + T(n-1)$$

در حقیقت برای هر T(i) رابطهٔ زیر را داریم:

$$T(i) = 1 + T(i-1)$$

حال این رابطه را حل می‌کنیم تا مقدار T(n) را به‌دست آوریم.

$$T(n) = 1 + T(n-1)$$
$$T(n-1) = 1 + T(n-2)$$
$$T(n-2) = 1 + T(n-3)$$

با خلاصه کردن این رابطه‌ها خواهیم داشت:

$$T(n) = 1 + T(n-1) = 1 + 1 + T(n-2) = 1 + 1 + 1 + T(n-3)$$

$$T(n) = \sum_{j=1}^{i} 1 + T(n-i) = \sum_{j=1}^{n} 1 + T(0) = n + C = \theta(n)$$

مثال ۱۰–۸ : فرض کنید یک جدول با ابعاد $1 \times n$ داریم، می‌خواهیم خانه‌های این جدول را با رنگ‌های آبی و قرمز و سبز رنگ کنیم به‌گونه‌ای که هیچ دو خانهٔ مجاور هم‌رنگ نباشند. یک الگوریتم بازگشتی برای محاسبهٔ تعداد راه‌های رنگ کردن جدول طراحی کنید.

باز ابتدا یک رابطهٔ بازگشتی برای مسأله پیدا می‌کنیم. فرض کنیم تعداد راه‌های رنگ‌آمیزی جدول n خانه‌ای با رنگ های آبی و قرمز و سبز به‌گونه‌ای که هیچ دو خانهٔ مجاور هم رنگ نباشند، $M(n)$ باشد و تعداد راه‌های رنگ‌آمیزی یک جدول n خانه‌ای که خانهٔ اول آن یک رنگ خاص مثلاً آبی است با رنگ‌های آبی و قرمز و سبز به‌گونه‌ای که هیچ دو خانهٔ مجاور هم رنگ نباشند، $K(n)$ باشد، پس برای $M(n)$ و $K(n)$ رابطهٔ $M(n) = 3K(n)$ را داریم. زیرا در حالت اول، خانهٔ اول سه رنگ متفاوت می‌تواند داشته باشد ولی در حالت دوم، خانهٔ اول فقط یک رنگ خاص دارد، حال یک رابطهٔ بازگشتی برای $M(n)$ پیدا می‌کنیم.

خانهٔ اول این جدول آبی یا قرمز و یا سبز است. فرض کنیم سبز باشد پس بقیهٔ جدول یک جدول با ابعاد $(n-1) \times 1$ است که خانهٔ اول آن سبز نیست، پس یا آبی است و با $K(n-1)$ حالت رنگ می‌شود و یا قرمز است و باز با $K(n-1)$ حالت رنگ می‌شود پس درکل با $2K(n-1)$ حالت رنگ می‌شود. مشابه همین استدلال در حالتی که خانهٔ اول آبی و یا قرمز هست داریم.

پس برای $M(n)$ رابطهٔ زیر را داریم:

$$M(n) = 6K(n-1)$$ رابطهٔ ۱۰–۸

حال رابطهٔ در رابطهٔ $M(n) = 3K(n)$ به‌جای n مقدار $n-1$ را قرار می‌دهیم :

$$M(n-1) = 3K(n-1)$$ رابطهٔ ۱۰–۹

حال با توجه به روابط ۱۰–۸ و ۱۰–۹ می‌توان رابطهٔ زیر را نتیجه گرفت:

$$M(n) = 2M(n-1)$$ رابطهٔ ۱۰–۱۰

با توجه به رابطهٔ بالا می‌توان الگوریتم بازگشتی زیر را برای این مسئله نوشت:

```
1. int Color(int n)
2. {
3.     if(n==1)
4.       return 3;
5.     else
6.       return (2*Color(n-1));
7. }
```

برای تحلیل این الگوریتم رابطهٔ بازگشتی $T(n)=1+T(n-1)$ را داریم که حل آن را درمثال قبل دیده‌اید.

مثال ۱۰–۹ : رابطه‌های بازگشتی $T(n) = n + T(n-1)$ را حل کنید.

$$T(n) = n + T(n-1) = n + n-1 + T(n-2) = n + n-1 + n-2 + T(n-3)$$

$$T(n) = \sum_{i=1}^{n} i = n(n-1)/2 = \theta(n^2)$$

۱۰-۴ : تمرین

۱. یک رابطهٔ بازگشتی برای تعداد رشته‌های nتایی متشکل از ۰ و ۱ و۲ که شامل دو صفر متوالی است پیداکنید و سپس یک الگوریتم بازگشتی بنویسید که با دریافت n تعداد رشته‌ها را محاسبه کند.

۲. یک رابطهٔ بازگشتی برای تعداد رشته‌های nتایی متشکل از ۰ و ۱ که شامل دو صفر متوالی نیست پیداکنید و سپس یک الگوریتم بازگشتی بنویسید که با دریافت n تعداد رشته‌ها را محاسبه کند.

۳. یک رابطهٔ بازگشتی برای تعداد راه‌های پرکردن یک مستطیل 2×n با مستطیل‌های 2×1 و 2×2 بیابید و سپس یک الگوریتم بازگشتی بنویسید که با دریافت n تعداد راه‌های پرکردن یک مستطیل 2×n با مستطیل‌های 2×1 و 2×2 را محاسبه کند.

٤. یک ماتریس M با درایه‌های صفر و یک و ابعاد $2^n \times 2^n$ موجود است، S رشتهٔ متناظر با M را اینگونه محاسبه می‌کنیم. اگر کلیهٔ درایه‌های M صفر بود S=0 و اگر کلیهٔ درایه‌های M یک بود S=1 و در غیر این‌صورت ماتریس را به ٤ ماتریس M_1,M_2,M_3,M_4 مطابق شکل زیر تقسیم می‌کنیم. رشته های S_1,S_2,S_3,S_4 را متنـاظر با M_1,M_2,M_3,M_4 پیدا می‌کنیم، سپـس $S = S_1S_2S_3S_4$ قرار می‌دهیم:

$$\begin{bmatrix} M_1 & M_2 \\ M_3 & M_4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow S = 1001$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \Rightarrow S = 1$$

الگوریتمی بازگشتی بنویسید که M را از ورودی گرفته و S را تولید کند.

۵. یک جدول n×1 داریم می‌خواهیم خانه‌های این جدول را با n رنگ، رنگ کنیم به‌گونه‌ای که هیچ دو خانهٔ مجاور هم رنگ نباشند. یک الگوریتم بازگشتی برای محاسبهٔ تعداد راه‌های رنگ کردن جدول طراحی کنید.

٦. الگوریتمی بازگشتی برای محاسبه $\binom{n}{r}$ با توجه به رابطه‌های بازگشتی زیر بنویسید.

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

$$\binom{n}{1} = n \quad \& \quad \binom{n}{0} = 1$$

۷. الگوریتمی بازگشتی برای تولید تمام زیر مجموعه‌های یک مجموعهٔ n عضوی بیان کنید. فرض کنیداعضای مجموعهٔ n عضوی عبارت باشند از {1, 2 , 3 , ..., n} .

۸. الگوریتمی بازگشتی بنویسید که عدد n را از ورودی خوانده و به وسیلهٔ اعداد ۱ و۲ عدد n بسازد که‌بر 2^n بخشپذیر باشد.

۹. یک رابطهٔ بازگشتی برای تعداد رشته‌های n تایی متشکل از ۰ و ۱ که شامل دو صفر متوالی است را پیدا کنید و سپس یک الگوریتم بازگشتی بنویسید که با دریافت n تعداد رشته‌ها را محاسبه کند.

۱۰. یک الگوریتم بازگشتی برای محاسبهٔ $x^n \bmod m$ وقتی که m و x و n اعدادی مثبت هستند بیابید.

۱۱. یک رشته داده شده است. الگوریتمی بازگشتی برای پیدا کردن معکوس آن بیابید.

۱۲. تابع f (x) به‌صورت زیر تعریف می‌شود. الگوریتمی بازگشتی برای محاسبهٔ آن بیابید.

$$\begin{cases} f(x) = f(x-1) + f(x-2) + \ldots + f(x-10) & x > 10 \\ f(x) = f(x+1) + f(x+2) + \ldots + f(x+10) & x < -10 \\ f(x) = 1 & -10 < x < 10 \\ f(x) = -1000 & x = \pm 10 \end{cases}$$

۱۳. تابع f (x) به‌صطورت زیر تعریف می‌شود. الگوریتمی بازگشتی برای محاسبهٔ آن بیابید.

$$\begin{cases} f(x) = 3 \times f(x-3) + 4 \times f(x-5) + 10 & x > 10 \\ f(x) = -3 \times f(x+3) - 4 \times f(x+5) + 10 & x < -10 \\ f(x) = -10000 & -10 < x < 10 \\ f(x) = 0 & x = \pm 10 \end{cases}$$

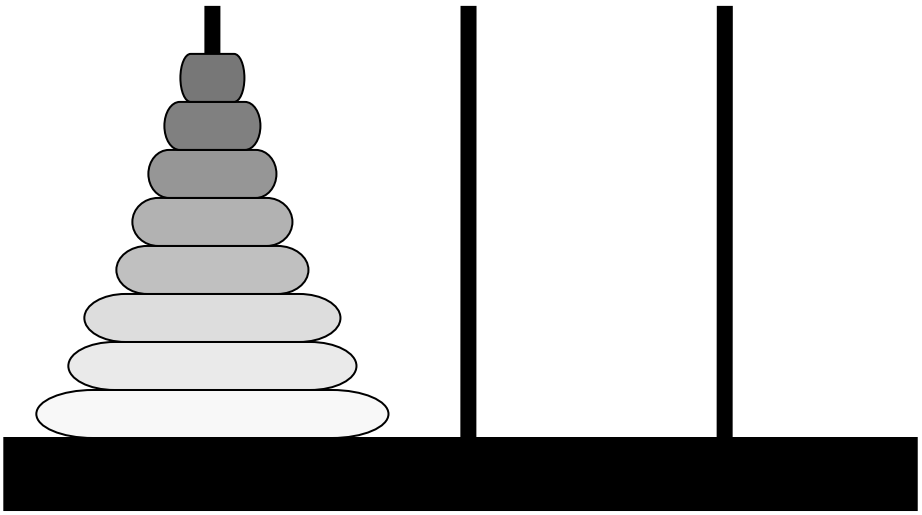
۱۴. تابعی بازگشتی به‌صورت power(base, exponent) بنویسید که با احضار آن، مقدار $base^{exponent}$ را به‌دست آورد. مثلاً power(3, 4) = 3* 3* 3* 3 . فرض کنید که exponent یک عدد صحیح بزرگتر یا مساوی با ۱ است. (راهنمایی: مرحلهٔ بازگشت از رابطهٔ زیر استفاده می‌کند:)

$$base^{exponent} = base * base^{exponent-1}$$

و شرط توقف هنگامی برقرار می‌شود که exponent برابر با ۱ باشد زیرا :

$$base^1 = base$$

۱۵. (برج های هانوی) هر متخصص مبتدی علوم کامپیوتری باید درگیر مسائل کلاسیک خاصی شده باشد. مسألهٔ برج‌های هانوی (شکل ۱۰-۲) یکی از معروف‌ترین مسائل کلاسیک است. طبق یک روایت افسانه‌ای، در معبدی در خاور دور، کشیشانی سعی‌دارند پشته‌ای از چند دیسک را از یک میله به میله‌ای دیگر منتقل کنند. پشتهٔ اولیه دارای ۶۴ دیسک است که این دیسک‌ها برروی یک میله قرار گرفته‌اند و از پایین به بالا به ترتیب نزولی اندازهٔ دیسک‌ها، چیده شده‌اند. کشیشان می‌خواهند این پشته را از این میله به میلهٔ دوم منتقل کنند با این شرط که اولاً در هر مرحله فقط یک دیسک را بتوان جابه‌جا کرد و ثانیاً در هیچ لحظه‌ای نباید دیسک بزرگتری بالای دیسک کوچک‌تری قرار گیرد. میلهٔ سومی نیز برای نگهداری موقت دیسک‌ها در اختیار است. گویند که هنگامی که کشیشان کار خود را به پایان برسانند پایان جهان نیز فرا خواهد رسید، بنابراین انگیزهٔ کمی در آسان کردن کار آنها وجود دارد.



شکل ۱۰-۲ : برج‌های هانوی برای ۸ دیسک

فرض کنید که کشیشان قصد دارند دیسک‌ها را از میلهٔ ۱ به میلهٔ ۳ منتقل کنند. می‌خواهیم الگوریتمی ارائه دهیم که دقیقاً ترتیب انتقال دیسک‌ها را از میله‌ای به میلهٔ دیگر را چاپ کند. اگر بخواهیم این مساله را با روش‌های متعارف حل کنیم خیلی زود از توانایی خود در ادارهٔ دیسک‌ها ناامید می‌شویم. در عوض اگر با دید بازگشتی به مسأله نگاه کنیم فوراً آن را قابل حل می‌یابیم. انتقال **n** دیسک را می‌توان برحسب انتقال **n−1** دیسک (بازگشتی) به‌صورت زیر در نظر گرفت:

الف- **n−1** دیسک را از میلهٔ ۱ به میلهٔ ۲ منتقل کن و از میلهٔ ۳ به‌عنوان محل موقت استفاده‌کن.

ب- آخرین (بزرگترین) دیسک را از میلهٔ ۱ به میلهٔ ۳ منتقل کن.

چ- **n−1** دیسک را از میلهٔ ۲ به میلهٔ ۳ منتقل کن و از میلهٔ۱ به‌عنوان محل موقت استفاده کن. این فرایند هنگامی خاتمه می‌یابد که تعداد دیسک‌هایی که باید منتقل شوند **n=1** شود (حالت پایه). این کار به سادگی با انتقال این دیسک و بدون نیاز به محل موقت انجام می‌پذیرد.

برنامه‌ای به زبان C++ بنویسید که مسألهٔ برج‌های هانوی را حل کند. از یک تابع بازگشتی با چهار پارامتر زیر استفاده کنید:

الف- تعداد دیسک‌هایی که باید انتقال یابند.

ب- میله‌ای که دیسک‌ها در ابتدا روی آن قرار گرفته‌اند.

ج- میله‌ای که این پشته از دیسک‌ها باید به آن منتقل شوند.

د- میله‌ای که باید به‌عنوان محل موقت مورد استفاده قرارگیرد.

برنامه باید دقیقاً دستورالعمل‌های لازم را برای انتقال دیسک‌ها از میلهٔ آغازین به میلهٔ مقصد چاپ کند.مثلاً برای انتقال پشته‌ای از سه دیسک از میلهٔ ۱ به میلهٔ ۳، برنامه باید دستورالعمل‌های زیر را چاپ کند:

⟹ 1 → 3 (یعنی بالاترین دیسک را از میلهٔ۱ به میلهٔ ۳ منتقل کن)
1 → 2
3 → 2
1 → 3
2 → 1
2 → 3
1 → 3

۱۶. بزرگترین مقسوم‌علیه مشترک اعداد صحیح x و y عبارت از بزرگترین عدد صحیحی است که هر دو عدد x و y بر آن قابل قسمت باشند. تابعی بازگشتی به نام gcd بنویسید که بزرگترین مقسوم‌علیه مشترک x و y را بازگرداند. بزرگترین مقسوم‌علیه مشترک x و y به‌طور بازگشتی به‌صورت زیر تعریف می‌شود: اگر y برابر با صفر باشد gcd(x, y) برابر با x است؛ در غیر این صورت gcd(x, y) برابر با gcd(y, x %y) است که در آن % عملگر باقی مانده تقسیم می‌باشد.

۱۷. آیا می‌توان تابع main را به‌طور بازگشتی فراخواند؟ برنامه‌ای حاوی یک تابع main بنویسید. متغیری محلی و استاتیک به نام count در آن تعریف کنید و مقدار اولیهٔ آن را برابر با ۱ قرار دهید. سپس هر بار که main فراخوانده می‌شود مقدار count را افزایش دهید و چاپ کنید. برنامهٔ خود را اجرا کنید. چه اتفاقی می‌افتد؟

۱۸. (چاپ آرایه) تابعی بازگشتی به نام printArray بنویسید که آرایه و اندازهٔ آن را به‌عنوان آرگومان بگیرد و بدون آنکه چیزی بازگرداند آن را چاپ کند. این تابع باید هنگام دریافت آرایه‌ای با اندازهٔ صفر، عملیات را متوقف کند و باز گردد.

۱۹. (چاپ یک رشته به‌طور معکوس) تابعی بازگشتی به نام stringReverse بنویسید که یک آرایهٔ کاراکتری، حاوی یک رشتهٔ زبان C را به‌عنوان آرگومان بگیرد، رشتهٔ مزبور را به‌طور معکوس چاپ کند و هیچ چیز بازنگرداند. این تابع باید هنگام مواجه شدن با کاراکتر نال، عملیات را متوقف کند و بازگردد.

۲۰. (یافتن کوچکترین مقدار در یک آرایه) تابعی بازگشتی به نام recursiveMinimum بنویسید که یک آرایهٔ صحیح و اندازهٔ آن را به‌عنوان آرگومان بگیرد و کوچکترین عنصر آن را بازگرداند. این تابع باید هنگام دریافت آرایه‌ای با ۱ عنصر، عملیات را متوقف کند و بازگردد.

۱۰-۵ : قضیهٔ اصلی (master theorem)

در این بخش به معرفی قضیهٔ اصلی می‌پردازیم که روش خوبی برای حل رابطه‌های بازگشتی به فرم زیر است:

$$T(n) = a\,T(n / b) + f(n)$$

که در آن $a \geq 1$ و $b>1$ ثوابت عددی هستند و $f(n)$ یک تابع مثبت است.

فرض کنید $T(n)$ نشان دهندهٔ پیچیدگی یک رابطهٔ بازگشتی باشد. قضیهٔ اصلی سه حالت دارد، که عبارت است از:

حالت ۱: اگر ثابت $\varepsilon > 0$ وجود داشته باشد به‌طوری که داشته باشیم:

$$f(n) = O(n^{\log_b^a - \varepsilon})$$

آنگاه رابطهٔ زیر برای $T(n)$ به‌دست خواهد آمد:

$$T(n) = O(n^{\log_b^a})$$

حالت ۲: اگر داشته باشیم:

$$f(n) = O(n^{\log_b^a})$$

آنگاه رابطهٔ زیر برای $T(n)$ به‌دست خواهد آمد:

$$T(n) = O(n^{\log_b^a} \lg n)$$

حالت ۳ : اگر ثابت $\varepsilon > 0$ وجود داشته باشد به‌طوری که داشته باشیم:

$$f(n) = \Omega(n^{\log_b^a + \varepsilon})$$

و اگر برای ثابت $c < 1$ و n های به اندازهٔ کافی بزرگ داشته باشیم $af(n / b) \leq cf(n)$ آنگاه داریم برای $T(n)$ رابطهٔ زیر را خواهیم داشت :

$$T(n) = \theta(f(n))$$

قبل از بیان کردن مثال‌های مربوط به این قضیه اجازه دهید به چند نکتهٔ مهم اشاره کنیم.

۱. در حالت اول باید $f(n)$ به میزان n^{ε} از $O(n^{\log_b^a})$ بزرگتر باشد. یعنی باید به‌صورت چند جمله‌ای از $f(n)$ بزرگتر باشد. به عبارت دیگر هنگامی که حاصل $O(n^{\log_b^a})$ را بر $f(n)$ تقسیم می‌کنیم باید حاصل، یک چند جمله‌ای با درجهٔ بزرگتر یا مساوی یک باشد. اگر این شرایط برقرار نباشد دیگر قضیهٔ اصلی قابل استفاده نیست، و به این حالات شکاف بین حالت ۱ و۲ گفته می‌شود.

۲. اگر شرایط مطرح شده در حالت ۳ نیز بر قرار نباشد، در این حالت نیز قضیهٔ اصلی قابل استفاده نیست، و به آن شکاف بین حالت ۲ و۳ گفته می‌شود.

مثال ۱۰–۱۰ : رابطهٔ بازگشتی $T(n) = 9T(n/3) + n$ را به‌واسطهٔ قضیهٔ اصلی حل کنید.

با توجه به رابطهٔ فوق و صورت قضیهٔ اصلی موارد زیر را داریم:

$$\begin{cases} a = 9 \\ b = 3 \\ f(n) = n \\ n^{\log_b^a} = n^{\log_3^9} = \theta(n^2) \end{cases}$$

حال اگر $\varepsilon=1$ انتخاب شود، حالت اول قضیه اصلی را داریم:

$$f(n) = O(n^{\log_3^9 - 1}) \quad \Rightarrow \quad T(n) = \theta(n^2)$$

مثال ۱۰–۱۱ : رابطهٔ بازگشتی $T(n) = T(2n/3) + 1$ را به‌واسطهٔ قضیهٔ اصلی حل کنید.

با توجه به رابطهٔ فوق و صورت قضیهٔ اصلی موارد زیر را داریم:

$$\begin{cases} a = 1 \\ b = \frac{3}{2} \\ f(n) = 1 \\ n^{\log_b^a} = n^{\log_{3/2}^1} = n^0 = 1 \end{cases}$$

با توجه به روابط فوق چون رابطهٔ $f(n) = n^{\log_b^a}$ برقرار است با حالت دوم قضیهٔ اصلی روبه‌رو هستیم، و داریم:

$$f(n) = \theta(n^{\log_b^a}) = \theta(1) \quad \Rightarrow \quad T(n) = \theta(\lg n)$$

مثال ۱۰–۱۲ : رابطهٔ بازگشتی $T(n) = 2T(n/2) + n\lg n$ را به‌واسطهٔ قضیهٔ اصلی حل کنید:

با توجه به رابطهٔ فوق و صورت قضیهٔ اصلی موارد زیر را داریم:

$$\begin{cases} a = 2 \\ b = 2 \\ f(n) = n \\ n^{\log_b^a} = n^{\log_2^2} = \theta(n) \end{cases}$$

شاید به‌نظر آید این مورد همان حالت سوم است، ولی چون $f(n) = n \lg n$ به‌صورت تقریبی از $n^{\log_2^2} = \theta(n)$ بیشتر است و نه به‌صورت چند جمله‌ای. نسبت $f(n) / n^{\log_b^a} = n \lg n / n = \lg n$ کمتر از n^{ε} است. پس در شکاف بین حالات۲و۳ می‌افتد و در نتیجه قضیهٔ اصلی را برای آن نمی‌توان به‌کار برد.

مثال ۱۰–۱۳ : الگوریتم زیر را با استفاده از قضیهٔ اصلی تحلیل کنید.

```
1.  int sample(int n)
2.  {
3.     if(n==1)
4.       return 2;
5.     for(i=1;i<=n;i++)
6.         a++;
7.     b = sample(n/2)+ sample(n/2);
8.      return b;
9.  }
```

ابتدا رابطهٔ بازگشتی این الگوریتم را می‌نویسیم. این رابطه عبارت است از:

$$T(n) = 2T(n/2) + \theta(n)$$

در این رابطه، عبارت $\Theta(n)$ هزینهٔ خطوط ۳ الی ٦ می‌باشد و عبارت $2\ T(n/2)$ نیز با توجه به دو فراخوانی تابع sample با ورودی $(n/2)$ در خط ۷ به‌دست آمده‌است. حال این رابطه را حل می‌کنیم. همان‌طور که مشخص است با حالت دوم قضیهٔ اصلی روبرو هستیم، زیرا:

$$\begin{cases} a = 2 \\ b = 2 \\ f(n) = n \\ n^{\log_b^a} = n^{\log_2^2} = \theta(n) \end{cases}$$

بنابراین با توجه با روابط فوق خواهیم داشت:

$$T(n) = \Theta(n \lg n)$$

مثال ۱۰–۱۴ : الگوریتم زیر را با استفاده از قضیهٔ اصلی تحلیل کنید.

```
1.  int sample(int n)
2.  {
3.     if(n==0)
4.        return 2;
5.    return  sample(n/2);
6.  }
```

ابتدا رابطهٔ بازگشتی این الگوریتم را می‌نویسیم:

$$T(n) = T(n / 2) + \Theta(1)$$

در این رابطه عبارت $\Theta(1)$ با توجه به خطوط ۳ و ٤ و عبارت $T(n / 2)$ با توجه به خط ۵ کد به‌دست آمده است. حال این رابطه را حل می‌کنیم. همان‌طور که مشخص است با حالت دوم قضیهٔ اصلی روبرو هستیم. زیرا:

$$\begin{cases} a = 1 \\ b = 2 \\ f(n) = 1 \\ n^{\log_b^a} = n^{\log_2^1} = \theta(1) \end{cases}$$

بنابراین با توجه با روابط فوق خواهیم داشت:

$$T(n) = \theta(\lg n)$$

۱۰-۶: تمرین

۱. به کمک قضیهٔ اصلی رابطه‌های زیر را حل کنید.

a. $T(n) = 4T(n / 2) + n$
b. $T(n) = 4T(n / 2) + n^2$
c. $T(n) = 4T(n / 2) + n^3$

۲. آیا قضیهٔ اصلی می‌تواند رابطهٔ زیر را حل کند.

$$T(n) = 4T(n / 2) + n^2 \lg n$$

۱۰–۷ : حل رابطه‌های بازگشتی

رابطه‌های بازگشتی خطی همگن

در بخش قبل با تعریف رابطه‌های بازگشتی آشنا شدیم. درزیر نمونه‌هایی از انواع مختلف رابطه‌های بازگشتی را مشاهده می‌کنید.

$T(n) = T(n-1) + 2$
$T(n) = T(n-1) + T(n-2)$
$T(n) = 2T(n-2) + T(n-4) + 12$
$T(n) = 4T(n-1)^2 + T(n-2)$
$T(n) = T(n-1)T(n-2) + 3$

در ابتدا به معرفی تعاریف مرتبه، خطی بودن و همگن بودن می‌پردازیم و سپس به سراغ حل رابطه‌های بازگشتی خطی همگن می‌رویم.

مرتبه رابطه بازگشتی (Order)

مرتبهٔ یک رابطه عبارت است از تفاضل بزرگترین و کوچکترین ورودی‌های تابع T در رابطهٔ موردنظر به‌عنوان مثال روابط زیر را در نظر بگیرید.

$T(n)=5T(n-1) + 2$ رابطهٔ ۱۰–۱۱
$T(n)=2T(n-1)+ 4T(n-2)$ رابطهٔ ۱۰–۱۲
$T(n)=T(n-1)$ رابطهٔ ۱۰–۱۳
$T(n)=T(n-4)+T(n-1)+2$ رابطهٔ ۱۰–۱۴

مرتبهٔ رابطهٔ ۱۰–۱۱ یک است، زیرا تفاضل n و n–1 یعنی بزرگترین و کوچکترین ورودی تابع T برابر یک است. مرتبهٔ رابطهٔ ۱۰–۱۲ برابر دو است زیرا تفاضل n و n–2 برابر دو است. به همین ترتیب مرتبهٔ رابطهٔ ۱۰–۱۳ برابر یک و مرتبهٔ رابطهٔ ۱۰–۱٤ برابر چهار محاسبه می‌شود.

رابطه‌های بازگشتی خطی[1]

یک رابطه‌های بازگشتی مرتبه K را *رابطهٔ بازگشتی خطی* گویند اگر $T(n), T(n-1), \ldots, T(n-k)$ خطی باشند. یعنی توان روابط $T(n), T(n-1), \ldots, T(n-k)$ بیشتر از یک نباشد و درهم نیز ضرب نشوند. مثلاً رابطهٔ زیر خطی است.

$T(n)=2T(n-1)+T(n-2)+3T(n-5)+100$

ولی رابطهٔ زیر به علت توان دوم $T(n-2)$ غیر خطی است.

$T(n) = T(n-1) + T(n-2)^2 + T(n-4) + 3$

1. linear recurrence

رابطهٔ زیر نیز به علت ضرب $T(n-1)$ در $T(n-2)$ غیر خطی است.

$$T(n) = 7T(n-1)T(n-2) + T(n-1)$$

حال یک رابطهٔ بازگشتی خطی مرتبه k را در نظر بگیرید. فرم زیر را خواهد داشت:

$$T(n) = c_1 \times T(n-1) + c_2 \times T(n-2) + \ldots + c_k \times T(n-k) + d$$

اگر d برابر صفر باشد رابطهٔ بالا را همگن می‌گویند و اگر مخالف صفر باشد آن را غیرهمگن می‌گویند.

حل یک رابطه بازگشتی خطی همگن مرتبه یک

حالت کلی زیر را با فرض شرایط اولیهٔ $T(0) = g$ در نظر بگیرید:

$$T(n)=cT(n-1)$$

به‌جای $T(n)$ مقدار x^n و به‌جای $T(n-1)$ مقدار x^{n-1} را می‌گذاریم و معادله را حل می‌کنیم. نتیجه به‌دست آمده به‌صورت زیر خواهد بود:

$$x = c \Rightarrow x^n = c^n \Rightarrow T(n) = c^n \times T(0) = c^n \times g$$

مثال ۱۰-۱۵ : رابطهٔ $T(n) = 3T(n-1)$ را با فرض $T(0) = 2$ حل کنید.

با توجه به توضیحات فوق حل این رابطه به‌صورت زیر است:

$$x = 3 \Rightarrow x^n = 3^n \Rightarrow T(n) = 3^n \times T(0) = 3^n \times 2$$

حل یک رابطهٔ بازگشتی خطی همگن مرتبه دو

حالت کلی زیر را با فرض شرایط اولیهٔ $T(0) = k_1$ و $T(1) = k_2$ در نظر بگیرید:

$$T(n) = c_1 \times T(n-1) + c_2 \times T(n-2)$$

به جای $T(n)$ مقدار x^n و به جای $T(n-1)$ مقدار x^{n-1} و به‌جای $T(n-2)$ مقدار x^{n-2} را می‌گذاریم و معادله را حل می‌کنیم. به این معادله، معادلهٔ مشخصهٔ رابطهٔ بازگشتی می‌گویند.

$$X^n = c_1 \times X^{n-1} + c_2 \times X^{n-2} \Rightarrow X^2 = c_1 \times X + c_2$$

فرض کنید معادلهٔ مشخصهٔ به‌دست آمده دارای دو ریشهٔ t_1 و t_2 باشد. بر اساس این دو ریشه می‌توان حالات زیر را برای حل رابطهٔ بازگشتی اولیه در نظر گرفت:

حالت اول $t_1 \neq t_2$

در این حالت جواب رابطهٔ بازگشتی به فرم زیر خواهد بود.

$$T(n) = c_1 t_1^n + c_2 t_2^n$$

برای به‌دست آوردن مقادیر c_1 و c_2 شرایط اولیه $T(0)$ و $T(1)$ را در رابطهٔ فوق جایگزین می‌کنیم. در نتیجه به یک دستگاه دو معـادله و دومجهول می‌رسیم، که با حل آن مقادیر c_1 و c_2 مشخص می‌شود.

حالت دوم $t_1 = t_2$

در این حالت جواب رابطهٔ بازگشتی به فرم زیر خواهد بود.

$$T(n) = c_1 t_1^n + c_2 n t_1^n$$

در این حالت نیز برای به‌دست آوردن مقادیر c_1 و c_2 شرایط اولیه $T(0)$ و $T(1)$ را در رابطهٔ فوق جایگزین می‌کنیم. در نتیجه دوباره به یک دستگاه دو معـادله و دومجهول خواهیم رسید، که با حل آن مقادیر c_1 و c_2 مشخص می‌شود.

مثال ۱۰–۱۶ : رابطهٔ بازگشتی $T(n) = -4T(n-1) + -3T(n-2)$ را حل کنید.

$$x^2 + 4x + 3 = 0 \Rightarrow t_1 = -3, t_2 = -1 \Rightarrow T(n) = c_1(-3)^n + c_2(-1)^n$$

مثال ۱۰–۱۷ : رابطه بازگشتی $T(n) = 6T(n-1) + -9T(n-2)$ را حل کنید.

$$x^2 - 6x + 9 = 0 \Rightarrow t_1 = 3 \ , \ t_2 = 3 \Rightarrow T(n) = c_1(3)^n + c_2 n(3)^n$$

حل رابطهٔ بازگشتی خطی همگن مرتبه K

برای حل این رابطه‌ها ابتدا معادلهٔ مشخصه آنها را پیدا کرده و سپس ریشه‌های معادلهٔ مشخصه را می‌یابیم.

برای روشن شدن مطلب به مثال‌های زیر توجه کنید.

1. $T(n) = 2T(n-1) + 6T(n-2) + T(n-3) \quad \Rightarrow \quad x^3 = 2x^2 + 6x^1 + 1$
2. $T(n) = T(n-2) + 4T(n-3) \quad \Rightarrow \quad x^3 = x^1 + 4$
3. $T(n) = c_1 \times T(n-1) + \ldots + c_k \times T(n-k) \quad \Rightarrow \quad x^k = c_1 \times x^{k-1} + c_2 \times x^{k-2} + \ldots + c_k$

حال معادلهٔ مشخصه را حل می‌کنیم تا ریشه‌های آن را بیابیم. اگر ریشه‌های t_1 تا t_k متمایز باشند یعنی:

$$t_1 \neq t_2 \neq t_3 \neq \ldots \neq t_k$$

آنگاه جواب رابطهٔ بازگشتی به فرم زیرخواهد بود:

$$T(n) = c_1(t_1)^n + c_2(t_2)^n + \ldots + c_k(t_k)^n$$

ولی اگر ریشهٔ تکراری داشتیم، فرض می‌کنیم پس از حل معادلهٔ مشخصه از مرتبه k به i تا ریشهٔ متمایز برسیم که تکرار ریشهٔ t_i ، k_i مرتبه باشد. و در نتیجه جواب رابطهٔ بازگشتی به فرم زیر خواهد بود که درآن $c_{i,j}$ ها ضریب هستند:

$$T(n) = \sum_{j=1}^{i} \left((c_{1,j} + c_{2,j} \times n + \ldots + c_{k_j,j} \times n^{k_j - 1}) \times t_j^n \right)$$

مثال ۱۰–۱۸ : رابطهٔ بازگشتی زیر را حل کنید.

$$T(n) + 7T(n-1) + 19T(n-2) + 25T(n-3) + 16T(n-4) + 4T(n-5) = 0$$

معادلهٔ مشخصهٔ این رابطه به فرم زیر خواهد بود:

$$x^5 + 7x^4 + 19x^3 + 25x^2 + 16x^1 + 4 = (x+1)^3(x+2)^2 = 0$$

بنابراین جواب رابطهٔ بازگشتی به‌صورت زیر خواهد بود:

$$T(n) = (c_1 + c_2n + c_3n^2)(-1)^n + (c_4 + c_5n)(-2)^n$$

۱۰–۸ : تمرین

۱. رابطه‌های بازگشتی زیر را با کمک روش حل رابطه‌های بازگشتی همگن و قضیهٔ اصلی حل کنید. مشخص کنید کدامیک از این روابط خطی و کدام یک همگن هستند.

a. $T(n) - 6T(n-1) - 7T(n-2) = 0$

b. $T(n) = T(9n / 10) + n$

c. $T(n) + 10T(n-1) + 25T(n-2) = 0$

d. $T(n) = 16T(n / 4) + n^2$

e. $T(n) - 6T(n-1) + 9T(n-2) - 4T(n-3) = 0$

f. $T(n) = 7T(n / 3) + n^2$

g. $T(n) = 7T(n / 2) + n^2$

h. $T(n) - 4T(n-1) + 8T(n-2) = 0$

i. $T(n) + 5T(n-1) + 12T(n-2) - 18T(n-3) = 0$

۲. n خط، یک صفحه را به چند ناحیه تقسیم می‌کنند؟ به‌طوری که هر جفت از خط‌ها متقاطع باشند، ولی بیش از ۲ خط در یک نقطهٔ مشترک همدیگر را قطع نکنند. یک معادلهٔ بازگشتی برای تعداد ناحیه‌ها برای n خط بنویسید و آن را حل کنید.

۳. رابطهٔ بازگشتی دنبالهٔ فیبوناچی را حل کنید.

۱۰–۹ : تکنیک تکرار در حل مسائل

اکنون زمان آن است تا با تکنیک‌های طراحی الگوریتم آشنا شویم. در طراحی الگوریتم آنچه مهم است حل زیاد مسأله است. ما نیز در این بخش سعی کردیم مثال‌های زیادی را برای شما مطرح کنیم. البته یادگیری این بخش بستگی به حل تمرین‌های این بخش دارد. در پایان ذکر این نکته ضروری است که الگوریتم‌هایی که ما در این بخش پیاده‌سازی کردیم همگی شبه کد هستند، با این وجود سعی کرده‌ایم تا جایی که ممکن است خواننده به زحمت نیفتد و شبه کدها را شبیه زبان C++ بیان کرده‌ایم. برای مثال می‌توان گفت اگرچه در عمل در یک آرایه A به طول n زیرنویس‌ها بین 0 تا n–1 تغییر می‌کنند، ولی ما در اینجا برای سادگی و جلوگیری از اشتباه درفهم الگوریتم‌هافرض می‌کنیم زیرنویس‌ها بین 1 تا n در آرایه ذخیره شوند.

استفاده از حلقه‌های تودرتو

بسیاری از مسأله‌ها هستند که احتیاج به هیچ تکنیک خاص برای حل ندارند. تنها به وسیلهٔ حلقه‌های تودرتومثل for و while به‌راحتی حل می‌شوند.

مثال ۱۰–۱۹ : مثالی از هندسهٔ محاسباتی[1] : هر پاره خط در کامپیوتر به‌وسیلهٔ دو نقطهٔ ابتدایی و انتهایی به‌طور یکتا در فضای دو بعدی مشخص می‌شود. فرض کنید به‌وسیلهٔ یک آرایه، مجموعه‌ای از n پاره خط به ما داده شده است. هدف نوشتن برنامه‌ای برای مشخص کردن نقاط تقاطع است. راه حل سادهٔ این مسأله تست دو به دوی خط‌ها با یکدیگر است. شبه کدِ این الگوریتم را می‌توانید در پایین مشاهده کنید.

```
void intersect()
{
      for(i=1; i<n; i++)
        for(j=i+1; j<=n; j++)
           if(line "i" have intersect with line "j")
                     printf(" i intersect j ");
}
```

تحلیل این الگوریتم را درپایین مشاهده کنید:

$$T(n) = \binom{n}{2} = \frac{n(n-1)}{2} = \theta(n^2)$$

از آنجایی که هر دو خط باید به‌طور جداگانه با یکدیگر تست شوند و از آنجایی که n خط داریم بنابراین با ترکیب 2 از n روبه رو هستیم. امروزه الگوریتم‌های با زمان $O(n \lg n + I)$ برای این مسأله پیدا شده است که در آن منظور از I تعداد نقاط تقاطع است.

1. Computational Geometry

مثال ۱۰–۲۰ : یک چند ضلعی به وسیلهٔ مختصات رأس‌هایش در فضای دو بعدی مشخص می‌شود دو چند ضلعی به‌وسیلهٔ آرایه‌ای به ما داده شده است. برنامه‌ای بنویسید که تعیین کند آیا این دو چندضلعی، اضلاع یکدیگر را قطع می‌کنند یا نه.

حل این مثال هم مانند مثال قبل با دو حلقهٔ **for** است. در این مثال تمام اضلاع چند ضلعی اول را با تمام اضلاع چند ضلعی دوم تست می‌کنیم.

مثال ۱۰–۲۱ : **n** نقطه در صفحهٔ دو بعدی داده شده است نزدیک‌ترین دو نقطه را در بین این **n** نقطه پیدا کنید.

می‌دانیم که فاصلهٔ دو نقطه در فضای دو بعدی از رابطه زیر به‌دست می‌آید.

$$\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

پس نقاط را به وسیلهٔ دو حلقهٔ **for** تودرتو، دو به دو مقایسه کرده و مینیمم را پیدا می‌کنیم.

مثال ۱۰–۲۲ : یک آرایه **n×n** از اعداد متفاوت داده شده است. اعداد در هر سطر و هر ستون به‌طور صعودی از کوچک به بزرگ مرتب شده‌اند. فردی ناشناس یک درایهٔ دلخواه که ما از محل آن بی‌خبر هستیم را انتخاب می‌کند وآن قدر از آن عدد کم می‌کند تا آن عدد ۱۰ واحد از عدد بالای آن و یا ۱۰ واحد از عدد سمت چپ آن کمتر شود. روشی برای یافتن عدد دستکاری شده بیابید.

در این مثال باید در همهٔ سطرها و همهٔ ستون‌ها اعداد مجاور را چک کنیم تا به عدد دستکاری شده برسیم. ممکن است در همان ابتدا عدد را بیابیم و ممکن است مجبور شویم کل آرایه را چک کنیم. پس با استفاده از حلقه‌های تودرتو این کار را به‌صورت زیر انجام می‌دهیم.

```
1. int Findchangednumber(int A[], int n)
2. {
3.        for(i=1 ;i<= n ; i++)
4.          for(j=1 ;j<n ;j++)
5.              if(A[i][j] > A[i][j+1])
6.                 return (A[i][j+1]);
7.        for(i=1 ;i<= n ; i++)
8.          for(j=1 ;j<n ;j++)
9.              if(A[j][i] > A[j+1][i])
10.                return (A[j+1][i]);
11. }
```

همان‌طور که در بالا گفتیم و در این مثال‌ها نیز دیدید، استفاده از تکنیک تکرار به کار خاصی نیاز ندارد فقط باید با یک یا چند حلقه تمام حالات مسأله را امتحان کرد. در پایان ذکر این نکته ضروری است که اگرچه حل مسأله با حلقه‌های تودرتو برای شما شاید ساده باشد و شما آن را به‌عنوان تکنیک حل مسأله قبول نکنید، ولی امروزه هنوز بسیاری از مسائل وجود دارد که راه‌حل بهینه‌ای ندارند و باید با حلقه‌های تودرتو تمام حالات را امتحان کرد. مثل برخی از مسائل **NP**– کامل که در درس طراحی الگوریتم‌ها با آنها آشنا خواهید شد.

۱۰-۱۰ : تمرین

۱. یک آرایه n×n از اعداد 0 و 1 داریم. می‌خواهیم از خانهٔ (1,1) به خانهٔ (n,n) برویم. در هر مرحله مجاز هستیم، از خانهٔ (i , j) به خانه‌های (i , j+1) و یا (i , j−1) و یا (i+1, j) و یا (i−1, j) بـرویم به شرطی که از آرایه خارج نشویم و به خانه‌ای که حاوی عدد صفر است وارد نشویم. برنامه‌ای طراحی کنید که در صورت وجـود چنین مسیری آن را بیابد. (راهنمایی: می‌توانید برای خودتان خانه‌ها را اندیس‌گذاری کنید. در ابتدا همان صفر و یک‌ها باقی بماند و سپس با استفاده از حلقهٔ while تا زمانی که ممکن است اعمال زیر را انجام دهید. اگر در خانهٔ (i, j) هستید و مقدار خانه یک است، اگر ممکن است به سمت بالا بروید و مقدار خانهٔ (i, j) را 2 کنید و گرنه خانهٔ (i, j) را 2 کنید. برای خانهٔ (i, j) عمل مربوط به مقدار 2 را انجام دهید. اگر مقدار خانهٔ (i, j) , 2 است اگر ممکن است به سمت راست بروید و مقدار خانه‌را 3 کنید و...)

۲. (ترسیم لاک پشتی) زبان لوگو که در بین کاربران کامپیوترهای شخصی از محبوبیت خاصی برخوردار است مفهوم ترسیم لاک پشتی را به خوبی رواج داده است. یک لاک پشت مکانیکی را در نظر بگیرید که تحت کنترل یک برنامهٔ C++ در اتاقی در گردش است. این لاک پشت، قلمی در اختیار دارد که در یکی از دو وضعیت بالا یا پایین قرار می‌گیرد. هنگامی که قلم پایین است لاک پشت در حین حرکت، شکلی را رسم می‌کند و هنگامی که قلم بالاست لاک‌پشت بدون رسم چیزی می‌تواند آزادانه حرکت کند. در این مسأله، باید عملیات لاک‌پشت، شبیه‌سازی و نیز یک تخته رسم کامپیوتری ساخته شود.

یک آرایهٔ ۲۰ در ۲۰ به نام floor بگیرد و مقدار اولیهٔ آن را برابر با صفر قرار دهید. فرمان‌ها را از آرایه‌ای بخوانید. محل جاری لاک‌پشت و بالا یا پایین بودن قلم را در هر لحظه دنبال کنید. فرض کنید که لاک‌پشت همواره از محل (0,0) صفحه و با قلم بالا شروع می‌کند. مجموعهٔ فرمان‌های برنامه به‌صورت زیر است:

فرمان	معنی
1	قلم با لا
2	قلم پایین
3	چرخش به راست
4	چرخش به چپ
5 , 10	حرکت به جلو به اندازهٔ۱۰ فاصله (یا عدد دیگری غیر از ۱۰)
6	چاپ آرایه ی ۲۰ در ۲۰
9	پایان داده‌ها(مقدار نگهبان)

فرض کنید که لاک‌پشت در نزدیکی مرکز صفحه قرار دارد. در این صورت برنامهٔ زیر یک مربع ۱۲ در ۱۲ رسم می‌کند و سپس قلم را بالا نگاه می‌دارد:

```
2
5 , 12
3
5 , 12
3
5 , 12
3
5 , 12
1
6
9
```

هنگامی که لاک‌پشت با قلم پایین حرکت می‌کند عناصر مناسب را در آرایهٔ **floor** یک کنید. با فرمان **6** (چاپ) هر جا که در آرایه ۱ وجود دارد یک ستاره (یا هر نویسهٔ دیگری که می‌خواهید) و هر جا صفر وجود دارد یک فاصلهٔ خالی چاپ کنید. برنامه‌ای بنویسید که قابلیت‌های بالا را برای ترسیم لاک پشتی پیاده‌سازی کند. چند برنامهٔ ترسیم لاک‌پشتی بنویسید و شکل‌های جالبی را رسم کنید. فرمان‌های دیگری نیز به برنامه اضافه کنید تا قدرت زبان ترسیم لاک پشتی را افزایش دهید. سپس مدت زمان اجرای برنامه را برای کشیدن یک مربع **n×n** تحلیل کنید.

۳. آرایه‌ای حاوی **n** عدد داریم. اعداد متفـاوت هستند به غیر از یک عدد که دوبار تکرار شده است.الگوریتمی از زمان **O(nlgn)** طراحی کنید که این عدد را پیدا کند.

٤. (مرتب کردن پیمانه‌ای) مرتب کردن پیمانه‌ای با استفاده از یک آرایهٔ دو اندیسی از اعداد صحیح شامل ردیف‌هایی با اندیس **0** تا **9** و ستون‌هایی با اندیس **0** تا **n−1** (که **n** تعداد اعدادی است که باید مرتب شوند) یک آرایهٔ تک اندیسی از اعداد صحیح مثبت را مرتب می‌کند. به هریک از ردیف‌های آرایهٔ دواندیسی، پیمانه گفته می‌شود. تابعی به نام **bucketsort** بنویسید که یک آرایهٔ صحیح و اندازهٔ آن را به‌عنوان آرگومان بگیرد و به‌صورت زیر عمل کند:

الف- هریک از مقادیرآرایهٔ تک اندیسی را براساس رقم یکان آن در ردیفی ازآرایهٔ پیمانه‌ای قراردهید. مثلاً عدد ۹۷ در ردیف ۷ ، عدد ۳ در ردیف ۳ و عدد ۱۰۰ در ردیف ۰ قرار داده می‌شوند. به این کار «مرحلهٔ توزیع» گویند.

ب- آرایهٔ پیمانه‌ای را به‌طور ردیفی بپیمایید و مقادیر فوق را به ترتیب به آرایهٔ اولیه برگردانید. به این کار «مرحلهٔ جمع‌آوری» گویند. ترتیب جدید مقادیر مثال بالا در آرایهٔ تک اندیسی به‌صورت ۱۰۰ ،۳ ، ۹۷ خواهد بود.

ج- این عملیات را برای ارقام بعدی (دهگان، صدگان، هزارگان و ...) نیز تکرار کنید.

در مرحله دوم، ۱۰۰ در ردیف ۰ ، ۳ در ردیف ۰ (چون عدد ۳ رقم دهگان ندارد) و ۹۷ در ردیف ۹ قرار می گیرد. پس از جمع‌آوری، ترتیب مقادیر آرایهٔ تک اندیسی به‌صورت ۱۰۰ ، ۳ ، ۹۷ خواهد بود. در مرحله سوم، ۱۰۰ در ردیف ۱، ۳ در ردیف۰ و ۹۷ در ردیف ۰ (پس از ۳) قرار می‌گیرد و پس از مرحله جمع‌آوری، آرایه به‌صورت مرتب شده خواهد بود. زمان لازم برای مرتب کردن **n** عدد **d** رقمی به‌وسیلهٔ این روش را پیدا کنید.

توجه کنید که اندازهٔ آرایهٔ دو اندیسی پیمانه‌ای، ده برابر اندازهٔ آرایهٔ صحیحی است که باید مرتب شود. این الگوریتم از مرتب کردن حبابی کارایی بهتری دارد اما به حافظهٔ بسیار بیشتری نیاز دارد. مرتب کردن حبابی فقط به یک عنصر اضافی نیاز داشت. این تمرین، مثالی از موازنهٔ بین فضا و زمان است. در این نسخه از مرتب کردن پیمانه‌ای، در هر مرحله باید داده‌ها به آرایهٔ اولیه برگردانده شوند. راه دیگر این است که یک آرایه پیمانه‌ای دواندیسی دیگر بگیریم و داده‌ها را مکرراً بین این دو آرایهٔ پیمانه‌ای جابه‌جا کنیم.

۵. هر مسأله‌ای که بتوان به‌صورت بازگشتی پیاده‌سازی کرد با تکرار نیز قابل پیاده‌سازی است، هر چند ممکن است آن راه‌حل دشوارتر و وضوح آن کمتر باشد. سعی کنید نسخه تکراری برج‌های هانوی را بنویسید. اگر موفق شدید، نسخهٔتکراری خود را با نسخهٔبازگشتی که در تمرین ۱۵ بخش ۱۰-٤ نوشتید مقایسه کنید. در این مقایسه ملاحظات کارایی، وضوح و توانایی شما در نمایش صحت عملکرد برنامه را، مد نظر قرار دهید.

٦. یک چندضلعی را محدب گویند اگرهر دو نقطهٔ دلخواه که درداخل چند ضلعی انتخاب کنیم پاره‌خطی‌که این دو نقطه را به هم وصل می‌کند و نقاط ابتدایی و انتهایی آن کاملاً در چند ضلعی قرار گیرد. دو چند ضلعی محـدب داده‌شده است. برنامه‌ای بنویسید که تعیین کند آیا یکی از این دوچندضلعی کاملاً در داخل دیگری قراردارد‌یا نه؟

۱۰-۱۱: روش تقسیم و حل

در روش تقسیم وحل[1]، که تا حدودی در فصل توابع با ایده‌های اینگونه حل مسائل آشنا شده‌اید، مسأله را به دو یا چند زیر مسألهٔ *مستقل* تقسیم می‌کنند. سپس هر زیر مسأله را به‌طور جداگانه حل می‌کنند و در پایان نتایج را ترکیب می‌کنند تا پاسخ مسأله اصلی را بیابند. در حقیقت هنر این روش به خاطر حل زیرمسأله‌ها به جای مسألهٔ اصلی است. به مثال زیر توجه کنید.

مثال ۱۰-۲۳ : یک آرایهٔ n-تایی از اعداد را داریم و می‌خواهیم ماکزیمم آنها را بیابیم. ابتدا به‌وسیلهٔ روش تکرارکه در بخش۱۰-۹ معرفی شد یک راه‌حل برای این مسأله پیدا می‌کنیم. سپس به‌وسیلهٔ تکنیک تقسیم و حل روش دیگری برای حل بیان می‌کنیم.

راه حل ساده: فرض کنید اعداد در آرایهٔ S با اندیسی در محدودهٔ x تا y هستند.

```
1.  int max(int s[], int x, int y)
2.  {
3.      max=s[x];
4.      for(i=x+1; i<=y; i++)
5.        if(max<s[i])
6.            max=s[i];
7.      return max;
8.  }
```

حال به راه‌حلی که از شیوهٔ تقسیم و حل در آن استفاده شده است توجه کنید.

```
1.  int max(int s[], int x, int y)
2.  {
3.      if(y-x<=1){
4.           if(s[x]<s[y])
5.              return s[y];
6.           else
7.              return s[x];
8.       }
9.      int max1, max2;
10.     max1 = max(s, x, (x+y)/2) );
11.     max2 = max(s, (x+y)/2+1, y);
12.      if(max1 < max2 )
13.          return max2;
14.      else
15.          return max1;
16. }
```

شاید در نگاه اول روش دوم پچیده به نظر آید. ولی روش دوم هیچ پیچیدگی ندارد، فقط آرایه را به دو بخش تقسیم می‌کند، و در هر بخش مقدار ماکزیمم را پیدا می‌کند و سپس ماکزیمم های به‌دست آمده را با هم مقایسه می‌کند تا ماکزیمم اصلی را بیابد.

---

1. Divide and Conquer

در مثال‌های بعد می‌بینید که روش تقسیم و حل چقدر بر سرعت برنامه اضافه می‌کند. در مثال بعد هدف نوشتن برنامه‌ای است که در یک آرایه ماکزیمم ومینیمم را پیدا کند.

مثال ۱۰–۲۴ : به‌کارگیری روش تکرار برای یافتن ماکزیمم و مینیمم یک آرایه.

```
1.  int max(int s[],int x,int y)
2.  {
3.        int max=s[x];
4.        int min=s[x];
5.        for(i=x+1;i<=y;i++)
6.          if(max < s[i])
7.            max = s[i];
8.        for(i=x+1;i<=y;i++)
9.          if(min > s[i])
10.           min = s[i];
11. }
```

همان‌طور که می‌بینید برنامهٔ بالا **2n** مقایسه انجام می‌دهد، **n** مقایسه برای یافتن ماکزیمم و **n** مقایسه برای یافتن مینیمم. ولی ما با استفاده از تکنیک تقسیم و حل با $3\lfloor n/2 \rfloor$ مقایسه مسأله را حل می‌کنیم. برنامه به این صورت است که در آن اعداد را به مجموعه‌های 2 تایی تقسیم‌بندی می‌کنیم تا به حداکثر $\lfloor n/2 \rfloor$ تا مجموعهٔ 2 تایی برسیم. در اولین مجموعهٔ 2 تایی با یک مقایسه ماکزیمم و مینیمم را می‌یابیم. در مراحل بعد هر دفعه با یک مقایسه ماکزیمم و مینیمم مجموعهٔ 2 تایی را می‌یابیم و با ماکزیمم و مینیمم فعلی مقایسه می‌کنیم. پس در کل در هر مرحله سه مقایسه صورت می‌گیرد. در پایان اگر تعداد اعداد فرد بود، یک عدد مقایسه نشده می‌ماند، آن را با ماکزیمم و مینیمم مقایسه می‌کنیم و به جواب نهایی می‌رسیم. چون در دور اول فقط یک مقایسه انجام دادیم در بقیه موارد حداکثر سه مقایسه و در پایان دو مقایسه انجام می‌دهیم، پس در کل حداکثر $3\lfloor n/2 \rfloor$ مقایسه انجام داده‌ایم.

مثالی از محاسبهٔ توان یک عدد را نیز در مثال زیر می‌بینید، که روش تقسیم و حل چقدر بر سرعت حل مسأله می‌افزاید.

مثال ۱۰–۲۵ : فرض کنید عدد **a** و عدد مثبت **n** داده شده است و می‌خواهیم مقدار a^n را محاسبه کنیم. یک روش عادی و بسیار کند استفاده از یک حلقهٔ **for** است. شبه کد این روش را در پایین ملاحظه می‌کنید. زمان مورد نیاز این برنامه از **O(n)** است.

```
1.  int Slowpower(int a, int n)
2.  {
3.      int x=a;
4.        for(i=2; i<=n; i++)
5.            x = x*a;
6.      return x;
7.  }
```

حال با تکنیک تقسیم و حل روشی ارائه می‌دهیم که از زمان **O(lgn)** باشد.

فرض کنید که **n** توانی از دو باشد. می‌توانیم به جای محاسبهٔ a^n ، $a^{n/2}$ را محاسبه کنیم و حاصل را در خودش ضرب کنیم. پس کار ما حدوداً نصف می‌شود. حال باز به جای محاسبهٔ $a^{n/2}$ مقدار $a^{n/4}$ را محاسبه می‌کنیم و این روند نصف کردن را تا رسیدن به مقدار یک ادامه می‌دهیم. تا اینجا در مورد **n** هایی که توانی از دو بودند مسأله را حل کردیم، حال فرض کنید **n** توانی از دو نباشد تا جایی که توان مورد محاسبه زوج است عمل تقسیم را ادامه می‌دهیم تا به عددی فرد مثل **p** برسیم. در این حالت نیز عدد را نصف کرده سپس جز صحیح آن را محاسبه می‌کنیم. پس به‌جای محاسبه $a^{p/2}$ مقدار $a^{\lfloor p/2 \rfloor}$ را محاسبه می‌کنیم، آن را در خودش و سپس در **a** ضرب می‌کنیم. پس برای هر **n** این روش جواب میدهد. به پیاده‌سازی این روش توجه کنید.

```
1.  int Fastpower(int a, int n)
2.  {
3.       int x;
4.       if(n==1)
5.          return a;
6.       else
7.          x = Fastpower(a,n/2);
8.      if(n%2==0)                  //n is even
9.          return x*x;
10.     else                        //n is odd
11.         return x*x*a;
12. }
```

در مورد تحلیل این الگوریتم باید گفت رابطهٔ بازگشتی این الگوریتم عبارت است از:

$$T(n)=T(n/2)+O(1)$$

اگر این رابطه را با توجه به حالت دوم قضیهٔ اصلی حل کنید به نتیجهٔ زیر می‌رسید:

$$T(n)=O(\lg n)$$

فرض کنید بخواهیم $2^{1,000,000,000}$ را محاسبه کنیم، در روش اول احتیاج به ۱.۰۰۰.۰۰۰.۰۰۰ محاسبه است ولی در روش دوم به حدوداً ۳۰ محاسبه احتیاج داریم.

مثال ۱۰–۲۶ : فرض کنید یک فرد، عددی بین **1** تا **n** را در ذهن دارد. شما در هر مرحله مجاز هستید سؤالاتی بپرسید که فرد در جواب آری یا خیر بگوید. برنامه‌ای طراحی کنید که:

۱. برنامه‌ای که با حداکثر **n** تا سؤال جواب را پیدا کند.

۲. برنامه‌ای که با حداکثر **lg n** تا سؤال جواب را پیدا کند.

حل قسمت اول مسئله بسیار راحت است، زیرا با یک حلقه تمام اعداد در بازه **1** تا **n** را می‌پرسیم تا به جواب برسیم. اما برای حل قسمت دوم احتیاج به کمی تکنیک دارد. برای این منظور از این روش استفاده می‌کنیم که در هر مرحله عدد وسط بازه را پیدا کرده و از فرد می‌پرسیم، آیا عدد موردنظر بزرگتر یا مساوی این عدد است یا نه. این عمل تا جایی ادامه پیدا می‌کند که طول بازه برابر یک شود.

در این حالت آخرین عدد پرسش شده، همان عدد موردنظر کاربر است. با توجه به آنکه در هر دفعه بازه نصف می‌شود، برای تحلیل این روش رابطهٔ زیر را داریم:

$$T(n) = T(n / 2) + O(1)$$

که طبق حالت دوم قضیهٔ اصلی، جواب این رابطه **O(lgn)** است.

شبه کد این روش را در زیر ملاحظه می‌کنید.

```
1.  int findnumber(int start, int end)
2.  {
3.      if(start == end )
4.         return start;
5.      int ave = (start+end)/2;
6.      if(number<=ave)
7.          return findnumber(start,ave);
8.      else
9.          return    findnumber(ave,end);
10. }
```

۱۰-۱۲: تمرین

۱. دومین بزرگترین عدد، عددی است که اگر اعداد را به‌طور نزولی از بزرگ به کوچک مرتب کنیم‌در مکان دوم قرار می‌گیرد. با استفاده از روش تقسیم و حل الگوریتمی بیان کنید که با حداکثر $n + \lceil \lg n \rceil - 2$ مقایسه آن را پیدا کند.

۲. نشان دهید حداکثر $\lceil 3n/2 \rceil - 2$ مقایسه برای یافتن ماکزیمم و مینیمم کافی است.

۳. فرض کنید دو عدد ۱۰۰۰ رقمی را می‌خواهیم در هم ضرب کنیم. با استفاده از روش تقسیم و حل‌الگوریتمی برای این منظور بیان کنید.

٤. یک روش مرتب‌سازی بر پایه تقسیم و حل به شکل زیر است.

```
1.  void Merge_sort(int A[], int p, int r)
2.  {
3.     int q;
4.     if(p<r)
5.       q =(p+r)/2;
6.     Merge_sort(A,p,q);
7.     Merge_sort(A,q+1,r);
8.     Merge(A,p,q,r);
9.  }
```

تابع **Merge** درشبه کد بالا دو آرایهٔ مرتب شده را از ورودی می‌گیرد و به یک آرایهٔ مرتب شده تبدیل می‌کند. تابع **Merge** را بنویسید. تابع شما باید از **O(n)** باشد. سپس تابع **Merge_sort** را به وسیلهٔ قضیه اصلی تحلیل کنید. در انتها کاربرد تکنیک تقسیم و حل را در این روش مرتب‌سازی شرح دهید.

۵. **n** تا گلوله داریم. یکی از این گلوله‌ها تقلبی واز بقیه سبک‌تر است. یک ترازوی دوکفه‌ای داریم به وسیلهٔ حداکثر $\lceil \log_3 n \rceil$ مقایسه گلولهٔ تقلبی را پیدا کنید.

٦. دو رشتهٔ مرتب به طول **m** و **n** داده شده است اگر دو رشته را ترکیب کنیم به یک رشتهٔ مرتب‌شدهٔ **(m+n)** عضوی می‌رسیم. هدف پیدا کردن **k** - اُمین عدد در رشتهٔ **(m+n)** عضوی است. برنامه‌ای بنویسیدکه این کار را در **O(lg(max(m, n)))** انجام دهد.

۷. مسألهٔ در متن درس به وسیلهٔ تکنیک تکرار حل شده است. در اینجا آن را با تکنیک تقسیم و حل حل کنید. **n** نقطه درصفحهٔ دو بعدی داده شده است. نزدیک‌ترین دو نقطه را دربین این **n** نقطه پیداکنید. الگوریتم شما باید از زمان **O(nlgn)**باشد.

راهنمایی: مجموعهٔ نقاط را به دو دسته تقسیم کنید و از اصل لانهٔ کبوتری هم کمک بگیرید.

۱۰–۱۳: برنامه نویسی پویا

برنامه‌نویسی پـویا[1] مانند روش تقـسیم و حل، مـسائل را از طریق تـرکیب جـواب زیر مسأله‌ها حل می‌کنـد (منظـور از برنامه‌نویسی در عنوان برنامه‌نویسی پویـا بهره‌گیری از تکنـیک استفاده از آرایه است نه نوشن کد). همان‌طور که می‌دانید در روش تقسیم و حل، مسأله را به چند زیر مسألهٔ مستقل تقسیم می‌کنیم وسپس زیر مسأله‌ها را به‌طور بازگشتی حل کرده و در نهایت جواب زیر مسأله‌ها را ترکیب می‌کنیم تا به حل مسألهٔ اصلی دست یابیم. در مقابل در روش برنامه‌نویسی پویا زیر مسأله‌ها مستقل نیستند. در روش تقسیم و حل همهٔ زیر مسأله‌ها به‌طور جداگانه حل می‌شوند، ولی در برنامه‌نویسی پویا ما یک زیر مسأله را حل کرده و جواب را در آرایه‌ای نگهداری می‌کنیم تا دیگر، احتیاجی به حل زیر مسأله‌های مشابه نداشته باشیم، و از مقادیر محاسبه شدهٔ قبلی استفاده کنیم.

برنامه‌نویسی پویا معمولاً در مسائل بهینه‌سازی ظاهر می‌شود. در اینگونه مسائل جواب‌های زیادی وجود دارد اما جواب بهینه موردنظر است، به عبارت دیگر همواره به دنبال ماکزیمم و یا مینیمم کردن یک مقدار هستیم. در برنامه‌نویسی پویا ۴ گام اساسی وجود دارد که عبارت هستند از:

۱. مشخص کردن ویژگی‌های جواب بهینه.
۲. به شیوهٔ بازگشتی جواب بهینه را تعریف می‌کنیم.
۳. جواب بهینهٔ مسأله را به شیوهٔ پایین به بالا پیدا می‌کنیم.
٤ . ساختار جواب بهینه را با اطلاعات محاسبه شدهٔ مراحل قبلی آن به‌دست می‌آوریم.

مراحل ۱ تا ۳ جزء مراحل اساسی برنامه‌نویسی پویا است ولی مرحلهٔ ۴ را وقتی فقط جواب بهینه را می‌خواهیم، می‌توان در نظر نگرفت. مرحله۴ برای به‌دست آوردن شیوهٔ رسیدن به جواب بهینه است. حال با تعدادی مثال برنامه‌نویسی پویا را به‌طور کامل معرفی می‌کنیم.

مثال ۱۰–۲۷ : (کمترین هزینه):آرایهٔ A به ابعاد $n\times 2$ داریم که درخانه‌های این آرایه اعدادی نوشته شده است. می‌خواهیم مسیری را از ستون اول به ستون n–اُم پیدا کنیم، یعنی مسیری از خانهٔ (1, 1) و یا (2, 1) به خانهٔ (1, n)ویا (2, n). درهر مرحله مجاز هستیم از خانهٔ (1, k) به خانهٔ (1, k+1) و یا خانهٔ (2, k+1) برویم و از خانهٔ (2, k) به خانهٔ (1, k+1) و یا خانهٔ (2, k+1) برویم. به هرخانه که وارد می‌شویم باید به اندازهٔ عدد نوشته شده در آن خانه هزینه بپردازیم. مسأله پیدا کردن مسیری با کمترین هزینه است.

یک راه حل برای این مسأله امتحان کردن تمام راه‌های ممکن و پیدا نمودن مسیر بهینه از بین تمامی راه‌های یافته شده است. چه تعداد راه برای امتحان وجود دارد؟ دو حالت برای ستون اول و دو حالت برای ستون دوم و دو حالت برای ستون ... و دو حالت برای ستون آخر، پس طبق اصل ضرب 2^n مسیر متفاوت داریم. پس امتحان کل مسیرها هزینه‌یی برابر $O(2^n)$ دارد که این هزینه بسیار زیاد است.

---

1. Daynamic Programming

حال با روش برنامه‌نویسی پویا راه‌حلی با هزینهٔ **O(n)** ارائه‌می‌دهیم. بنابراین مطابق چهار مرحله‌ای که پیشتر مطرح کردیم به ارائهٔ راه‌حل می‌پردازیم.

مرحلهٔ اول باید ویژگی‌های جواب بهینه را پیدا کنیم. بدون کاستن از کلیت مسأله فرض کنید **P** یک جواب بهینهٔ مسأله از ستون اول به ستون آخر باشد و از خانهٔ **(1, k)** عبور کند. مسیر **P** را به دومسیر P_1 و P_2 تقسیم می‌کنیم که P_1 مسیر بین ستون اول و خانهٔ **(1,k)** است و P_2 مسیر بین خانهٔ **(1,k)** و ستون آخر است. با کمی دقت متوجه می‌شویم که مسیرهای P_1 و P_2 باید بهینه باشند. فرض کنید مسیر P_1 بهینه نباشد و هزینهٔ مسیر P_1 برابر **r** باشد. حال مسیر بهینه بین ستون اول و خانهٔ **(1,k)** را می‌یابیم و آن را P_3 می‌نامیم. فرض کنید هزینهٔ مسیر P_3 برابر $r-\varepsilon$ باشد که ε عددی مثبت است. حال مسیر **Q** را از ترکیب مسیر P_3 و مسیر P_2 می‌سازیم. می‌بینیم که هزینهٔ مسیر **Q** به اندازهٔ ε از مسیر **P** کمتر است و این مورد خلاف فرض ما است که گفتیم مسیر **P** بهینه است. پس نتیجه می‌گیریم مسیرهای P_1 و P_2 بهینه هستند.

خوب پس به ساختار بهینه مسأله یعنی بهینه بودن زیر مسیرها پی بردیم. به سراغ مرحلهٔ دوم می‌رویم و به شیوهٔ بازگشتی جواب بهینه را تعریف می‌کنیم. جواب بهینه در نهایت برای رسیدن به ستون آخر یا به خانهٔ **(1, n)** و یا به خانهٔ **(2, n)** می‌آید. پس جواب بهینه عبارت است از مسیر بهینه برای رسیدن از ستون اول به ستون **(n−1)** -اُم بعلاوه یکی از دو خانهٔ **(1, n)** ویا **(2, n)** که مقدار کمتری دارد، یعنی اگر P_{n-1} مسیر بهینه بین ستون اول و ستون **(n−1)** -اُم باشد مسیر بهینهٔ **P** بین ستون اول و آخر با رابطهٔ زیر به‌دست می‌آید:

$$P = P_{n-1} + \min \{A[1, n], A[2, n]\}$$

حال به سراغ مرحله سوم می‌رویم و جواب بهینهٔ مسأله را به شیوهٔ پایین به بالا می‌یابیم. منظوراز شیوهٔ پایین به بالا این است که، ابتدا مسیر بهینهٔ به ستون اول و سپس مسیر بهینهٔ به ستون دوم و بعد ستون سوم و... در نهایت مسیر بهینه به ستون آخر را می‌یابیم. درهر مرحله با توجه به رابطهٔ بازگشتی که در مرحله دوم یافتیم و جواب بهینه مرحلهٔ قبل، با یک مقایسه جواب بهینهٔ آن مرحله رامی‌یابیم. در کل **n** مرحله انجام می‌دهیم، از ستون اول تا ستون آخر، و در هر مرحله فقط یک مقایسه.پس هزینهٔ این روش از **O(n)** است. به پیاده‌سازی این روش توجه کنید.

```
1.  int optimalvalue(int A[2][n])
2.  {
3.      int value =0;
4.        for(i=1; i<=n; i++)
5.          if(A[1][i]<A[2][i])
6.            value = A[1][i]+value;
7.          else
8.            value = A[2][i]+value;
9.      return value;
10. }
```

اگر مسأله فقط یافتن مقدار مسیر بهینه باشد، مسأله تمام است و احتیاج به مرحلهٔ ٤ نیست. ولی اگر مسیر بهینه را نیز بخواهیم، باید مرحلهٔ ٤ را نیز انجام دهیم.

در مرحله چهار ساختار جواب بهینه را با اطلاعات محاسبه شده به‌دست می‌آوریم. برای این منظور از آرایهٔ کمکی **B** با ابعاد **1×n** استفاده می‌کنیم و در هر مرحلهٔ از الگوریتم بالا اگرخانهٔ **(1, k)** را انتخاب کردیم در خانهٔ **k** -اُم **B** عدد **1** را می‌گذاریم و اگرخانهٔ **(2, k)** را انتخاب کردیم در خانهٔ **k**- اُم **B** عدد **2** را می‌گذاریم. پس کد بالا را به‌صورت زیر تغییر می‌دهیم.

```
1. int optimalvalue(int A[2][n])
2. {
3.     int value =0;
4.     for(i=1; i<=n; i++)
5.       if(A[1][i]<A[2][i])
6.        {
7.           value =A[1][i]+value;
8.           B[i]=1;
9.        }
10.      else
11.       {
12.          value =A[2][i]+value;
13.          B[i]=2;
14.       }
15.   return value;
16. }
17. void printway(int B[n])
18. {
19.     for(i=1; i<=n; i++)
20.       cout<<i<<"→"<<B[i]<<endl;
21. }
```

مثال ۱۰-۲۸ : (یافتن بزرگترین زیر رشته مشترک): بسیاری از مواقع در علم زیست‌شناسی دو یا چند رشتهٔ DNA اُرگان‌های مختلف بدن را با هم مقایسه می‌کنند.یک رشته DNA معمولاً از٤ نوع عنصر تشکیل می‌شود که عبارت هستند از adenine ، cytosine ،guanine وthymine که به اختصار با حروف اولشان آنها را مشخص می‌کنند. به‌عنوان مثال رشتهٔ DNA یک اُرگان ممکن است به‌صورت { S={A,C,C,C,G,A,T,T,T,G,C,C باشد.

در این‌مسئله هدف مشخص کردن بزرگترین زیررشتهٔ مشترک دو رشته است. فرض کنید $X=\{x_1, x_2, \ldots, x_m\}$ یک رشته باشد، آنگاه $Z=\{z_1, z_2, \ldots, z_k\}$ یک زیر رشتهٔ X است اگر و تنها اگر، یک دنبالهٔ اکیداً صعودی $\{i_1, i_2, \ldots, i_k\}$ از اندیس‌های X وجود داشته باشد که برای هر j که بین 1 تا k است داشته باشیم:

$$x_{i_j} = z_j$$

به‌عنوان مثال Z={ B, C, D, B} یک زیر رشته X={ A, B, C, B, D, A, B } است که متناظر با اندیس‌های { 2, 3, 5, 7 } از رشتهٔ X است ولی W={B, D, D, D, D} یک زیر رشتهٔ X نیست.

فرض کنید دو رشتهٔ X و Y داده شده است. گوییم Z یک زیر رشتهٔ مشترک X و Y است اگر Z زیر رشتهٔ X و همچنین زیر رشتهٔ Y باشد. برای عنوان مثال اگر X={A, B, C, B, D, A, B} و Y={B, D, C, A, B, A } باشد، آنگاه رشتهٔ { B,C,A } یک زیر رشتهٔ مشترک X و Y است ولی بزرگترین زیر رشتهٔ مشترک نیست، زیرا رشتهٔ { B, C, B, A} بزرگترین زیر رشتهٔ مشترک X و Y است.

دو رشتهٔ $X=\{x_1, x_2, \ldots, x_m\}$ و $Y=\{y_1, y_2, \ldots, y_n\}$ داده شده‌اند. حال می‌خواهیم بزرگترین زیر رشتهٔ مشترک این دو رشته را بیابیم. یک روش غیر بهینه بررسی تمام زیر رشته‌های X است تا ببینیم کدام یک از این زیررشته‌ها، زیر رشتهٔ Y هم است. و در پایان باید طولانی‌ترین زیررشته را از بین زیر رشته‌های مشترک مشخص کنیم. چون رشتهٔ X از m عضو تشکیل شده است پس 2^m تا زیررشته دارد در نتیجه باید 2^m تا زیررشته‌را امتحان کرد که این یک زمان نمایی است. بنابراین این روش غیر عملی است. در ادامه این مسأله را به‌روش برنامه‌نویسی پویا در $O(m\times n)$ حل می‌کنیم. برای این منظور باید مراحل چهارگانهٔ مربوط به حل مسائل برنامه‌نویسی پویا را برای مسئله انجام دهیم.

برای مشخص کردن ویژگی‌های جواب بهینه ابتدا یک تعریف را بیان می‌کنیم. اگر رشتهٔ X به‌صورت زیر باشد:

$X=\{x_1, x_2, \ldots, x_m\}$

I- اُمین پیشوند X را به‌صورت $X_I = \{ x_1, x_2, \ldots, x_I \}$ تعریف می‌کنیم.

به‌عنوان مثال اگر X={ A, B, C, B, D, A, B} باشد آنگاه $X_4 = \{A, B, C, B\}$ و $X_2 = \{A, B\}$ است.

لم بزرگترین زیر رشته مشترک

فرض کنید دو رشتهٔ $X=\{x_1, \ldots, x_m\}$ و $Y=\{y_1, \ldots, y_n\}$ داده شده‌اند و رشتهٔ $Z=\{z_1, \ldots, z_k\}$ بزرگترین زیر رشتهٔ مشترک X و Y است. آنگاه یکی از سه حالت زیر دو مورد Z امکانپذیر است:

حالت ۱ : اگر $x_m=y_n$ باشد آنگاه $z_k = y_n = x_m$ و Z_{k-1} بزرگترین زیر رشتهٔ مشترک X_{m-1} و Y_{n-1} است.

حالت۲ : اگر $x_m \neq y_n$ باشد و $z_k \neq x_m$ آنگاه Z بزرگترین زیر رشتهٔ مشترک X_{m-1} و Y است.

حالت۳ : اگر $x_m \neq y_n$ باشد و $z_k \neq y_n$ آنگاه Z بزرگترین زیر رشتهٔ مشترک Y_{n-1} و X است.

اثبات این لم به کمک برهان خلف بسیار آسان است، لذا اثبات آن را به عهدهٔ خوانندگان می‌گذاریم.

در مرحلهٔ دوم به شیوهٔ بازگشتی جواب بهینه را تعریف می‌کنیم. با توجه به لم مطرح شده در بالا به‌راحتی جواب را به‌صورت بازگشتی تعریف می‌کنیم.

فرض کنید C یک آرایهٔ دو بعدی با ابعاد m×n باشد و c[i][j] طول بزرگترین زیر رشتهٔ مشترک X_i و Y_j باشد. اگر i و یا j برابر صفر باشد مقدار c[i][j] برابر صفر است. بنابراین رابطهٔ بازگشتی به‌صورت زیر به‌دست می‌آید.

$$c[i][j]=\begin{cases} 0 & i=0\ ,\ j=0 \\ c[i-1][j-1]+1 & i,j>0\ \&\ x_i=y_j \\ \max\{c[i-1][j],c[i][j-1]\} & i,j>0\ \&\ x_i\neq y_j \end{cases}$$

در مرحلهٔ سوم جواب بهینهٔ مسأله را به شیوهٔ پایین به بالا پیدا می‌کنیم. با توجه به رابطهٔ بازگشتی مرحلهٔ قبل آرایهٔ C را پر می‌کنیم. چون هر خانه در مدت زمان O(1) پر می‌شود و درکل m×n خانه داریم، پسس در زمان O(m×n) به جواب می‌رسیم. جواب نهایی در خانهٔ C[m][n] خواهد بود. شبه کد این روش را در ادامه مشاهده می‌کنید.

```
1.  void LCS_length(String X,String Y)
2.  {
3.      for(i=1; i<=m; i++)
4.         c[i][0]=0;
5.      for(i=1; i<=n; i++)
6.         c[0][i]=0;
7.      for(i=1;i<=m;i++)
8.       for(j=1;j<=n;j++)
9.         if(xi=yi)
10.          c[i][j]=c[i-1][j-1]+1;
11.        else
12.          if(c[i-1][j] > c[i][j-1])
13.              c[i][j]=c[i-1][j];
14.          else
15.              c[i][j]=c[i][j-1];
16. }
```

در مرحلهٔ چهارم ساختار جواب بهینه را با اطلاعات محاسبه شده به‌دست می‌آوریم. برای این منظور از یک آرایهٔ کمکی دیگر به اسم d استفاده می‌کنیم. طول آرایه را برابر max{m, n} انتخاب می‌کنیم و کد بالا را به‌صورت زیر تغییر می‌دهیم.

```
1.  void LCS-length(String X,String Y)
2.  {
3.        int k=1;
4.          for(i=1; i<=m; i++)
5.              c[i][0]=0;
6.          for(i=1; i<=n; i++)
7.              c[0][i]=0;
8.          for(i=1; i<=m; i++)
9.            for(j=1; j<=n; j++)
10.             if(xi=yj)
11.               {
12.                   c[i][j]=c[i-1][j-1]+1;
13.                   d[k]=xi;
14.                   k++;
15.               }
16.             else
17.               if(c[i-1][j] > c[i][j-1])
18.                  c[i,j]=c[i-1][j];
19.               else
20.                  c[i,j]=c[i][j-1];
21. }
22. //************************
23. void printstring(int d[],int k)
24. {
25.       for(i=1; i<k; i++)
26.         cout<<d[i]<<"\t,\t";
27. }
```

کار در کلاس ۱۰-۲ :

الگوریتمی برای یافتن جملهٔ n-اُم دنباله‌ی فیبوناچی را به‌وسیلهٔ برنامه‌نویسی پویا ارائه کنید.

۱۰-۱۴: تمرین

۱. یک جدول $n \times n$ داریم و آن را با اعداد **1** و **0** پرکرده‌ایم. هدف یافتن مسیری از سطر اول به سطر آخراست. در هر مرحله مجاز هستیم به خانه‌های پایین و یا راست و یا چپ برویم، به این شرط که ازجدول‌خارج نشویم. در هر خانه که وارد می‌شویم به اندازهٔ عدد نوشته شده در آن خانه به ما پول می‌دهند.مسیری را بیابید که بیشترین پول را نصیب ما کند.

۲. فرض کنید یک ماشین داریم و $\{a_1, a_2, \ldots, a_n\}$ مجموعه‌ای از n کار است که باید ماشین آنها را انجام دهد. هرکارِ a_j یک مدت زمان برای اجرا می‌خواهد که برابر t_j است و یک سود که برابر p_j است، و یک مهلت که برابر d_j است، دارد. ماشین در هر لحظه فقط یک کار را می‌تواند انجام دهد و اگر کاری را در مهلت مشخص شدهٔ آن کار انجام دهیم سود آن کار را می‌گیریم وگرنه پولی به ما تعلق نمی‌گیرد. هرکار را باید به‌طور پیوسته انجام داد، یعنی یک کار را که شروع می‌کنیم باید آن‌را تمام کنیم و نمی‌توانیم‌آن را ناتمام رها کرده و به سراغ کار دیگری برویم و سپس به سراغ کار ناتمام بیاییم. ماشین از زمان صفرشروع به‌کار می‌کند. یک الگوریتم برای به‌دست آوردن ماکزیمم سود طراحی کنید. زمان اجرای برنامهٔ خود را تحلیل کنید. اگر از برنامه‌نویسی پویا کمک نگیریم زمان حل مسأله چقدر خواهد بود.

۳. رشتهٔ S از اعداد حقیقی که شامل لااقل یک عدد مثبت است، مفروض است. منظور از زیر رشتهٔ S تعدادی از عناصر این رشته است که متوالی باشند، یعنی اگر عناصر رشتهٔ S از اندیس i تا اندیس j که $(i \le j)$ می‌باشد را در نظر بگیرید. حاصل، زیر رشته‌ای از S است که اندیس عنصر اولش i و اندیس عنصرآخرش j است. زیر رشته‌ای از S مانند S_1 را می‌خواهیم پیدا کنیم که مجموع عناصر آن ماکزیمم باشد. الگوریتمی به روش برنامه-نویسی پویا برای این مسئله که از $O(n)$ باشد طراحی کنید.

٤. فرض کنید عمل ستاره را روی حروف **a** و **b** و **c** به شیوه‌ای که در جدول پایین آمده است مشخص کنیم. یعنی $a \times b = b$ و $b \times a = c$ و...

×	a	b	c
a	b	b	a
b	c	b	a
c	a	c	c

در برنامه یک رشته مثل **bbbac** به ما می‌دهند و از ما می‌خواهند مشخص کنیم آیا پرانتزگذاریی وجود دارد که حاصل رشته **a** شود. مثلاً برای این رشته پرانتزگذاری زیر به‌منظور رسیدن به **a** وجود دارد.

((b (b b)) (b a)) c = a

با استفاده از برنامه‌نویسی‌پویا الگوریتمی برای حل این مسئله ارائه دهید.

۱۰– ۱۵ : تمرینات تکمیلی

۱. یکی از راه‌های حل روابـط استفاده از انتـگرال هست. می‌دانیم که اگر f(k) یک تابع صـعودی باشد رابطهٔ زیر را خواهیم داشت.

$$\int_{m-1}^{n} f(x)\,dx \le \sum_{k=m}^{n} f(k) \le \int_{m}^{n+1} f(x)\,dx$$

و اگر f(k) نزولی باشد داریم:

$$\int_{m}^{n+1} f(x)\,dx \le \sum_{k=m}^{n} f(k) \le \int_{m-1}^{n} f(x)\,dx$$

حال با استفاده از این اطلاعات، رابطهٔ زیر را حل می‌کنیم.

$$T(n) = \sum_{k=1}^{n} \frac{1}{k}$$

پس داریم:

$$\ln(n+1) = \int_{1}^{n+1} \frac{dx}{x} \le \sum_{k=1}^{n} \frac{1}{k} \le \int_{1}^{n} \frac{dx}{x} = \ln n \rightarrow T(n) = \ln n$$

حال رابطه‌های زیر را با استفاده از مطالب فوق حل کنید.

$$T(n) = \sum_{k=1}^{n} \frac{1}{k^2}$$

$$T(n) = \sum_{k=1}^{n} k^3$$

$$T(n) = \sum_{k=1}^{n} k^2$$

$$T(n) = \sum_{k=1}^{n} k^3 + k^2 + k + 12$$

$$T(n) = \sum_{k=1}^{n} \frac{2}{3k}$$

$$T(n) = \sum_{k=1}^{n} k^4 + \sum_{k=1}^{n} \frac{1}{k^2} + \sum_{k=1}^{n} 100$$

$$T(n) = \sum_{k=1}^{n} \lg^s k$$

$$T(n) = \sum_{k=1}^{n} k^r \lg k$$

$$T(n) = \sum_{k=1}^{n} \left\lceil \frac{n}{2k} \right\rceil$$

۲. یک آرایهٔ N×N داده شده است. این آرایه را با 0 و 1 پرکردیم. ابعاد کوچکترین زیر آرایه (مستطیل)را بیابید که شامل تمام 1ها باشد مثلاً در شکل زیر آرایهٔ اصلی 3×3 است و ابعاد کوچکترین زیرآرایه‌که شامل تمام 1ها باشد 3×2 است.

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \rightarrow 3 \times 2$$

و در این شـکل هم ابعاد آرایه 4×4است و ابعاد کوچکترین زیر آرایه که شامـل تمام 1ها باشد1×1 است.

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \rightarrow (1) \rightarrow 1 \times 1$$

الگوریتمی بیابید که ابعاد زیرآرایه را بیابد.

۳. مسألهٔ ۲ را برای حالتی حل کنید که هدف یافتن یک مـربع شامل تمام 1ها باشد. الـگوریتم خود رادقیق بنویسید.

٤. آرایهٔ مرتب شدهٔ A به طول n از اعداد داریم. در این آرایه هیـچ دو عددی برابر نیستند. می‌خواهیم محل‌هایی را مشخص کنیم که در آنها A[i]=i با اسـتفاده از روش تقسـیم و حل الـگوریتمی از زمان(O(lgn برای این مسأله بیابید.

۵. روابط بازگشتی همزمان زیر را حل کنید.

$a_n = 3a_{n-1} + 2b_{n-1}$ $\quad a_0 = 1$
$b_n = a_{n-1} + 2b_{n-1}$ $\quad b_0 = 2$

٦. تابع f(x) به‌صورت زیر تعریف می‌شود. الگوریتمی بازگشتی برای محاسبهٔ آن بیابید.

$f(x) = 3g(x-1) + 4f(x-2) + 1 \quad x > 5$
$f(x) = 3g(x+1) + 9f(x+2) + 1 \quad x < 5$
$g(x) = 3g(x-1) + 5f(x-2) + 1 \quad x > 5$
$g(x) = 2g(x+1) + 4f(x+2) + 1 \quad x < 5$
$f(x) = 1 \quad x = \pm 5$
$g(x) = -1 \quad x = \pm 5$

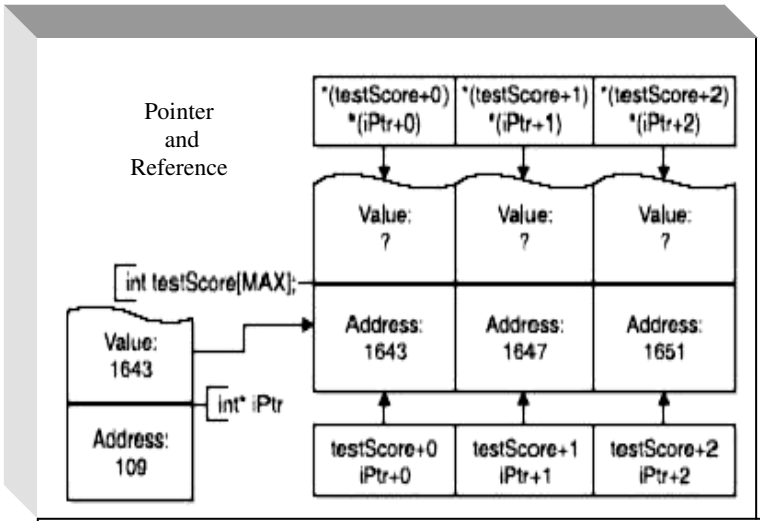
۷. تابع f(x) به‌صورت زیر تعریف می‌شود الگوریتمی بنویسید که f(x) را برای مقادیر کمتر از ۱۰۱به‌دست آورد.

$$f(x) = \begin{cases} x-10 & x>100 \\ f(f(x+11)) & 0 \le x \le 100 \\ 0 & x<0 \end{cases}$$

۸. یک الگوریتم بازگشتی دیگر برای محاسبهٔ مقادیر دنبالهٔ فیبوناچی در زیر ارائه شده است، با مراجعه به لوح فشردهٔ همراه کتاب می‌توانید برنامهٔ مربوط به این الگوریتم را بیابید و اجرا کنید. با مطالعهٔ کد مربوط به این برنامه چه نتیجه یا نتایجی در خصوص شیوهٔ حل الگوریتم‌ها به‌دست می‌آورید. مقدار انواع پیچیدگی‌های این الگوریتم را به‌دست آورید.

```
/* A Faster recursive Fibonaci function*/
/* Use a formula from Knuth Vol 1 page 80, section 1.2.8:
  If F[n] is the n'th Fibonaci number then
    F[n+m] = F[m]*F[n+1] + F[m-1]*F[n].
  First set m = n+1
    F[ 2*n+1 ] = F[n+1]**2 + F[n]*2.
  Then put m = n,
    F[ 2*n ] = F[n]*F[n+1] + F[n-1]* F[n],
  and replace F[n+1] by F[n]+F[n-1],
    F[ 2*n ] = F[n]*(F[n] + 2*F[n-1]).
*/
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
/*GLOBAL*/  long fib_count;
//*****************************
long fibo(int n)
{
        cerr << "      fibo(" << n << ") called\n";
    fib_count = fib_count + 1; //count each call
    if(n==0)
     {
       cerr << "      fibo returned " << 0 << "\n" ;
       return 0;
     }
    else if( n <= 2 )
     {
       cerr << "      fibo returned " << 1 << "\n" ;
       return 1;
     }
    else if( n%2 == 0 ) // n is even
     {
        const int half = n/2;
         const long f1 = fibo(half);
          const long f2 = fibo(half-1);
       cerr<<"   fibo returned "<<f1*(f1 + 2*f2) << "\n";
        return f1*(f1 + 2*f2);
     }
    else // n is odd
     {
```

```
42.       const int nearhalf = (n-1)/2;
43.        const long f1 = fibo(nearhalf+1);
44.         const long f2 = fibo(nearhalf);
45.       cerr<<"   fibo returned " << f1*f1 + f2*f2 << "\n";
46.       return f1*f1 + f2*f2;
47.     }
48. }//end fibo
49. //*********************************
50. int main ( int argc, char* argv[] )
51. {
52.        int first, last;
53.
54.        if( argc <= 1 )
55.          {
56.            first= 0;
57.            last = 5;
58.          }
59.        else if ( argc <= 2 )
60.          {
61.            first= 0;
62.            last = atoi(argv[1]);//ASCII to int conversion
63.          }
64.       else
65.          {
66.            first= atoi(argv[1]);
67.            last = atoi(argv[2]);
68.          }
69.
70.      for(int i=first; i<= last; i++)
71.          {
72.             fib_count = 0;
73.             cout << setw(4) <<setw(4) <<i <<", "
74.             << setw(10) << fibo(i) ;
75.             // Do not output fibo(i) and then
76.             // fib_count in one cout.
77.             // C++ will print the fib_count BEFORE
78.             // fibo(n) is executed
79.              cout<< setw(10) << "  fib_count: "
80.                  << fib_count << endl;
81.           }
82.
83.     return 0;
84. }
```



فصل یازدهم

اشاره‌گرها و مرجع

اهداف فصل و چکیده مطالب :

۱. آشنایی با مفهوم اشاره‌گر و کاربردهای آن
۲. انواع کاربرد دو عملگر * و & در رابطه با اشاره‌گرها و مرجع
۳. فراخوانی توابع با آدرس به وسیلهٔ اشاره‌گرها
۴. اشاره‌گر به توابع و کاربردهای آن
۵. اشاره‌گرها و تخصیص حافظه به‌صورت پویا
۶. طراحی منو و کارکردن با کلیدهای ترکیبی صفحه کلید
۷. آرایه‌ای از اشاره‌گرها
۸. ارتباط رشته‌های زبان C و اشاره‌گرها
۹. مرجع و فراخوانی توابع با ارجاع

۱۱-۱ : مبانی اشاره گرها (pointer)

مقدمه

مشکل‌بودن فهم اشاره‌گرها، و کارکردن با آنهایکی از دلایل عمدهٔ شهرت آنها محسوب می‌شود. اشاره‌گرها در زبان C و C++ بیشتر از هر زبان دیگری (مثل پاسکال یا بیسیک) مورد استفاده قرار می‌گیرد، و یکی از دلایل قدرت زبان‌های C و C++ نسبت به زبان‌هایی چون جاوا و C# که فاقد اشاره‌گر هستند محسوب می‌شود. هنگامی‌که می‌خواهیم با آدرس های حافظه به‌طور مستقیم کار کنیم ناگزیر باید به سراغ اشاره‌گر و مباحث مرتبط با آن برویم. اشاره‌گرها کاربردهای متعددی دارند که از آن جمله می‌توان موارد زیر را نام برد:

۱. دسترسی مستقیم به آدرس‌های حافظه
۲. دسترسی به عضوهای آرایه
۳. انتقال آرگومان به یک تابع، وقتی که می‌خواهیم تابع پارامترهای ارسالی را دستکاری کند
۴. انتقال آرایه و رشته‌ها به توابع
۵. گرفتن حافظه از سیستم به‌طور پویا
۶. ایجاد ساختمان داده‌هایی نظیر لیست‌های پیوندی و درخت‌های دودویی

به دلیل اهمیت مبحث اشاره‌گر و پیچیدگی‌هایی که این مبحث دارد سعی بر آن داشتیم تا مثال‌های این بخش در نهایت سادگی طرح ریزی شوند تا خوانندگان بیشتر با نحوهٔ به‌کارگیری و عملکرد اشاره‌گرها آشنایی پیدا کنند.

اما اگر بخواهیم تعریفی از اشاره‌گرها ارائه دهیم به‌صورت زیر خواهد بود:

« اشاره گر عبارت است از متغیری که قادر است آدرس یک نوع متغیر را در خود ذخیره کند.»

متأسفانه یکی از عمده مسائلی که بر پیچیدگی مبحث اشاره‌گرها و مرجع می‌افزاید وجود دو عملگر × و & می‌باشد که هر کدام دارای دو کاربرد متفاوت هستند، و لذا عمدتاً درک و تمییز دادن آنها برای برنامه‌نویسان مبتدی تا حدودی سخت است. لذا برای انتقال بهتر این مفاهیم به خوانندگان مجبور شده‌ایم برخی مفاهیم استاندارد و معمول در کار با اشاره‌گرها را نادیده بگیریم و قراردادهایی را (که منحصراً در این کتاب به‌کار رفته است) وضع کنیم. همچنین داشتن نهایت دقت و توجه از سوی خوانندگان عزیز به کاربردهای مختلف این دو عملگر بسیار مهم است.

تعریف متغیر اشاره‌گر

اشاره‌گرها نیز همانند دیگر انواع متغیرها که پیشتر دیدید، دارای نوع هستند. به‌عنوان مثال گفته می شود فلان متغیر، " اشاره‌گری به نوع **int** " است، و یا اشاره‌گر دیگری را " اشاره‌گری از نوع **float** " می‌دانند. این بدین معنی است که هر متغیر اشاره‌گر تنها قادر است آدرس یک نوع خاص از متغیرها مثلاً نوع **int** یا نوع **char** یا نوع **float** و... را در خود ذخیره کند. برای آنکه دلیل این امر را بفهمید باید توجه شما را به مطالبی که در خصوص تقسیم بندیِ سگمنت داده‌ها در بخش ۳-۲ مطرح ساختیم، جلب کنیم. در قسمت ۳-۲ مطرح شد که سگمنت داده جهت ذخیرهٔ انواع مختلف داده تقسیم‌بندی می‌شود. هنگامی که یک کامپیوتر آغاز به‌کار می‌کند، شروع به شناسایی قطعات مختلف سخت‌افزاری خود از جمله حافظهٔ اصلی(**RAM**) می‌کند. هنگامی که حافظهٔ اصلی شناخته می‌شود، پردازشگر مقدار این حافظه را تشخیص داده و به تک تک خانه های حافظه (منظور از یک خانهٔ حافظه یک بایت است نه بیت های حافظه) یک آدرس نسبت می‌دهد. آدرس خانه‌های حافظه همچون پلاک منازل یک عدد است، با این تفاوت که آدرس خانه‌های حافظه بر مبنای ۱۶ (هگزادسیمال) است. وظیفهٔ آدرس‌دهی به خانه‌های حافظه به عهدهٔ **CPU** می‌باشد. هر**CPU** براساس قدرتی که در خصوص آدرس‌دهی خانه‌های حافظه دارد تعدادی از پایه‌های **IC** آن به **Address Bus** سیستم متصل می‌شود.

بر این اساس اگر شما یک مگابایت حافظه در سیستم خود داشته باشید اولین بایت از حافظه دارای آدرس **0H** (معادل صفر در مبنای ۱۰) و آخرین خانهٔ آن دارای آدرس **FFFFFH** معادل (۱.۰۴۸.۵۷۵در مبنای ۱۰) خواهد بود. لذا از لحاظ آدرس‌دهی، خانه‌های حافظه هیچ تفاوتی با یکدیگر ندارند، پس چرا هر اشاره‌گر تنها قادر به ذخیرهٔ آدرس یک نوع متغیر خاص است!؟ چنانکه در بخش ۳-۲ نیز دیدید، سگمنت داده ها بر اساس انواع مختلف موجود در برنامه تقسیم‌بندی می‌شود. لذا هنگامی که آدرس متغیری از نوع **int** را در اشاره‌گری از همان نوع ذخیره کرده و می‌خواهید به محتویات متغیر **int** از طریق آدرس آن (به کمک اشاره‌گر) دست پیدا کنید، با توجه به آنکه نوع **int** دارای طولی برابر ۴ بایت است، کنترل اجرای برنامه از محلی که متغیر اشاره‌گر، به آن اشاره می‌کند (یعنی محلی که آدرس آن را دربر دارد) تا چهار بایت بعدی را می‌خواند، بدین ترتیب دسترسی به محتویات متغیری از نوع **int** به واسطهٔ اشاره‌گری که حـاوی آدرس آن باشد، میسّر می‌شود. حال فرض کنید می‌خواهید به محتویات متغیری از نوع **double** به‌واسطهٔ اشاره‌گری به نوع **double** که حاوی آدرس این متغیر است دست پیدا کنید. دوباره کنترل اجرای برنامه به محلی از حافظه که اشاره‌گر مذکور به آن اشاره می‌کند رفته، امّا این بار هشت بایت از حافظه را **load** (بارگذاری) می‌کند. لذا با وجود آنکه تمامی انواع اشاره‌گر یک آدرس ۳۲ بیتی را در خود ذخیره می‌سازند و دارای طولی برابر ۴ بایت هستند، تفکیک انواع متغیر اشاره‌گر برای کامپایلر امری اجتناب ناپذیر است. حال ببینیم یک متغیر اشاره‌گر را چگونه می‌توان تعریف کرد.

جهت تعریف یک متغیر اشاره‌گر به‌صورت زیر باید عمل کرد:

* نوع اشاره‌گر ; نام اشاره‌گر

به‌عنوان مثال در زیر متغیر اشاره‌گری با نام **p** تعریف شده که قادر است آدرس متغیری از نوع **int** را در خود ذخیره سازد:

```
int* p;
```

مثال ۱۱-۱ : برنامه‌ای که نشان می‌دهد انواع مختلف اشاره‌گر تنها به ٤ بایت جهت ذخیرهٔ آدرس یک متغیر نیازمند هستند.

```
1. //This program shows the size of pointer variable.
2. #include<iostream>
3. using namespace std;
4. int main()
5. {
6.    int* p;
7.    float* q;
8.    double* r;
9.   cout<<"The size of an int pointer is ="<<sizeof(p);
10.  cout<<"\nThe size of a float pointer is ="<<sizeof(q);
11.  cout<<"\nThe size of a double pointer is ="<<sizeof(r);
12.  cout<<endl;
13. return 0;
14.}
```

توضیح مثال : خروجی حاصل از اجرای این برنامه به‌صورت زیر خواهد بود:

```
The size of an int pointer is =4
The size of a float pointer is =4
The size of a double pointer is =4
```

در این قسمت اولین کاربرد عملگر * را به‌منظور تعریف متغیری از نوع اشاره‌گر دیدید. اندکی بعد کاربرد دوم این عملگر را نیز به‌عنوان عملگر مقدار خواهید دید.

تذکر مهم : چنانکه دیدید جهت تعریف یک متغیر اشاره‌گر به نوع **int** به‌صورت زیر عمل باید کرد:

```
int* p;
```

امّا آیا می‌توان برای تعریف دو اشاره‌گر از نوع **int** نیز به‌صورت زیر عمل کرد:

```
int* p, q;
```

اگر چنین دستوری را بنویسید، متغیر**p** به‌صورت یک اشاره‌گر از نوع **int** و متغیر **q** یک متغیر معمولی از نوع **int** تعریف می‌شود. علت این امر بدین جهت است که به‌طور استاندارد برای تعریف یک متغیر اشاره‌گر باید به‌صورت زیر عمل کرد:

```
int *p;
```

یعنی باید علامت * را به نام متغیر بچسبانیم. امّا متأسفانه اگر نحوهٔ تعریف اشاره‌گر را بدین صورت انجام بدهیم، خوانندگان را در درک و تمییز، عملکرد عملگر * در موقع تعریف اشاره‌گر با زمانی‌که این عملگر به‌عنوان عملگر مقدار به‌کار می‌رود با مشکل مواجه خواهیم کرد. لذا به‌عنوان یک قرارداد منحصر به این کتاب به جای چسباندن ستاره قبل از نام اشاره‌گر، علامت * را به نوع اشاره‌گر چسبانده و از این پس قرارداد می‌کنیم که اشاره‌گری به نوع صحیح را به‌صورت **int*** نشان دهیم، در حالی که در تمامی منابع دیگر C++ اشاره‌گری به نوع صحیح را به‌صورت * **int** نشان می‌دهند. به همین شکل **float*** به مفهوم اشاره‌گر به نوع **float** تعبیر می‌شود.

امّا در بالا دیدید که تعریف دو اشاره‌گر به‌طور همزمان به شکل زیر:

```
int*  p, q;
```

امکانپذیر نیست. به نظر شما برای رفع این مشکل چه باید کرد؟

به‌طور استاندارد برای تعریف دو اشاره‌گر از نوع **int** در یک خط باید به‌صورت زیر عمل کرد:

```
int  *p, *q;
```

امّا مطابق آنچه قرارداد کردیم، برای خواناتر کردن دستورات برنامه از چسباندن علامت * به نام اشاره‌گر خودداری می‌کنیم. البته دلیل این پیشنهاد را اندکی بعد در مبحث «* به‌عنوان عملگر مقدار» خودتان متوجه خواهید شد. امّا برای تعریف چند متغیر اشاره‌گر به‌طور همزمان در این کتاب به‌صورت زیر عمل خواهیم کرد:

```
int* p,* q,* r;
```

چنانکه در دستور فوق می‌بینید برای تک تک متغیرهای اشاره‌گر **p** و **q** و **r** علامت * را قرار داده‌ایم، امّا در عین‌حال علامت * را به این متغیرها نچسبانده‌ایم.

علامت & به‌عنوان عملگر آدرس

تا اینجا یاد گرفتیم که چگونه یک اشاره‌گر را برای ذخیرهٔ آدرس یک متغیر تعریف کنیم، امّا صحبتی از نحوهٔ دستیابی به آدرس یک متغیر در حافظه و ذخیرهٔ آن در اشاره‌گر نکردیم. به واسطهٔ عملگر & می‌توان به آدرس هر متغیر یا شئی در حافظه دسترسی پیدا کرد. به‌عنوان مثال در زیر آدرس متغیر **i** در اشاره‌گری از نوع صحیح به نام**p** ذخیره شده است:

```
int  i = 7236;
int* p = &i;
```

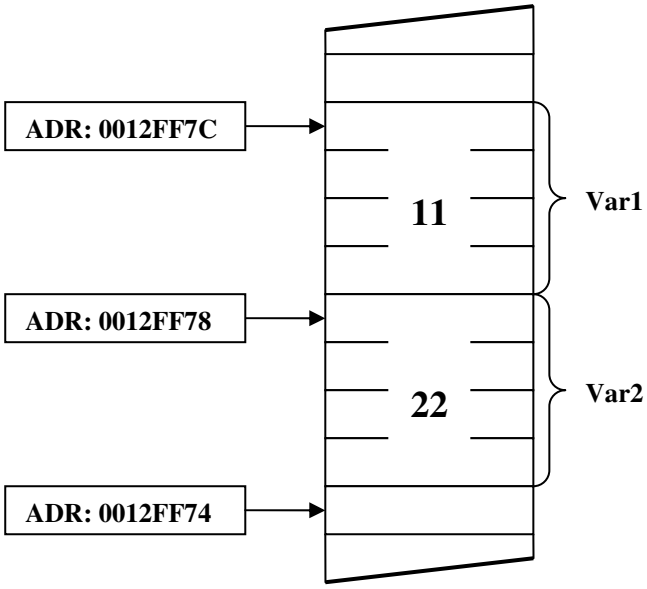
مثال ۱۱-۲: برنامه‌ای که چگونگی به‌کارگیری عملگر آدرس را نشان می‌دهد.

```
1. //This program shows the usage of address operator.
2. #include <iostream>
3. using namespace  std;
4. int main()
5. {
6.       int var1=11, var2=22;
7.        cout<<" Address of var1 is:"<<&var1<<endl;
8.       int* p;
9.       p = &var2;
10.        cout<<" Address of var2 is:"<<p<<endl;
11.   return 0;
12. }
```

توضیح مثال : چون مدیریت حافظه امکان دارد این برنامه را در نقاط متفاوتی از حافظهٔ **RAM** اجرا کند لذا ممکن است خروجی حاصل از اجرای این برنامه در کامپیوتر شما مشابه خروجی زیر نباشد. امّا ما با اجرای این برنامه خروجی زیر را دریافت کردیم:

```
Address of var1 is: 0012FF7C
Address of var2 is: 0012FF78
```

با اجرا شدن این برنامه دو متغیر var2 وvar1 به‌صورت زیر در حافظه ایجاد می‌شوند:



شکل ۱۱- ۱ : آدرس‌دهی متغیرها در حافظه

اگر 12FF78H, 12FF7CH را به مبنای ده تبدیل کنیم به ترتیب اعداد 1245052 , 1245048 به‌دست می‌آیند. که اگر این دو عدد را از یکدیگر کم کنیم اختلافی برابر ۴ بایت به‌دست می‌آید که با شکل فوق مطابقت دارد. همچنین اگر اشاره‌گرهای مورد نظر اشاره‌گرهایی از نوعdouble بودند اختلاف اعداد چاپ شده برابر ۸ بایت بود. امّا ممکن است برای شما این سؤالی پیش‌آید که چرا آدرس نسبت داده شده بهvar1 بزرگتر از آدرس نسبت داده شده به var2 است، باید گفت این مسئله به ساختمان کامپیوتر و عملکردCPU در نحوهٔ آدرس‌دهی خانه‌های حافظه برمی‌گردد، که به اولین خانه بزرگترین آدرس را داده و به ترتیب یک واحد از مقدار آدرس کم می‌کند تا به بایت بعدی برسد. البته ما در کارکردن با اشاره‌گرها نیازی به توجه به این مطلب نداریم.

توجه: در برخی از کامپایلرها عملگر درج (<<) پیشوند 0x را قبل از آدرس‌ها چاپ می‌کند یعنی ممکن است خروجی این برنامه در سیستم شما به‌صورت زیر باشد:

```
Address of var1 is: 0x12FF7C
Address of var2 is: 0x12FF78
```

علامت * به‌عنوان عملگر مقدار

اولین کاربرد عملگر ستاره را در هنگام تعریف یک اشاره‌گر دیدید. در اینجا کاربرد دوم این عملگر را خواهید دید که از عملگر ستاره به جهت دستیابی به محتویات مکانی که یک اشاره‌گر به آن اشاره می‌کند استفاده خواهیم کرد. پیشتر دیدید که با چاپ کردن یک اشاره‌گر محتویات داخل خود متغیر اشاره‌گر یعنی آدرس یک متغیر دیگر چاپ می‌شود. به‌عنوان مثال دستور زیر موجب چاپ یک عدد هگزا دسیمال می‌شود که آدرس متغیر **x** است:

```
int x;
int*  p = &x;
cout<<p; //print the address of x
```

امّا برای آنکه بتوانیم به محتویات متغیر **x** به واسطهٔ آدرس آن (به عبارت دیگر به واسطهٔ اشاره‌گر **p**) دست پیدا کنیم باید از یک علامت * به‌عنوان عملگری که محتویات آدرس بعد از خود را به‌دست می‌دهد استفاده کنیم. به‌عنوان مثال قطعه کد زیر موجب می‌شود که محتویات **x** یعنی عدد ۴۲۵ به‌جای آدرس آن چاپ گردد:

```
int x = 425;
int*  p = &x;
cout<<*p; //print 425
```

تذکر۱: اگر به اشتباه بخواهید آدرس یک متغیر از یک نوع را در اشاره‌گری از نوع دیگر بریزید، کامپایلر یک پیغام خطا صادر می‌کند. به‌عنوان مثال دستورات زیر را در نظر بگیرید:

```
int* p;
float f;
p = &f;
```

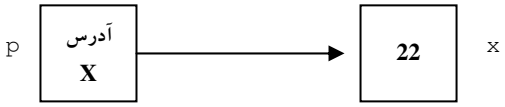
با کامپایل این دستورات با خطای زیر مواجه خواهید شد:

***' error C2440: '=' : cannot convert from 'float *' to 'int**

تذکر۲: اگر عملگر آدرس را جلوی یک متغیر از نوع اشاره‌گر به‌کار ببرید آدرس خود اشاره‌گر چاپ می‌شود. به‌عنوان مثال دستورات زیر را در نظر بگیرید:

```
int  x = 22;
int* p = &x;
cout<<p<<"="<<&x<<endl;
cout<<p<<"!="<<&p<<endl;
```

با اجرای این دستورات خواهید دید که آدرس متغیر **x** با محتویات داخل اشاره‌گر **p** برابر است امّا آدرس خود اشاره‌گر **p** با محتویات داخل آن متفاوت است. به‌طوری که می‌توانید شکل زیر را در مورد موقعیت **p** و **x** در حافظه به‌صورت نمادین تصور کنید.



در حقیقت متغیر اشاره‌گر همانند یک ظرف است که محتویات آن می‌تواند یک آدرس باشد بنابراین خود اشاره‌گر نیز فضایی معادل ۴ بایت را در حافظه اشغال می‌کند.

اعمال برروی اشاره‌گرها

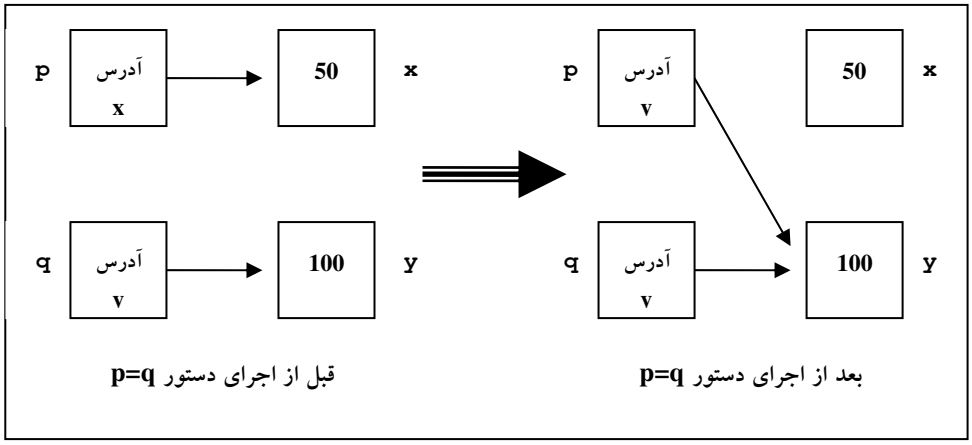
از آنجا که اشاره‌گرها متغیر هستند پس می‌توان برخی از عملگرها را در مورد آنها به کاربرد البته برخی از اعمال برروی اشاره‌گرها مثلاً ضرب یک اشاره‌گر در اشاره‌گر دیگری امکانپذیر است امّا شاید هیچ کاربرد و ارزشی از نظر برنامه‌نویسی نداشته باشد، لذا در این قسمت تنها به بررسی سه نوع عملیات تقریباً پرکاربرد برروی اشاره‌گرها می‌پردازیم.

۱. عمل انتساب اشاره‌گرها به یکدیگر

چنانکه در مورد متغیرها دیدید دستور K = L موجب می‌شود که محتویات متغیر L در متغیر K قرارگیرد. بنابراین با اجرای دستور انتساب برروی دو اشاره‌گر، محتویات اشاره‌گر سمت راست دستور انتساب در اشاره‌گر سمت چپ دستور انتساب قرار می‌گیرد. به‌عنوان مثال دستور زیر را در نظر بگیرید:

```
int  x = 50, y = 100;
int* p = &x;
int* q = &y;
p = q;
```

آنچه که پس از اجرای آخرین دستور قطعه کد فوق رخ می‌دهد در شکل زیر نشان داده شده است:



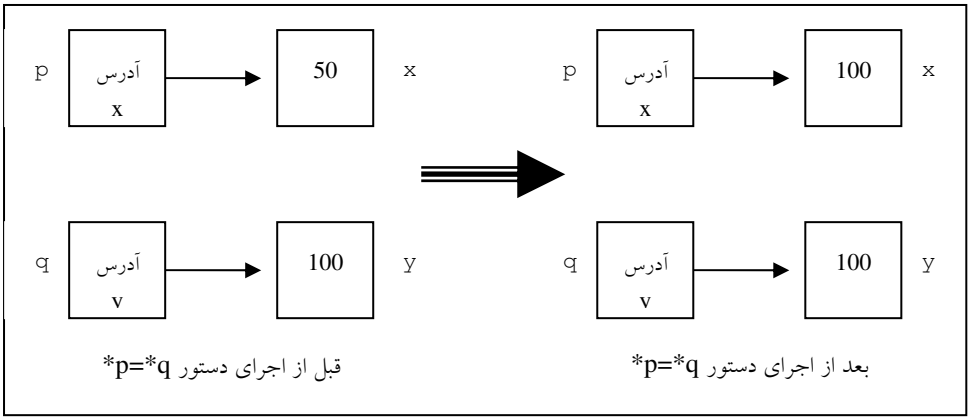
شکل ۱۱- ۲ : عمل انتساب برروی اشاره‌گرها بدون استفاده از عملگر مقدار

مطابق آنچه در شکل ۱۱-۲ نشان داده شده پس از اجرای دستور p=q محتویات هر دو اشاره‌گر p و q مشابه یکدیگر می‌شود. به عبارت دیگر با اجرای این دستور آدرس y در اشاره‌گر p نیز قرار می‌گیرد زیرا محتویات q که شامل آدرس y بوده در p ریخته می‌شود.

حال اگر بخواهید به‌واسطهٔ اشاره‌گرهای p و q دستوری معادل x = y بنویسید که محتویات متغیر y را در داخل متغیر x بریزید باید از عملگر مقدار استفاده کنیم، مطابق آنچه در زیر نوشته شده است:

```
int   x = 50, y = 100;
int*  p = &x;
int*  q = &y;
*p = *q;
```

آنچه که پس از اجرای آخرین دستور قطعه کد فوق رخ می‌دهد در شکل زیر نشان داده شده است:

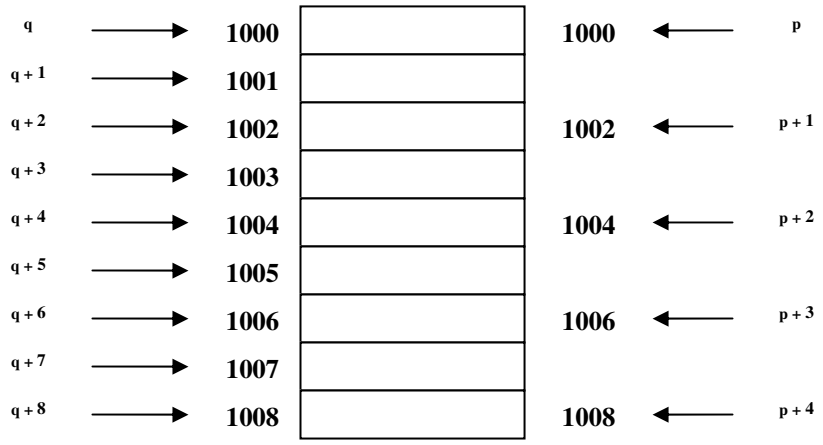


شکل ۱۱- ۳ : عمل انتساب بر وی اشاره‌گرها همراه با عملگر مقدار

۲. اعمال محاسباتی جمع و تفریق برروی اشاره‌گرها

اعمال جمع و تفریق را می‌توان برروی اشاره‌گرها انجام داد. با افزایش یک واحد به یک اشاره‌گر، به اندازهٔ طول نوع اشاره‌گر به محتویات داخل آن اشاره‌گر اضافه می‌شود. به‌عنوان مثال اگر **p** یک متغیر اشاره‌گر از نوع **short** باشد که به محل ۱۰۰۰ حافظه اشاره نماید، **p++** موجب می‌شود تا **p** به محل ۱۰۰۲ حافظه یعنی به متغیر **short** بعدی اشاره کند. امّا اگر **q** متغیر اشاره‌گر دیگر از نوع **char** باشد که به همان محل ۱۰۰۰ حافظه اشاره کند، **q++** موجب می‌شود، تنها یک بایت **q** جلو برود و به محل ۱۰۰۱ حافظه اشاره کند. لذا می‌توانید شکل زیر را در خصوص این دو اشاره‌گر مدنظر قرار دهید تا به تفاوت عملکرد عملگر جمع برروی آن دو پی ببرید:

```
short*  p;
char*   q;
```



شکل ۱۱- ۴ : عملیات جمع با اشاره‌گرها

تذکر مهم: باید به تقدم عملگر * در عبارات محاسباتی توجه داشته باشید. چنانکه دستور **++p*** به صورت **(++p)*** ارزیابی می‌شود و هیچگاه به مکانی که **p** به آن اشاره می‌کند یک واحد نمی‌افزاید، بلکه ابتدا موقعیت اشاره‌گر **p** به میزان یک متغیر جلو برده و سپس دسترسی ما را به محتویات آن میسر می‌سازد، امّا **(p*)++** موجب افزایش محتویات محلی از حافظه که **p** به آن اشاره می‌کند، می‌گردد.

برروی اشاره‌گر عملیات دیگری نیز، همچون عملیات مقایسه را نیز می‌توان انجام داد، امّا به دلیلِ کمی کاربرد آنها از پرداختن به دیگر عملگرها خودداری کرده و مطالعهٔ بیشتر در این زمینه را به عهده خوانندگان عزیز محول می‌سازیم.

اشاره‌گر نوع void

قبلاً متذکر شدیم که یک اشاره‌گر نمی‌تواند آدرس متغیری که هم نوع با خودش نباشد را در خود ذخیره سازد. امّا این قانون، یک استثناء دارد. یک نوع اشاره‌گر هم منظوره با نام اشاره‌گر نوع **void** وجود دارد که می‌تواند به هر نوع داده‌ای اشاره کند و به‌صورت زیر تعریف می‌شود:

```
void* ptr;
```

اینگونه اشاره‌گرها دارای کاربرد ویژه‌ای هستند به‌طوری که در انتقال اشاره‌گر به توابعی که به‌طور مستقل بر روی انواع دادهٔ مختلف عمل می‌کنند(توابع قالب) کاربرد دارند.

مثال ۱۱-۳: برنامه‌ای که نحوهٔ عملکرد اشاره‌گر نوع **void** را نشان می‌دهد.

```
1. //This program shows the usage of void pointer.
2. #include  <iostream>
3. using namespace  std;
4. int main()
5. {
6.    int  int_var;
7.    float flt_var;
8.    int* int_ptr = &int_var ;     //ok, int* to int*
9.    float*  flt_ptr = &flt_var ; //ok, float* to float*
10. //int_ptr = &flt_var ;         //error, float* to int*
11. //flt_ptr = & int_ptr ;        //error, int* to float*
12.   void* vid_ptr;
13.   vid_ptr = &int_var ;         //ok, int* to void*
14.   vid_ptr = flt_ptr ;          //ok, float* to void*
15.  return 0;
16. }
```

البته اگر به دلایل غیرمعمول، واقعاً نیاز داشته باشید که یک نوع اشاره‌گر را به نوع دیگر تبدیل کنید می‌توانید از تبدیل نوع **reinterpret_cast** استفاده کنید. به‌عنوان مثال خطوط ۱۰ و ۱۱ برنامهٔ مثال ۱۱-۳ را می‌توان به‌صورت زیر بدون هیچگونه خطای زمان کامپایلر بازنویسی کرد:

```
int_ptr = reinterpret_cast <int*> (&flt_var);
flt_ptr = reinterpret_cast <float*> (int_ptr);
```

۱۱-۲: اشاره گرها و توابع

ارسال آرگومان به توابع از طریق اشاره‌گر

در فصل نهم متذکر شدیم که می‌توان با استفاده از اشاره‌گرها پارامترهایی را به توابع ارسال کرد که، هرگونه عملیات که در داخل تابع برروی آرگومان صورت می‌گیرد بر روی پارامترهای ورودی نیز تأثیر بگذارد. لذا اگر بخواهید از داخل یک تابع، متغیرهای داخل برنامهٔ احضار کننده را تغییر دهید، آنگاه این متغیرها را نمی‌توانید از طریق مقدار منتقل کنید زیرا تنها مقادیر متغیرها در آرگومان‌های تابع کپی می‌شود. اما هنگامی که از اشاره‌گرها در تعریف آرگومان‌های یک تابع بهره می‌گیریم در واقع به جای آن‌که فراخوانی تابع با مقدار صورت پذیرد، فراخوانی تابع با آدرس صورت می‌گیرد، و لذا آرگومان های داخل تابع همان متغیرهای ارسالی در هنگام فراخوانی خواهند بود. قبل از آن که روش کلی در به‌کارگیری اشاره‌گرها را در آرگومان توابع مدنظر قرار دهیم، پیشنهاد می‌کنیم مثال زیر را به دقت بررسی کنید، زیرا با توجه به این مثال نکات کلیدی در این خصوص را بیان خواهیم کرد.

مثال ۱۱-۴: برنامه‌ای که با دریافت طول و عرض یک مستطیل مساحت و محیط آن را توسط توابع دیگری به روش فراخوانی با آدرس محاسبه کرده و به خروجی می‌برد.

```
1. //This program calls function by address
2. #include <iostream>
3. using namespace std;
4. void input(int*, int*);
5. void rect(int, int, int*, int*);
6. void main()
7. {
8.     int len, wid, area, perimeter;
9.      input(&len, &wid);
10.      rect(len, wid, &area, &perimeter);
11.     cout<<" Length = "<<len<<" , Width = "<<wid<<endl;
12.      cout<<" Area = "<<area<<" , Perimeter = "<<perimeter;
13.       cout<<endl;
14. }
15. //*************************
16. void input(int* ptr_len, int* ptr_wid)
17. {
18.      cout<<" Enter Length:";
19.        cin>>*ptr_len;
20.      cout<<" Enter Width:";
21.        cin>>*ptr_wid;
22. }
23. //*************************
24. void rect(int x,int y, int* ptr_area, int* ptr_perimeter)
25. {
26.     *ptr_area = x * y;
27.     *ptr_perimeter = 2 * (x + y);
28. }
```

چنانکه در این مثال می‌بینید هنگامی‌که از اشاره‌گرها می‌خواهیم در ارسال پارامترها به توابع استفاده کنیم باید نکات زیر را مد نظر قرار دهیم:

۱. در هنگام اعلان تابع تنها باید نوع اشاره‌گرها را ذکر کنیم، چنانکه در خط ۴ مثال ۱۱-۴ دیدید اعلان توابع به‌صورت زیر انجام گرفت:

```
void  input(int* , int* );
```

در حقیقت هر کدام از آرگومان‌ها که تغییرات آنها در تابعِ فراخوانده شده برای ما حائز اهمیت است، و می خواهیم که تغییرات اعمال شده بر آرگومان‌ها در تابع فرعی بر پارامترهای ورودی نیز اعمال شود باید به‌صورت اشاره‌گر اعلان گردند. همچنین چنانکه از فصل نهم به خاطر دارید جهت بازگرداندن بیش از یک مقدار توسط توابع باید از اشاره‌گرها استفاده کنیم. به‌عنوان مثال در تابع **rect** دیدید که طول و عرض مستطیل را به‌صورت عادی به تابع ارسال کردیم، امّا مساحت و محیط را که می‌خواستیم مقادیر آنها توسط تابع محاسبه و برگردانده شوند، با اشاره‌گرها به‌صورت زیر اعلان نمودیم:

```
void  rect(int, int, int*, int* );
```

۲. در هنگام تعریف تابع باید یک نام برای هر متغیر اشاره‌گر تعریف شده در اعلان تابع در نظر بگیریم تا آدرس‌های ارسالی به تابع را در خود ذخیره سازند، لذا در هنگام تعریف تابع **input** به‌صورت زیر عمل کردیم:

```
void input(int* ptr_len, int* ptr_wid)
{
   . . .
}
```

۳. در هنگام فراخوانی باید به ازای هر اشاره‌گری که در اعلان و عنوان تابع، تعریف کردیم، آدرس یک متغیر هم نوع با اشاره‌گر را به تابع ارسال کنیم. لذا عموماً در هنگام فراخوانی تابع از عملگر آدرس قبل از نام متغیرها استفاده می‌شود، مگر آنکه به‌جای یک متغیر معمولی از یک اشاره‌گر که حاوی آدرس متغیر موردنظر است در هنگام فراخوانی استفاده کنیم.

چنانکه در فراخوانی تابع **rect** می‌بینید برای متغیرهای نگه دارنده مقدار مساحت و محیط از عملگر آدرس استفاده کرده‌ایم، امّا برای متغیرهای حاوی مقدار طول و عرض که تنها ارسال مقادیر آنها کفایت می‌کند، عملگر آدرس قرار داده نشده است.

```
rect(len, wid, &area, &perimeter);
```

فراخوانی این تابع را می‌توان به‌صورت زیر بدون استفاده از عملگر آدرس انجام داد:

```
int* p = &area;
int* q = &perimeter;
rect(len, wid, p, q);
```

۴. چنانکه در مثال ۱۱-۴ دیدید در داخل توابع **input** و **rect** جهت اعمال تغییرات بر آرگومان‌ها از عملگر مقدار در جلوی نام اشاره‌گرها استفاده کردیم، لذا با اجرای دستور:

```
cin>>*ptr_len;
cin>>*ptr_wid;
```

مقادیری که توسط شئ **cin** از کاربر دریافت می‌شود در حقیقت در داخل متغیرهای **len** و **wid** ذخیره می شوند، چرا که به‌واسطهٔ دو اشاره‌گر **ptr_len** و **ptr_wid** و عملگر مقدار، محتویات دو متغیر **len** و **wid** را دستکاری کرده‌ایم. متأسفانه بسیاری از برنامه‌نویسان مبتدی فراموش می کنند که عملگر مقدار را در داخل توابع فرعی در جلوی نام اشاره‌گرها به‌کار ببرند، و مسلماً برنامهٔ خود را با مشکل مواجه می‌سازند.

اجرای توابع به واسطهٔ آدرس آنها

چنانکه از بخش ۳-۲ به خاطر دارید کدهایِ زبان ماشین معادل دستورات C++ برنامه، در بخشی از حافظه با نام **code segment** ذخیره می‌شود. کد سگمنت نیز به نوبهٔ خود بدین‌صورت قسمت‌بندی می‌شود که برای هر تابع قسمتی از کد سگمنت اختصاص داده می‌شود. نام یک تابع مانند نام آرایه که اشاره‌گری به اولین خانهٔ آن آرایه است، اشاره‌گری به آدرس شروع دستورات زبان ماشین معادل تابع، در حافظهٔ اصلی می‌باشد. یکی از ویژگی‌های جالب اشاره‌گرهای زبان C++ امکان اجرای توابع به واسطهٔ آدرس آن توابع در حافظه است. بدین‌ترتیب که می‌توان آدرس یک تابع را در درون یک اشاره‌گر قرار داد و سپس تابع را به واسطهٔ اشاره‌گر مذکور اجرا کرد. در این خصوص نیز ابتدا به مثال زیر توجه کنید و سپس توضیحات مندرج پس از برنامه را به دقت مطالعه نمایید.

مثال ۱۱-۵ : برنامه‌ای که نحوهٔ به‌کارگیری اشاره‌گر به توابع را نشان می‌دهد. در این برنامه با استفاده از اشاره‌گر به تابع، تابع **strcmp** را فراخوانی کرده و دو رشته را با یکدیگر مقایسه می‌کند.

```
1.  //This program shows how to use function pointers.
2.  #include <iostream>
3.  #include <string>
4.  using namespace std;
5.  void check(char*,char*,int(*)(const char*, const char*));
6.  int main()
7.  {
8.     char str1[80], str2[80];
9.     int(* p)(const char*, const char*);
10.    p = strcmp;
11.    cout<<" Enter first string :" ;
12.     cin>>str1 ;
13.    cout<<" Enter second string :";
14.     cin>>str2 ;
15.    check(str1, str2, p);
16.   return 0;
17. }
```

```
18. //*********************
19. void check(char* s1, char* s2,
20.             int (* cmp)(const char* , const char* ))
21. {
22.    if(!cmp(s1, s2))
23.       cout<<" Strings are equal.";
24.    else
25.       cout<<" Strings are not equal.";
26. }
```

با توجه به این مثال می‌توان نکات زیر را در خصوص استفاده از اشاره‌گر به تابع مطرح ساخت:

۱. اولین نکته نحوهٔ تعریف یک اشاره‌گرِ به تابع است. به خط نهم کد فوق توجه کنید. در این خط اشاره‌گری با نام **p** جهت اشاره به تابع **strcmp** تعریف شده است. نوع یک اشاره‌گرِ به تابع، باید هم نوع تابعی باشد که قرار است به آن اشاره کند. لذا نوع اشاره‌گر**p** را از نوع **int** در نظر گرفته‌ایم، زیرا تابع **strcmp** از نوع **int** می‌باشد. نکتهٔ دیگر آنکه در حین تعریف یک اشاره‌گرِ به تابع، باید اطراف علامت ستاره و نام اشاره‌گر یک جفت پرانتز باز و بسته قرار دهیم، و در نهایت اگر به خط نهم توجه کرده باشید نوع آرگومان‌های تابعی که قرار است اشاره‌گر به آن اشاره کند، باید پس از نام اشاره‌گر در داخل یک جفت پرانتز به‌عنوان آرگومان‌های اشاره‌گرِ به تابع، ذکر گردد. اگر به مستندات **MSDN** مراجعه کنید و تابع **strcmp** را جستجو کنید، خواهید دید اعلان این تابع به‌عنوان شکل کلی به کارگیری آن برای شما نشان داده خواهد شد:

```
int strcmp( const char *string1, const char *string2 );
```

حال با توجه به خط فوق به‌راحتی می‌توانید نوع پارامترهای ورودی تابع **strcmp** را تشخیص دهید.

۲. با توجه به آنکه گفتیم نام یک تابع خود یک اشاره‌گر می‌باشد، جهت قرار دادن آدرس یک تابع در یک اشاره‌گرِ به تابع هیچ نیازی به علامت **&** نیست، و تنها نسبت دادن نام تابع به اشاره‌گر مربوطه کفایت می‌کند.

۳. اگر به خط ۲۲ این کد توجه کنید خواهید دید که فراخوانی یک تابع به واسطهٔ اشاره گرِ به تابع، هیچ تفاوتی با فراخوانی آن تابع به‌صورت معمول ندارد. تنها به جای نام تابع باید نام اشاره‌گر را قرار داد. لذا خط ۲۲ این کد را به‌صورت زیر هم می‌توان نوشت:

```
if(!strcmp(s1, s2))
```

۴. اشاره‌گرِ به یک تابع، ممکن است مانند هر اشاره‌گری، به‌عنوان آرگومان به تابع دیگری ارسال شود. شاید این مورد مهمترین کاربرد اشاره‌گرهای به توابع باشد، که نمونهٔ آن را در مثال ۱۱-۶ خواهید دید. اما باید در اینجا نحوهٔ به‌کارگیری اشاره‌گر به یک تابع را به‌عنوان آرگومان تابعی دیگر فرابگیرید تا در فهم مثال ۱۱-۶ با مشکل مواجه نشوید. لذا به خطوط ۵ و ۱۵ و ۱۹ که به ترتیب شامل اعلان، فراخوانی و تعریف یک تابع با آرگومانی به‌صورت اشاره گر به تابع است، خوب توجه کنید.

مثال ۱۱-۶: برنامه‌ای که با استفاده از اشاره‌گر توابع یک مرتب‌سازی همه منظوره را پیاده‌سازی می‌کند.

```
1.  //This program uses function pointer to sort an array.
2.  #include <iostream>
3.  #include <iomanip>
4.  using namespace std;
5.  void input_array(int []);
6.  void show_array(int []);
7.  void bubble(int [], int(*)(int,int));
8.  int ascending(int, int);
9.  int descending(int, int);
10. const int arraysize = 5;
11. int main()
12. {
13.    bool order;
14.    int array[arraysize]= {0};
15.      input_array(array);
16.    system("cls");
17.    cout<<"\nData items in original order:\n";
18.      show_array(array);
19.    cout<<"Enter 1 to sort in ascending order,\n"
20.        <<"Enter 2 to sort in descending order: ";
21.     cin>>order;
22.    if(order == 1)
23.      {
24.         bubble(array,ascending);
25.         cout<<"\nData items in ascending order:\n";
26.      }
27.    else
28.     {
29.        bubble(array, descending);
30.        cout<<"\nData items in descending order:\n";
31.     }
32.    show_array(array);
33.  return 0;
34. }
35. //*********************
36. void input_array(int array[])
37. {
38.    for(int i=0;i<arraysize;i++)
39.       {
40.          cout<<" Enter number "<<(i+1)<<" : ";
41.           cin>>array[i];
42.       }
43. }
```

```
44. //*********************
45. void show_array(int array[])
46. {
47.      for(int i=0;i<arraysize;i++)
48.          cout<<setw(4)<<array[i];
49.    cout<<endl;
50. }
51. //*********************
52. void bubble(int work[], int (*compare)(int,int))
53. {
54.      void swap(int* , int*); //function declaration
55.
56.      for(int i=1; i<arraysize; i++)
57.        for(int j=0; j<arraysize-1; j++)
58.           if( (*compare)(work[j],work[j+1]) )
59.                swap(&work[j],&work[j+1]);
60. }
61. //*********************
62. void swap(int* element1, int* element2)
63. {
64.        int temp;
65.        temp= *element1;
66.        *element1=*element2;
67.        *element2=temp;
68. }
69. //*********************
70. int ascending(int a,int b)
71. {
72.        return (b < a) ;  //swap if b is less than a
73. }
74. //*********************
75. int descending(int a, int b)
76. {
77.        return (b > a) ; //swap if b is greater than a
78. }
```

توضیح مثال : در این مثال الگوریتم مرتب‌سازی حبابی را برای مرتب‌سازی صعودی و مرتب‌سازی نزولی با استفاده از اشاره‌گر به توابع پیاده‌سازی کرده‌ایم. اگر کاربر مقدار **order** را برابر ۱ وارد کند مرتب‌سازی صعودی و اگر کاربر مقدار صفر را وارد کند مرتب‌سازی نزولی صورت می‌گیرد. هستهٔ عمل مقایسه که منجر به مرتب‌سازی نزولی یا صعودی می شود در دو تابع با نام‌های **ascending** و **descending** آورده شده است. تابع **swap** که نقش جابه‌جایی در مقادیر دو خانه از آرایه را دارد در خط ۵۴ در داخل تابع **bubble** اعلان شده است. در خط ۵۹ با فراخوانی تابع **swap** روبرو هستید. به‌خاطر داشته باشید که تنها نام آرایه‌است که یک اشاره‌گر می‌باشد لذا برای ارسال آدرس عنصر **j**- اُم آرایه در خط ۵۹ کد فوق از عملگر آدرس استفاده کرده‌ایم. نحوهٔ فراخوانی یک تابع با اشاره‌گرِ به تابع در خط ۵۸ این کد با آنچه که پیشتر فرا گرفتید کمی متفاوت است. لذا خط ۵۸ را به‌صورت زیر هم می‌توان نوشت:

```
if( compare(work[j],work[j+1]) )
```

۱۱-۳ : اشاره گرها و آرایه ها

متغیرهای پویا

از آنجا که یک اشاره‌گر می‌تواند آدرس محلی از حافظه را در خود نگهداری کند، از طریق آن آدرس می‌توان محتویات آن محل از حافظه را دستکاری کرد. بنابراین لزومی ندارد آدرس محلی که در اشاره‌گر قرار می‌گیرد، دارای نام باشد و تنها قرارگرفتن آدرس یک محل از حافظه در یک اشاره‌گر برای دستیابی و اعمال تغییرات برروی آن محل از حافظه، کفایت می‌کند. امتیاز این روش این است که پس از اتمام کار با آن محل از حافظه، می‌توان آن حافظه را آزاد کرد و به سیستم بازگرداند. این روش تخصیص حافظه را تخصیص پویای حافظه می‌گویند و به متغیرهایی که بدین روش ایجاد می‌گردند متغیرهای پویای می‌گویند، زیرا در زمان اجرای برنامه، ایجاد می‌شوند و سپس از بین خواهند رفت. این متغیرها از فضای **heap** اختصاص داده می‌شوند.

کاربرد عملگرهای new و delete در تخصیص حافظه به‌صورت پویا از heap سیستم

برای تخصیص حافظه به‌صورت پویا از عملگر **new** و برای برگردان آن به سیستم از عملگر **delete** به‌صورت زیر استفاده می‌شود، توجه کنید که می‌توانید در هنگام استفاده از **new** مقدار اولیه نیز به متغیر بدهید:

```
نوع  *      نام اشاره‌گر  ;
نام اشاره‌گر = new   نوع;
delete   نام اشاره‌گر  ;
```

مثال ۱۱-۷ : برنامهٔ که نحوهٔ عملکرد، عملگرهای **new** و **delete** را نشان می‌دهد.

```
1.  #include <iostream.h>
2.  #include <stdlib.h>
3.  void main(){
4.     int* x,cube;
5.     x = new int;//or write: x = new int(initial_value);
6.      if(!x) {
7.         cout << "\n Allocation failure.";
8.         exit(1);
9.      }
10.    cin>>*x;
11.     cube= *x * *x * *x;
12.      cout<<"\nThe cube of "<<*x<<" is : "<<cube<<endl;
13.    delete x;
14. }
```

توضیح مثال : در خط چهارم این کد متغیر **x** یک اشاره‌گر به نوع **int** تعریف شده است. در خط پنجم به واسطهٔ عملگر **new** فضای لازم جهت ذخیرهٔ یک عدد **int** از سیستم اخذ گردیده. اگر عملگر **new** به هر دلیل نتواند فضای لازم را از سیستم اخذ کند مقدار **NULL** در متغیر **x** قرار می‌گیرد. لذا در خطوط ۶ الی ۹ در صورت اخذ نشدن فضای لازم برنامه خاتمه می‌یابد. در خط یازدهم نیز ستاره‌های دوم و چهارم به معنای عملگر ضرب و دیگر ستاره‌ها به معنای عملگر مقدار می‌باشند. در خط ۱۳ نیز حافظهٔ اختصاص یافته به **x** را به سیستم باز گردانده‌ایم.

توابع malloc و free

توابع **malloc** و **free** به ترتیب جهت تخصیص پویای حافظه و آزادکردن حافظه در زبان **C** به‌کار می‌رفته‌اند و در زبان **C++** نیز قابل استفاده هستند. تابع **malloc** حافظه‌ای را از سیستم گرفته و آدرس آن به‌صورت یک اشاره‌گر برمی‌گرداند. این تابع که دارای صورت کلی زیر است در سر فایل **cstdlib** یا **malloc.h** قرار دارد:

```
نوع* نام اشاره‌گر = (نوع*)malloc(size);
```

در این کاربرد اگر تابع **malloc** بتواند حافظهٔ موردنظر را از سیستم اخذ کند آدرس آن به‌صورت یک اشاره‌گر نوع **void** بازمی‌گرداند و با تبدیل نوع موقت به‌کار رفته در جلوی نام تابع آدرس مورد نظر در اشاره‌گر مربوطه قرار می‌گیرد، و در صورتی که نتواند حافظهٔ مورد نظر را از سیستم اخذ کند مقدار **NULL** در اشاره‌گر مربوطه قرارمی‌گیرد. همچنین متغیر **size** حاوی مقدار بایتی است که باید به اشاره‌گر تخصیص داده شود. به‌عنوان مثال جهت اخذ فضای لازم برای ذخیرهٔ یک متغیر صحیح به‌صورت زیر عمل می‌کنیم:

```
int*  p = (int*) malloc( sizeof(int) );
```

در دستور فوق عبارت(**int***) اشاره‌گر نوع**void*** برگشت داده‌شده توسط **malloc** را به نوع**int*** تبدیل می‌کند. همچنین تابع **free** قادر است فضای اختصاص یافته به یک متغیرِ پویا را به سیستم بازگرداند. این تابع که در سر فایل **cstdlib** قرار دارد به‌صورت کلی زیر به کار می‌رود:

```
free( نام اشاره گر );
```

به‌عنوان مثال جهت بازگرداندن حافظهٔتخصیص یافته به اشاره‌گر**p** به‌صورت زیر عمل می‌کنیم:

```
free(p);
```

گرچه تخصیص پویای حافظه تنها برای ذخیرهٔ یک عدد کار منطقی نیست و تعریف یک متغیر به‌صورت معمولی بسیار آسان‌تر و مقبول‌تر است، امّا به واسطهٔ عملگرهای **new** و **delete** می‌توان آرایه‌هایی پویا، با هر اندازهٔ دلخواهی را تعریف کرد، و عمده کاربرد این دو عملگر در همین زمینه است. شاید بد نباشد که بدانید، حافظه‌ای که بدین روش از سیستم اخذ می‌گردد از قسمت **Heap** سیستم عامل گرفته می‌شود.

تعریف آرایه‌های یک بعدی به‌صورت پویا

چنانکه به‌خاطر دارید تعریف یک آرایه با اندازهٔ دلخواه کاربر امکانپذیر نبود، و لذا نوشتن دستوراتی به‌صورت زیر موجب بروز خطا در زمان کامپایل برنامه می‌شود.

```
int n;
 cin>>n;
  int array[n];
```

امّا به‌واسطهٔ اشاره‌گرها می‌توان آرایه‌ای را با طول دلخواه کاربر به‌صورت زیر تعریف کرد. با اجرای این دستورات آرایه‌ای با نام **ptr_array** به طول **n** عنصر تعریف می‌شود.

```
int  n;
 cin>>n;
  int* ptr_array = new int[n];
```

امّا دسترسی به اعضای آرایه‌هایی که بدین‌صورت تعریف می‌شوند به دو طریق امکانپذیر است:

۱. با استفاده از عملگر[ ] همانند آنچه که در مورد آرایه‌های معمولی انجام می‌دادیم. به‌عنوان مثال جهت دریافت عناصر آرایه‌ای که در بالا تعریف کردیم به‌صورت زیر می‌توان عمل کرد:

```
for(int i=0 ; i<n; i++)
   cin>>ptr_array[i];
```

۲. با استفاده از عملگر مقدار. چنانکه در فصل هفتم مطرح شد نام آرایه اشاره‌گری است به عنصر اول آرایه. شاید در آن زمان که فصل هفتم را مطالعه می‌کردید مفهوم این جمله چندان برای شما واضح نبوده است. امّا هم اکنون باید بتوانید مفهوم این جمله را به درستی درک کنید. در حقیقت زمانی که کنترل اجرای برنامه به دستور:

```
int array[5];
```

می‌رسد ۲۰ بایت از حافظه را به آرایهٔ **array** اختصاص می‌دهد و آدرس اولین خانه از این ۵ متغیر **int** را در نام آرایه، که خود اشاره‌گری ثابت از نوع **int** است، قرار می‌دهد. و دقیقاً به همین جهت است که وقتی نوشتیم **ptr_array[i]** عملگر[ ] دسترسی ما را به محتویات خانهٔ **i**- اُم آرایه فراهم ساخت، و لذا نحوهٔ دسترسی به عناصر آرایهٔ پویای **ptr_array** همانند نحوهٔ دسترسی به عنصر آرایهٔ **array** می‌باشد که پیشتر توضیح آن را در فصل هفتم داده‌ایم. اما به جهت یادآوری یک بار دیگر مفاهیم مطرح شده در بخش ۷-۱ را مرور می‌کنیم.

جهت دسترسی به عنصر **i**- اُم آرایهٔ پویای**ptr_array** به وسیلهٔ عملگر [ ] ، کنترل اجرای برنامه ابتدا به مکانی که اشاره‌گر **ptr_array** به آن اشاره می‌کند رفته و سپس **(i*4)** خانه جلو‌می‌رود تا به ابتدای ۴ بایت مربوط به عنصر **i**- اُم آرایهٔ پویای**ptr_array** دست پیدا کند. سپس با **load** کردن ۴ بایت بعد از ابتدای مکان فعلی که کنترل اجرای برنامه در آن قرار دارد به محتویات عنصر مورد نظر آرایه دست می‌یابد.

با توجه به این توضیحات اگر آرایه‌ای با نام **array** مطابق آنچه که در بالا تعریف کردیم داشته باشیم، به‌راحتی می‌توان با قرار دادن آدرس ابتدای آرایه در اشاره‌گر دیگری مثل **ptr** به محتویات آرایهٔ **array** با استفاده از عملگر [ ] دسترسی پیدا کرد، و لذا قطعه کد زیر به درستی عناصر آرایهٔ **array** را از کاربر دریافت می‌کند.

```
int* ptr;
ptr = array;
for(int i=0; i<5; i++)
    cin>>ptr[i];       //same to cin>>array[i];
```

امّا با توجه به مطالبی که در خصوص عمل محاسباتی جمع برروی اشاره‌گرها در بخش ۱۱-۱ بیان داشتیم حلقهٔ **for** بالا را به‌صورت زیر هم می‌توان نوشت:

```
for(int i=0; i<5; i++)
    cin>>*(ptr + i);  //same to cin>>*(array + i);
```

چنانکه در این مثال می‌بینید عبارت (ptr + i)* دسترسی ما را به محتویات مکانی که**ptr[i]** به آن دسترسی دارد، امکانپذیر می سازد. بنابراین به همین روش نیز می‌توان به عناصر آرایهٔ پویای **ptr_array** که پیشتر تعریف کردیم دسترسی پیدا کرد. زیرا دیدید که هیچ تفاوتی بین نام یک آرایه و یک اشاره‌گر از همان نوع وجود ندارد.

```
for(int i=0; i<n; i++)
  cin>>*(ptr_array + i);
```

بازگرداندن فضای یک آرایهٔ یک بعدی پویا

اگر در برنامهٔ خود با استفاده از عملگر **new** ، مقدار زیادی حافظه از سیستم اخذ کنید آنگاه تمام فضای امکانپذیر موجود در**RAM** ، در اختیار یک برنامه قرار می گیرد و ممکن است سیستم از کار بیفتد (به اصطلاح سیستم **Hang** می‌کند). برای اطمینان از کاربرد مطمئن عملگر **new** ، باید بین عملگر **new** و عملگر **delete** متناظر با آن هماهنگی لازم را به‌وجود آورید تا حافظهٔ اخذ شده را به سیستم عامل برگردانید. به‌طور کلی هرگاه که با مقداری حافظه که به‌طور پویا از سیستم اخذ کرده‌اید کارتان به اتمام رسید، بهتر است آن را بی درنگ به سیستم‌عامل برگردانید. نحوهٔ بازگرداندن حافظهٔ تخصیص یافته به یک آرایهٔ یک‌بعدی‌پویا به‌صورت زیراست.

```
delete[]   نام اشاره‌گر ;
```

مثال ۱۱-۸ : برنامه‌ای که **n** عدد را از ورودی خوانده و سپس مقدار میانه را در بین این **n** عدد پیدا می‌کند. میانه عنصری است که نیمی از اعداد از آن بزرگتر و نیمی دیگر از آن کوچکتر باشند. بنابراین اگر **n** فرد باشد میانه عبارت است از عنصر وسط و اگر **n** زوج باشد میانه عبارت است از میانگین دو عدد میانی. لذا به نحوهٔ یافتن میانه در خطوط ۵۹ و ۶۱ توجه کافی نمایید.

```
1.  //This program finds the median of n numbers.
2.  #include <iostream>
3.  #include <cstdlib>
4.  using namespace std;
5.  void  input(int*, int);
6.  void output(int*, int);
7.  void bubble(int*, int);
8.  void median(int*, int, float*);
9.  int main()
10. {
11.    int* ptr_arr, n;
12.    float mead;
13.    cout<<"Enter number of items :";
14.     cin>>n;
15.    ptr_arr = new int[n];
16.     if(!ptr_arr)
17.       {
18.           cout<<" Allocation failure!";
19.           cin.get();
20.           exit(1);
21.       }
```

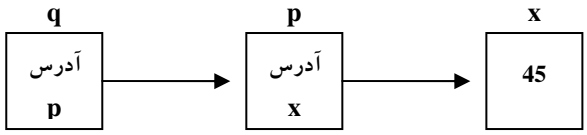
```
22.       input(ptr_arr, n);
23.       bubble(ptr_arr, n);
24.        system("cls");
25.        cout<<"\n Sorted data are:";
26.       output(ptr_arr, n);
27.       median(ptr_arr, n, &mead);
28.        cout<<"\n Median = "<<mead;
29.       delete[] ptr_arr;
30.        cout<<endl;
31.    return 0;
32. }//end of main function
33. //***********************
34. void input(int* p, int n)
35. {
36.     for(int i=0; i<n; i++)
37.       {
38.           cout<<"Enter number "<<(i + 1)<<": ";
39.            cin>>*(p + i);
40.       }
41. }
42. //*********************************
43. void bubble(int* ptr_arr, int len)
44. {
45.     int i, j, temp;
46.       for(i=len-1; i>0; i--)
47.           for(j=0; j<i; j++)
48.              if(*(ptr_arr + j) > *(ptr_arr + j + 1))
49.                {
50.                     temp = *(ptr_arr + j);
51.                     *(ptr_arr + j) = *(ptr_arr + j + 1);
52.                     *(ptr_arr + j + 1) = temp ;
53.                 }//end of if
54. }
55. //************************************
56. void median(int* p, int n, float* mead)
57. {
58.      if(n % 2 == 0)
59.           *mead =(float)(*(p+((n-1)/2))+*(p+(n/2))) / 2;
60.      else
61.           *mead = *(p+(n-1)/2);
62. }
63. //************************
64. void output(int* p, int n)
65. {
66.      for(int i=0; i<n; i++)
67.          cout<<*(p + i)<<" ";
68. }
```

اشاره‌گر به اشاره‌گر

چنانکه دیدید هر اشاره‌گر قادر است آدرس یک متغیر هم نوع خود را، نگهداری کند. حال اگر متغیر اشاره‌گر دیگری پیدا شود که قادر باشد آدرس اشاره‌گر دیگری را در خود ذخیره کند، اشاره‌گر به اشاره‌گر خوانده می‌شود. جهت تعریف متغیری به‌صورت اشاره‌گر به اشاره‌گر باید از دو عملگر * در هنگام تعریف آن استفاده کنیم. به‌عنوان مثال قطعه کد زیر را در نظر بگیرید:

```
int    x = 45;
int*   p = &x;
int** q = &p;
```

چیزی که با اجرای دستورات فوق رخ می‌دهد در شکل زیر نشان داده شده است:



شکل ۱۱- ۵ : اشاره‌گر به اشاره‌گر

امّا چنانکه پیشتر دیدید، جهت دستیابی به محتویات مکانی که**p** به آن اشاره می‌کرد از عملگر مقدار یعنی علامت ستاره بهره می‌گرفتیم و (**p***) امکان دستیابی ما را به محتویات **x** فراهم می‌کرد. حال برای دستیابی به محتویات متغیر **x** توسط اشاره‌گر به اشاره‌گر **q** باید از دو عملگر ستاره به‌عنوان عملگر مقدار به‌صورت (**q****) استفاده کنیم. این عبارت بدین‌صورت ارزیابی می‌شود که (**q***) محتویات مکانی را که **q** به آن اشاره می‌کند را برمی‌گرداند، که همان آدرس**x** می‌باشد. لذا در هنگام ارزیابی به جای **q*** ، اشاره‌گر**p** قرار می‌گیرد. حال با عبارت (**p**)* روبرو هستیم که در نهایت دسترسی ما را به محتویات مکانی که **p** به آن اشاره می‌کند یعنی محتویات متغیر **x** ، امکانپذیر می‌سازد.

مثال ۱۱-۹: برنامه‌ای که نحوهٔ به‌کارگیری اشاره‌گر به اشاره‌گر را نشان می‌دهد.

```
1. //This program uses pointer to pointer.
2. #include <iostream>
3. using namespace std;
4. void main()
5. {
6.    int x,* p,** q;
7.        x = 10 ;
8.        p = &x ;
9.        q = &p ;
10.    cout<<"x value is : "<<*p;
11.    cout<<"\nAddress of x is : "<<p;
12.    cout<<"\nAddress of p is : "<<q;
13.    cout<<"\nThe q points to value : "<<**q ;
14.    cout<<endl;
15. }
```

امّا کاربرد اصلی و مهم اشاره‌گر به اشاره‌گر در تعریف آرایه‌های دو بعدی به‌صورت پویا است. به‌عنوان مثال در زیر قطعه کدی را مشاهده می‌کنید که نتیجهٔ اجرای آن تعریف یک آرایهٔ دو بعدی با نام **q** و با ابعاد **m** سطر و **n** ستون است:

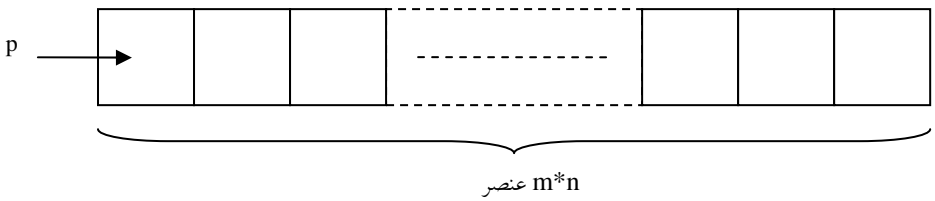
```
1. int  m, n;
2. cin>>m>>n;
3. int* p;
4. int** q;
5.   p=new int [m*n];
6.   q=new int* [m];
7.      for(int i=0; i<m; i++ )
8.            q[i] = p + n*i;
```

دسترسی به عناصر این آرایه به‌صورت **q[i][j]** امکان پذیر است. به‌عنوان مثال برای دریافت عناصر این آرایه از کاربر به‌صورت زیر عمل می‌کنیم:

```
for(int i=0; i<m; i++)
  for(int j=0; j<n; j++)
       cin>>q[i][j];
```

امّا قبل از آنکه به بحث در خصوص نحوهٔ عملکرد این کد در تخصیص پویای حافظه و ایجاد یک آرایهٔ دو بعدی پردازیم باید توجه شما را به مطالبی که در خصوص نحوهٔ ایجاد یک آرایهٔ دو بعدی به‌صورت معمول در **C++** در فصل هفتم مطرح ساختیم جلب کنیم. چنانکه پیشتر مطرح شد یک آرایهٔ دو بعدی در حافظه به‌صورت یک آرایهٔ یک بعدی پیاده‌سازی می‌شود، امّا هنگامی که در خصوص آرایهٔ دو بعدی **array** می‌نویسیم **array[i][j]** عبارت **array[i]** ما را به ابتدای ردیف **i**- اُم و به دنبال آن عبارت **[j]** ما را به عنصر موجود در این ردیف و ستون **j**-اُم منتقل می‌کند، که نحوهٔ پردازش و آدرس‌یابی آن پیشتر در فصل هفتم به‌طور کامل توضیح داده شده است. امّا عملکرد کدی که در بالا به‌منظور ایجاد یک آرایهٔ دو بعدی به‌صورت پویا دیدید چنین است:

در خطوط اول و دوم طول و عرض آرایه از کاربر دریافت شده که **m** معرف تعداد سطرها و **n** معرف تعداد ستون‌ها است. در خط سوم اشاره‌گر **p** یک اشاره‌گر به نوع **int** و **q** اشاره‌گری به اشاره‌گری از نوع **int** تعریف شده است. از آنجا که یک آرایهٔ دو بعدی با **m** سطر و **n** ستون دارای **(m*n)** عنصر است در خط پنجم آرایه‌ای یک بعدی با **(m*n)** عنصر **int** تعریف کرده‌ایم که **p** به اولین خانهٔ آن اشاره می‌کند.

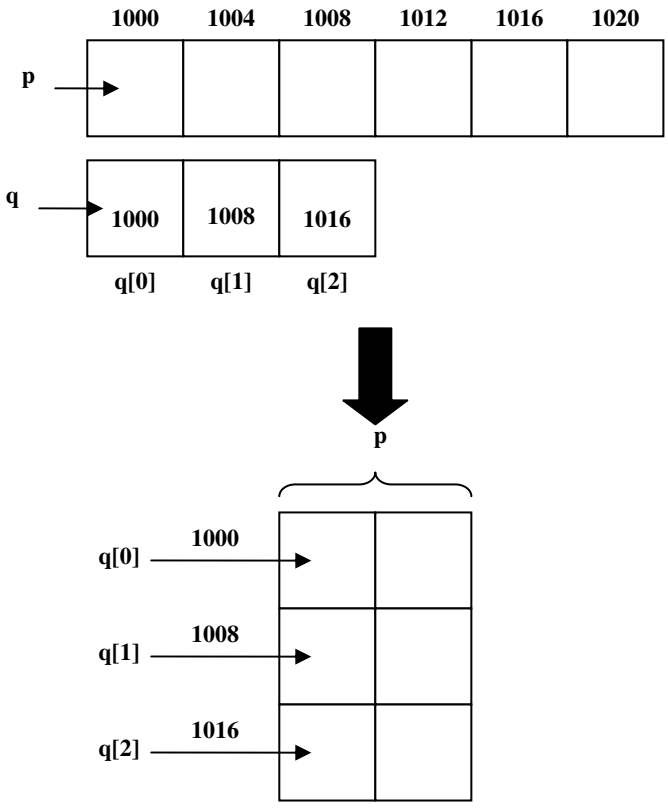


شکل ۱۱- ۶: تخصیص فضای لازم برای یک آرایهٔ دو بعدی پویا در حافظه

حال اگر بتوانیم آدرس عناصر ابتدای هر سطر را در اشاره‌گر دیگری ذخیره کنیم به راحتی با توجه به آنچه در خصوص نحوهٔ آدرس‌یابی آرایه‌های دو بعدی غیر پویا پیشتر بیان داشتیم قادر خواهیم بود با دو اندیس **i** و **j** به عناصر آرایهٔ پویای **p** دسترسی پیدا کنیم. لذا چون **m** سطر داریم باید آرایه‌ای از اشاره‌گر به اشاره‌گر و به تعداد **m** عنصر تعریف کنیم تا بتوانیم آدرس عناصر ابتدای هر سطر را در عناصر آن ذخیره کنیم. لذا در خط ششم این کد آرایه‌ای از اشاره‌گر به اشاره‌گر(یعنی آرایهٔ **q** از نوع **int*** می‌باشد) و به طول **m** عنصر تعریف کرده‌ایم.

```
q=new int* [m];
```

و در نهایت در خطوط هفتم و هشتم آدرس‌عناصر ابتدای هر سطر آرایهٔ **p** را به‌واسطهٔ اشاره‌گر **p** که حاوی آدرس نخستین عنصر آرایهٔ پویای **(m*n)** عنصری است در عناصر آرایهٔ **q** ذخیره می‌کنیم. اگر **m** را برابر ۳ و **n** را برابر ۲ فرض کنید چنین نتیجه‌ای پس از اجرای حلقهٔ **for** به‌دست خواهد آمد:



شکل ۱۱-۷ : ایجاد یک آرایهٔ دو بعدی پویا در حافظه

لذا هنگامی که بنویسیم **q[i][j]** عبارت **q[i]** آدرس عنصر ابتدای سطر **i**-اُم را به‌دست می‌دهد و با قرار گرفتن عبارت **[j]** دستیابی به عنصری که در سطر **i**- اُم و ستون **j**- اُم قرار دارد میسر می‌شود.

اشاره‌گرهایی با مراتب بالاتر

با افزایش تعداد ستاره‌ها در تعریف اشاره‌گرها می‌توان اشاره‌گرهایی با مراتب بالاتر، مثل اشاره‌گر به اشاره‌گر به اشاره‌گر و... تعریف‌کرد. لذا مطابق روش فوق نیز می‌توان آرایه‌های سه بعدی، چهار بعدی و... را به‌صورت پویا تعریف کرد. در زیر کد مربوط به تعریف آرایه‌ای سه بعدی را به‌صورت پویا مشاهده می‌کنید امّا تفسیر نحوهٔ عملکرد این کد به‌عنوان تمرین بر عهدهٔ‌خوانندگان محترم می‌باشد.

کار در کلاس ۱۱-۱ :

قطعه برنامهٔ زیر آرایه ای سه بعدی به صورت **r[n][n][n]** *را به‌طور پویا تعریف می کند. ضمن تشریح نحوهٔ عملکرد این کد، آن را برای آرایه ای پویا با ابعاد* **r[m][n][k]** *تعمیم دهید.*

```
1.  int i, j, n;
2.   cin>>n;
3.  int*   p;
4.  int**  q;
5.  int*** r;
6.  p = new int [n*n*n];
7.  q = new int*  [n*n];
8.  r = new int**   [n];
9.  //assign first of each row in q
10.   for(i=0; i<n*n; i++)
11.       q[i] = p + i*n;
12. //assign first of each page in r
13.   for(i=0; i<n; i++)
14.       r[i] = q + i*n;
15. //assign first of each row of each page in r
16.   for(i=0; i<n; i++)
17.     for(j=0; j<n; j++)
18.         r[i][j]=p + i*n*n + j*n;
```

آزاد کردن حافظهٔ آرایه‌های چند بعدی پویا

جهت بازگرداندن حافظهٔ تخصیص‌یافته به یک آرایهٔ چند بعدی پویا، باید آرایه‌های پویایی را که ساختیم به ترتیب عکس ساخته شدن با عملگر **delete** و یا ترجیحاً به‌وسیلهٔ تابع **free** آزاد کنیم. به‌عنوان مثال جهت آزاد کردن آرایهٔ **r[n][n][n]** که در کاردرکلاس فوق ایجاد کردیم باید به‌صورت زیر عمل کنیم:

```
delete[] r;
delete[] q;
delete[] p;
```

ارسال آرایه‌ها به توابع به واسطهٔ اشاره گر

در فصل نهم با ارسال آرایه‌ها به‌عنوان آرگومان، به توابع آشنا شدید. در آنجا از علامت آرایه یعنی [ ] برای مشخص کردن یک آرایه استفاده کردیم. اما حال که نحوهٔ تعریف اشاره‌گرها را فراگرفتید می‌توانید به‌جای استفاده از علامت کروشه از اشاره‌گرها استفاده کنید. به‌عنوان مثال توابع **fun1** و **fun2** و **fun3** که در زیر اعلان شده‌اند به ترتیب آرایه های یک بعدی، دو بعدی و سه بعدی را به‌عنوان پارامتر ورودی می‌پذیرند.

```
void fun1(int*);
int  fun2(float**);
char fun3(double***);
```

به‌خاطر داشته باشید آرایه‌های چند بعدی پویا را تنها از همین طریق قابل ارسال به توابع هستند و نمی‌توان از عملگر [ ] برای تعریف آرگومان‌هایی به‌صورت آرایهٔ چند بعدی پویا استفاده کرد. در هنگام فراخوانی این توابع نیز تنها ذکر نام آرایه یا اشاره‌گر معادل آنها، کفایت می‌کند.

عملگر const و اشاره‌گرها

استفاده از عملگر **const** به همراه اعلان اشاره‌گرها می‌تواند باعث پیچیدگی برنامه شود زیرا با توجه به مکان قرار گرفتن عملگر **const** ممکن است یکی از دو حالت زیر پیش بیاید. به دستورهای زیر را توجه کنید:

```
const  int*  ptr_con_int ;//It's a pointer to constant int
int*  const  con_ptr_int ;//It's a constant pointer to int
```

حالت اول تعریف اشاره‌گری با نام **ptr_con_int** است که به یک ثابت از نوع صحیح می‌تواند اشاره کند. به این نوع اشاره‌گر، *«ثابت اشاره‌گر»* می‌گویند. لذا بعد از اعلان اول نمی‌توان مقدار آنچه را که **ptr_con_int** به آن اشاره می‌کند تغییر داد. البته این بدان‌معنا نیست که این اشاره‌گر تنها قادر به ذخیرهٔ ثوابتی از نوع صحیح است. بلکه به هر متغیری که اشاره کند مقدار آن قفل شده و قادر به تغییر مقدار آن نیستید. نمونه‌ای از این کاربرد عملگر **const** را در اعلان توابع مشاهده کرده‌اید. به‌عنوان مثال با مراجعه به مستندات **MSDN** در خصوص اعلان تابع **strcpy** خط زیر را خواهید دید:

```
char* strcpy( char* strDestination, const char* strSource );
```

اگر به آرگومان دوم این تابع توجه کنید، متوجه می‌شوید که تابع **strcpy** قادر نیست هیچگونه تغییری را بر پارامتر دوم خود اعمال کند، اما قادر است پارامتر اول خود را تغییر دهد، زیرا از نوع **const** اعلان نشده است. لذا اگر عملگر **const** را به این روش با آرگومان‌هایی از تابع که به‌صورت اشاره‌گر هستند به‌کار ببرید، در حقیقت امکان اعمال تغییرات بر پارامترهای ورودی از طریق فراخوانی با آدرس را حذف کرده‌اید.

اما در حالت دوم اشاره‌گر **con_ptr_int** قادر است در طول عمر خود تنها به یک نقطه اشاره کند. به عبارت دیگر آدرسی که در این اشاره‌گر قرار می‌گیرد قفل می‌شود، و شما قادر به تغییر آن نیستید. اما می‌توانید محتویات مکانی را که این اشاره‌گر به آن اشاره می‌کند را تغییر دهید. به این نوع اشاره‌گر، *«اشاره‌گر ثابت»* گفته می‌شود. به‌عنوان مثال نام هر آرایه یک اشاره‌گر ثابت به خانهٔ اول آرایه است. بدین معنا که آدرس موجود در نام آرایه را نمی‌توان تغییر داد. بنابراین

چه خوب است که پس از تعریف یک آرایهٔ چند بعدی به‌صورت پویا مانند **r[n][n][n]** که در کار در کلاس ۱۱-۱ تعریف کردیم در نهایت آدرس **r** را در یک اشاره‌گر ثابت ذخیره کنیم، تا دقیقاً تعریف آرایه‌های پویا نیز مانند آرایه‌های معمولی خود **C++** بشود. به‌عنوان مثال در خصوص آرایهٔ مذکور می‌توان چنین عمل کرد:

```
int*** const array = r;
```

و در این صورت به جای استفاده از عبارت **r[i][j][k]** از عبارت **array[i][j][k]** باید استفاده کنیم.

از طرفی درحین نوشتن برنامه‌های خود به این نکته توجه داشته باشید که، هیچگاه یک اشاره‌گر ثابت را نمی‌توان تعریف کرد مگر آن که در زمان تعریف به آن مقدار اولیه داده شود (یعنی آدرس مورد نظر به آن انتساب داده شود). به‌عنوان مثال دستور زیر را در نظر بگیرید:

```
int*  const  con_ptr_int ;
```

اگر خط فوق را در برنامه‌ای بنویسید و آن را کامپایل کنید با خطای زیر مواجه خواهید شد:

error C2734: 'ptr_con_int' : const object must be initialized if not extern

البته اگر کلاس حافظهٔ اشاره‌گر **con_ptr_int** را از نوع **extern** تعریف کنید نیازی به مقدار اولیه دادن در زمان تعریف نیست، که البته از بحث فعلی ما خارج است. از طرفی اگر پس از تعریف صحیح یک اشاره‌گر ثابت بخواهید آدرس دیگری را به آن نسبت دهید با خطای زیر مواجه خواهید شد:

error C2166: l-value specifies const object

نکتهٔ آخر آن که اگر اشاره‌گری با نام **ptr** را به‌صورت زیر تعریف کنیم:

```
const int* const ptr = p ;
```

این تعریف از راست به چپ به این صورت خوانده می شود: " **ptr** اشاره‌گر ثابت به ثابت صحیح است ". هنگام تلاش در جهت تغییر داده‌ای که **ptr** به آن اشاره می‌کند و نیز هنگام تلاش در جهت تغییر آدرس ذخیره شده در اشاره گر ثابت **ptr** با پیغام خطا مواجه خواهید شد.

اشاره‌گرها و رشته‌های زبان C

از آنجا که رشته‌های زبان **C** به‌صورت آرایه پیاده‌سازی می‌شوند، تمامی مطالبی که تا اینجا در خصوص ارتباط آرایه‌ها و اشاره‌گرها مطرح ساختیم در خصوص رشته‌های زبان **C** نیز صادق است. پیشتر دیدید که برای تعریف یک رشته از نوع رشته‌های زبان **C** می‌توانستیم به‌صورت زیر عمل کنیم:

```
char str[] = "Address";
```

در این تعریف طول رشته توسط کامپایلر تشخیص داده می‌شود و طول **str** نیز به همان میزان تنظیم می‌گردد. این رشته را به‌صورت زیر نیز می‌توان تعریف کرد که هیچ تفاوتی از نظر نحوهٔ عملکرد با یکدیگر ندارند:

```
char* str = "Address";
```

درحقیقت عبارت ***char** معرف یک رشته از زبان **C** می‌باشد، که در اعلان بسیاری از توابع رشته‌ای به‌کارمی‌رود.

تعریف آرایه‌ای از اشاره‌گرها (آرایه‌های دندانه‌ای)

اگر یک آرایه از انواع ساده‌ای چون **int** یا **float** و امثال آنها باشد، آنچه که در خانه‌های آن آرایه ذخیره می‌گردد اعدادی از نوع **int** یا **float** و... خواهد بود. حال حالتی را فرض کنید که آرایه‌ای از نوع اشاره‌گر داشته باشیم. یعنی به جای آن که خانه‌های آرایه متغیرهایی از نوع **int** باشند، متغیرهایی از نوع ***int** باشند. در این حالت در خانه‌های آرایه می‌توان آدرس‌های متغیرهایی از نوع **int** را ذخیره کرد. در حقیقت شما پیشتر از اینگونه آرایه‌ها استفاده کرده‌اید! آیا به خاطر می‌آورید!؟

درست حدس زدید، هنگامی‌که آرایه‌های چند بعدی را تعریف می‌کردیم، از اینگونه آرایه‌ها استفاده کردیم. اما مهمترین کاربرد آرایه‌هایی از اشاره‌گرها، زمانی است که بخواهیم آرایه‌هایی دندانه‌ای جهت ذخیرهٔ رشته‌ها بسازیم. با استفاده از آرایه‌ای از اشاره‌گرها می‌توان به هر رشته به میزان مورد نیازش حافظه اختصاص داد. به‌عنوان مثال به تعریف زیر توجه کنید:

```
char* str_teeth[]={"Hearts", "Diamonds", "Clubs", "Spades"};
```

شکلی‌که با اجرای این دستور در حافظه ایجاد می‌شود مطابق قسمت (الف) شکل ۱۱-۸ است. حال اگر قرار باشد برای ذخیرهٔ این چهار رشته یک آرایهٔ دو بعدی از نوع **char** تشکیل بدهیم، باید تعداد ستون‌های تمامی ردیف‌ها را برابر ۹ عدد در نظر بگیریم. زیرا بزرگترین رشته در بین این چهار رشته دارای ۸ کاراکتر است به انضمام کاراکتر **'\0'** که در انتهای رشته‌های زبان **C** قرار می‌گیرد، مجموعاً نیازمند وجود ۹ کاراکتر در هر ردیف هستیم. شکلی که با اجرای دستور زیر در حافظه ایجاد می‌شود مطابق قسمت (ب) شکل ۱۱-۸ خواهد بود.

```
char str_teeth[4][8]={"Hearts","Diamonds","Clubs","Spades"};
```

H	e	a	r	t	s	ø		
D	i	a	m	o	n	d	s	ø
C	l	u	b	s	ø			
S	p	a	d	e	s	ø		

(الف) : آرایه‌ای از اشاره‌گرها در حافظه

H	e	a	r	t	s	ø	?	?
D	i	a	m	o	n	d	s	ø
C	l	u	b	s	ø	?	?	?
S	p	a	d	e	s	ø	?	?

(ب): آرایه‌ای دو بعدی از کاراکترها در حافظه

شکل ۱۱- ۸ : تفاوت آرایه‌ای از اشاره‌گرها با آرایه‌های معمولی

یکی از کاربردهای آرایه‌ای از اشاره‌گرها در آرگومان‌های تابع **main** بود. شاید آن زمان مفهوم آرگومان دوم تابع **main** را به خوبی درک نمی‌کردید. اما حالا به‌راحتی می‌توانید در خصوص **argv** قضاوت کنید. در پروژه‌ها از شما خواهیم خواست که با استفاده از حالت (الف) در بالا یک بازی با ورق را شبیه‌سازی کنید.

مثال ۱۱-۱۰: در برنامهٔ زیر مثال ۷-۹ را یکبار دیگر به واسطهٔ آرایه‌ای از اشاره‌گرها پیاده‌سازی کرده‌ایم.

```
1.  //This program uses an array of pointers to string.
2.  #include <iostream>
3.  using namespace std;
4.  int main()
5.  {
6.   const int Days=7;
7.    char* arr_ptr[Days]= {"Sunday","Monday",
8.     "Tuesday","WednesDay","Thursday","Friday","Saturday"};
9.      for(int i=0;i<Days;i++)
10.          cout<<arr_ptr[i]<<endl;
11.   return 0;
12. }
```

طراحی منو و کلیدهای تابع

یکی از مهمترین کاربردهای اشاره‌گرهای تابع که در ارتباط با آرایه‌هایی از اشاره‌گرها نیز هست، در سیستم‌های راه‌اندازی منویی است. به‌عنوان مثال می‌توان به کاربر پیغام داد که یکی از کلیدهای **F1** تا **F3** را بفشارد و آنگاه کار هر منو توسط توابع مختلفی انجام شود. در این حالت اشاره‌گر به تابع در آرایه‌ای از اشاره‌گرها به توابع ذخیره می‌شوند. سپس گزینهٔ وارد شده از سوی کاربر به‌صورت اندیس در آرایه به‌کار گرفته می‌شود، و اشاره‌گر داخل آرایه برای احضار تابع به کار می‌رود. که نمونه‌ای از این کاربرد را در مثال ۱۱-۱۱ مشاهده می‌کنید.

نکتهٔ قابل توجه دیگر آنکه برخی از کلیدهای روی صفحه کلید مثل کلیدهای تابع (کلیدهای **F1** تا **F12**) یا کلیدهای جهتی یا کلیدهایی که به‌صورت ترکیبی عمل می‌کنند مثل **Alt + KEY** ، دارای کد اسکی مستقل نیستند. لذا هنگامی‌که یکی از این کلیدها فشرده می‌شوند، میکروپروسسور داخل صفحه کلید یک کد گسترش یافته که شامل دو بایت است می سازد. بایت با ارزش شامل صفر و بایت کم ارزش دارای یک کد مختص آن کلید است. لذا هنگامی که می‌خواهید دادهٔ ارسالی از طرف چنین کلیدهایی را بخوانید باید دوبار عمل خواندن کاراکتر را انجام دهید. بدین ترتیب که اگر بار اول که کاراکتری را می‌خوانید آن کاراکتر صفر بود، به این معنا است که یکی از این کلیدهای ویژه فشرده شده است. سپس باید بایت دوم را بخوانید که شامل یک عدد در محدودهٔ **char** است. به‌عنوان مثال کد مربوط به کلید **F1** برابر ۵۹ می‌باشد. لذا اگر کاربر این کلید را بفشارد عدد اول صفر و عدد دوم برابر ۵۹ خواهد بود، و یا به‌عنوان مثالی دیگر اگر کلید **ALT + F1** فشرده شود دوباره عدد اول صفر و عدد دوم ۱۰۴ خواهد بود. اگر به پیوست شمارهٔ ۵ مراجعه کنید کدهای مربوط به تمامی کلیدهای ترکیبی و غیرترکیبی را مشاهده خواهید کرد. در مثال زیر نحوهٔ کاربرد این کلیدها نشان داده شده است.

مثال ۱۱-۱۱: برنامه‌ای که نحوهٔ استفاده از آرایه‌ای از اشاره‌گرهای به توابع را نشان می‌دهد. در این برنامه اگر کاربر یکی از کلیدهای **F1** تا **F3** را بزند تابع مربوطه اجرا می‌شود و با کلید **Esc** برنامه خاتمه می‌یابد.

```
#include <iostream>
#include <conio.h>
void func1(int);
void func2(int);
void func3(int);
using namespace std;
void main()
{
   void (*f[3])(int) = {func1, func2, func3};
    cout<<"Press one of function keys F1 or F2 or F3 to
           run menu :";
    char choice=getch();
     while(1)
       {
            if(choice == 27)
                 break;
            else if(choice==0)
             {
                  choice=getch();
                  choice-=59;
                 if(choice>=0 && choice <3)
                  {
                    f[choice](choice+59);
                    cout<<"Press one of function keys
                           F1 or F2 or F3 to run menu :";
                  }
             }
          choice=getch();
        }
    cout<<"\nProgram execution completed."<<endl;
}//**************
void func1(int x)
{
    cout<<"\nYou press F1 but we received "<<x<<endl;
}//**************
void func2(int x)
{
    cout<<"\nYou press F2 but we received "<<x<<endl;
}//**************
void func3(int x)
{
    cout<<"\nYou press F3 but we received "<<x<<endl;
}
```

۱۱-۴: مرجع (reference)

چنانکه دیدید دو روش برای احضار توابع وجود دارد، که عبارت‌اند از *روش فراخوانی با مقدار* و *روش فراخوانی با آدرس*. هنگامی که پارامتری از طریق مقدار منتقل می‌شود، به اصطلاح یک کپی از آن پارامتر در آرگومان متناظر در داخل تابع ایجاد می‌گردد. لذا تغییر در مقدار کپی، هیچ تأثیری در مقدار متغیر اصلی در تابع احضار کننده ندارد. با این روش از احضار توابع، در فصل نهم آشنا شدید. از طرفی با روش فراخوانی توابع با آدرس نیز پیشتر در همین فصل آشنایی پیدا کردید، و چنانکه در بخش ۱۱-۲ دیدید، به واسطهٔ اشاره‌گرها می‌توان آدرس پارامترها را به تابع ارسال کرد و از این طریق تغییراتی که در داخل تابع برروی مقادیر ورودی صورت می‌گیرد را به متغیرهای اصلی داخل تابع احضار کننده نیز تأثیر داد. اما متأسفانه استفاده از اشاره‌گرها همراه با پیچیدگی‌هایی است که برنامه‌نویسان رغبت چندانی به استفاده از آنها ندارند، لذا پس از استانداردسازی C++ روش دیگری برای کار با آدرس متغیرها ابداع شد که در زیر به بررسی آن می‌پردازیم.

تعریف مرجع

مرجع‌ها همانند اشاره‌گرها قادر به ذخیرهٔ آدرس متغیرها هستند. جهت تعریف یک مرجع از عملگر & به فرم کلی زیر استفاده می‌کنیم:

نوع & نام مرجع = نام متغیر ;

به‌عنوان مثال دستورات زیر را در نظر بگیرید:

```
int  i=4;
int& r=i;
cout<<r;
```

در تکه کد فوق مرجعی با نام **r** و از نوع **int** تعریف شده که آدرس متغیری از نوع صحیح با نام **i** در آن ذخیره شده است. سپس با اجرای دستور **cout** مقدار **4** به‌عنوان مقدار متغیر **i** چاپ می‌شود، زیرا **r** حاوی آدرس این متغیر است. در این بین تفاوت‌هایی را بین مرجع و اشاره‌گرها می‌توان برشمرد که به قرار زیر است:

۱. عمده‌ترین تفاوت اشاره‌گر و مرجع در این است که اشاره‌گر یک متغیر است و لذا می‌توان آدرس ذخیره شده در آن را تغییر داد. اما مرجع چنین نیست و می‌توانید مرجع را مثل اشاره‌گرهای ثابت در نظر بگیرید که آدرس داخل آن قفل شده است. لذا مرجع را نمی‌توان جدا از متغیر تعریف کرد، زیرا آدرس مکانی که مرجع قرار است به آن اشاره کند از زمان تعریف تا انتهای عمر آن مرجع ثابت بوده و تغییر ناپذیر است. بنابراین قرار دادن دستور زیر در یک برنامه تولید خطا می‌کند:

```
int& r;
```

و چنانکه پیشتر در خط دوم کد فوق دیدید، در هنگام تعریف یک مرجع متغیری که قرار است به آن ارجاع شود باید به آن مرجع نسبت داده شود، تا آدرس متغیر مذکور در داخل مرجع قرار داده شود.

۲. جهت نسبت دادن آدرس یک متغیر به یک مرجع احتیاجی به قرار دادن عملگر آدرس قبل از نام متغیر نیست.

۳. هنگامی که می‌خواهیم به محتویات جایی که یک مرجع به آن ارجاع می‌کند دسترسی پیدا کنیم نیز نیازی به قرار دادن عملگر مقدار نیست، و لذا همین که در دستوری، نام مرجع را ذکر کنیم دسترسی ما به محتویات مکانی که مرجع به آن ارجاع می‌کند امکانپذیر می‌شود. لذا دسترسی به آدرس ذخیره شده در داخل مرجع نیز امکانپذیر نمی‌باشد، اما می‌توان به آدرس خود مرجع دست پیدا کرد که از نظر برنامه‌نویسی هیچ ارزش کاربردی ندارد.

مثال ۱۱-۱۲: برنامه‌ای که چگونگی به‌کارگیری مرجع را نشان می‌دهد.

```
1.  //This program shows usage of refrence parameters.
2.  #include <iostream.h>
3.  int main()
4.  {
5.    int a;
6.    int& ref = a; //independent reference
7.    a = 10;
8.    cout<<a<<"\t"<<ref<<endl;
9.    ref = 100;
10.   cout<<a<<"\t"<<ref<<endl;
11.   int b = 1000;
12.   ref = b;      // this puts b's value into a
13.   cout<<a<<"\t"<<ref<<endl;
14.   ref--;        //decrement a
15.   cout<<a<<"\t"<<ref<<endl;
16.  return 0;
17. }
```

توضیح مثال: خروجی این برنامه به‌صورت زیر خواهد بود:

```
10          10
100         100
1000        1000
999         999
```

محدودیت‌های مرجع

در عین حالی که با استفاده از مرجع، فراخوانی توابع به شیوهٔ ارجاع (از طریق آدرس)، بسیار راحت‌تر امکانپذیر است، در استفاده از مرجع با محدودیت‌هایی روبرو هستیم که عبارتند از:

۱. نمی‌توان مرجع به مرجع (همانند اشاره‌گر به اشاره‌گر) تعریف کرد، لذا با استفاده از مراجع قادر به تعریف‌آرایه یا رشته نیستیم.

۲. تعریف آرایه‌ای از مرجع نیز امکانپذیر نیست.

۳. تعریف اشاره‌گری به مرجع نیز امکانپذیر نیست، اما تعریف مرجعی به اشاره‌گر امکانپذیر است.

مرجع در رابطه با توابع دارای دو کاربرد عمده است که در ادامه به بررسی آنها می‌پردازیم.

مرجع به‌عنوان آرگومان تابع

چنانکه پیشتر نیز مطرح شد مهمترین کاربرد مرجع، در ارسال پارامترهایی به تابع از طریق ارجاع می‌باشد. به‌جهت آنکه پیشتر مثال‌های متعددی در زمینهٔ فراخوانی توابع از طریق آدرس به وسیلهٔ اشاره‌گرها مطرح کردیم در این قسمت تنها به بیان تفاوت‌های به‌کارگیری مرجع به‌جای اشاره‌گرها در فراخوانی توابع از طریق آدرس می‌پردازیم.

مثال ۱۱-۱۳ : برنامه‌ای که چگونگی فراخوانی توابع را از طریق ارجاع نشان می‌دهد. در این برنامه یک مقدار اعشاری از کاربر دریافت شده و سپس در تابعی قسمت صحیح و اعشاری این عدد تجزیه شده و به نمایش در می‌آید.

```
1.  //This program demonstrates calling by reference
2.  #include <iostream>
3.  using namespace std;
4.  int main()
5.  {
6.     void intfrac(float, float&, float& ); //declaration
7.
8.     float number, intpart, fracpart;
9.
10.    do{
11.       cout<<"\nEnter a real number:";//a number from user
12.        cin>>number;
13.        //find int and frac********************
14.        intfrac(number, intpart, fracpart);
15. //**Notice:it doesn't need & operator in calling function
16.        cout<<"The integer part is "<<intpart //print them
17.           <<",the fraction part is "<<fracpart<<endl;
18.       }while( number != 0);
19.    return 0;
20. }
21. //*************************
22. void intfrac(float n,float& intp, float& fracp)
23. {
24.    long temp = static_cast<long>(n);
25.    intp = static_cast<float>(temp);    //find int part
26.    fracp = n - intp;                   //subtract int part
27. }
```

توضیح مثال: احتمالاً مهمترین نکته در این برنامه نحوهٔ فراخوانی تابع **intfrac** در خط ۱۴ کد است. چنانچه پیشتر نیز مطرح شد جهت نسبت دادن آدرس یک متغیر به یک مرجع نیازی به قرار دادن عملگر آدرس نیست. لذا در این خط بر خلاف آنچه که در هنگام فراخوانی توابع از طریق اشاره‌گرها دیده بودید، جهت ارسال پارامترهای دو و سوم هیچ عملگر آدرسی قرار ندادیم. نکتهٔ دیگر در خصوص این کد نحوهٔ جداسازی قسمت صحیح و اعشاری عدد موردنظر است. آیا می‌توانید در این خصوص توضیح دهید؟

مرجع به‌عنوان مقدار بازگشتی

پیشتر دیدید که توابع قادر هستند علاوه بر انواع استاندارد، اشاره‌گرها را نیز برگردانند. به‌عنوان مثال پیشتر با اعلان تابع **strcpy** آشنا شدید که اشاره‌گری از نوع ***char** را باز می‌گرداند. بر گرداندن یک آدرس از حافظه و یا به‌عبارت بهتر برگرداندن یک اشاره‌گر، در مباحث ساختمان داده‌های پیوندی بسیار پرکاربرد و با اهمیت جلوه می‌کند، که در جلد دوم به تفصیل در این خصوص بحث خواهیم کرد. اما بازگرداندن مرجع توسط یک تابع کاربرد کمتری دارد، و شاید از نظر برنامه‌نویسی جزء در مواردی خاص، چون بازگرداندن اشیاء که مربوط به برنامه‌نویسی شئ‌گرا می‌گردد، اهمیت ویژه‌ای در زمینهٔ برنامه‌نویسی ساخت‌یافته نداشته باشد. هنگامی که تابعی مرجعی را باز می‌گرداند تنها آدرس یک خانه از حافظه بازگردانده می‌شود، که قادر هستیم بدین‌وسیله مقادیر مورد نظر را به آن خانه از حافظه نسبت دهیم. این حالت یکی از نادرترین حالاتی است که تابع در سمت چپ دستور انتساب قرار می‌گیرد، چراکه در حقیقت با اجرای تابع، در نهایت یک مرجع در سمت چپ دستور انتساب قرار می‌گیرد که مقدار سمت راست دستور انتساب به آن نسبت داده می‌شود.

مثال ۱۱-۱۴: برنامه‌ای که کاربرد نوع برگشتی مرجع را در توابع نشان می‌دهد.

```
1.  //returning reference values
2.  #include <iostream>
3.  using namespace std;
4.  int X;                  //global variable
5.  int& setx();
6.  int main()
7.  {
8.    setx() = 110;
9.      cout<<"X = "<<X<<endl;
10.   return 0;
11. }
12. //**********************
13. int& setx()
14. {
15.   return X;//return the value to be modified.(with out &)
16. }
```

توضیح مثال: آنچه که در این برنامه رخ داده بازگشت آدرس متغیر سراسری **X** توسط تابع **setx** است. این تابع در خط هشتم برنامه فراخوانی شده و با بازگرداندن مرجعی به متغیر **X** موجب می‌شود که مقدار **110** در این متغیر قرار گیرد. در حقیقت خط هشتم این برنامه معادل با دستور زیر است:

```
X = 110;
```

خروجی این برنامه نیز به‌صورت زیر خواهد بود:

```
X = 110
```

کار در کلاس ۱۱-۲ :

برنامه ای بنویسید که با استفاده از مرجع به عنوان مقدار بازگشتی یک تابع، فضاهای خالی بین یک رشتهٔ دریافتی را با کاراکتر خط تیره پرکند و در نهایت رشتهٔ جدید را در تابع فراخوان چاپ نماید.

عملگر const و مرجع

پیشتر با تأثیرات عملگر const بر اشاره‌گرها آشنا شدید، و دیدید که چگونه این عملگر موجب قفل شدن آدرس و یا مقدار در رابطه با اشاره‌گرها می‌شود. اما از آنجا که آدرس داخل یک مرجع به‌طور خودکار قفل شده است، تنها کاربرد این عملگر بر روی مرجع می‌تواند با قفل شدن مقدار همراه باشد. به‌عنوان مثال فرض کنید تابعی را به‌صورت زیر اعلان کنید:

```
void aFunc(int& a, const int& b);
```

هرگونه تلاش در جهت تغییر مقداری که مرجع b به آن ارجاع می‌کند در داخل بدنهٔ این تابع می‌تواند موجب بروز خطای زمان کامپایل شود، این در حالی است که می‌توانید مقدار مرجع a را دستکاری کنید.

۱۱-۵ : تمرین

۱. آرایه‌ای از اشاره‌گرها به رشته‌ها را در نظر بگیرید. به‌عنوان مثال فرض کنید لیستی از اسامی یک کلاس در اختیار دارید. برنامه‌ای بنویسید که این لیست را برحسب حروف الفبای لاتین مرتب کند.

۲. نسخه‌هایی از توابع **strcmp** و **strcpy** را خودتان بازنویسی کنید.

۳. برنامه‌ای بنویسید که با دریافت یک کلمه، آن کلمه را در درون یک رشته جستجو کند. سپس تحقیق کنید چه الگوریتم‌ها و روش‌های کارآمدتری نسبت به الگوریتم شما وجود دارد. در برنامهٔ خود از اشاره‌گرها بهره بگیرید.

۴. آرایه‌ای چهار بعدی با ابعاد **m** و **n** و **k** و **l** به روش پویا پیاده‌سازی کنید. سپس در مورد نحوهٔ عملکرد پیاده‌سازی خود شرح دهید. آرایهٔ خود را در برنامه‌ای بکار گیرید و مطمئن شوید که آرایه را به‌طرز صحیحی تعریف کرده‌اید.

۵. تابعی بنویسید که به‌عنوان آرگومان، دو مقدار را به واسطهٔ ارجاع دریافت کند و آنگاه مقدار کوچکتر را برابر صفر قرار دهد.

۶. تابعی به نام **swap** بنویسید که به وسیلهٔ قالب‌های تابع و ارجاع هر دو مقدار ورودی را با یکدیگر تعویض کند. این دو مقدار ممکن است دو رشته یا دو عدد صحیح یا هر دو ورودی مشابه دیگر باشند.

۷. فرض کنید یک آرایهٔ ۲۴×۲۴ دارید، این آرایه را با دو علامت . و # پرکنید؛ به‌طوری که # به معنای دیوار یا حصار و نقطه به معنای جاده باشد. هدف طرح‌ریزی برنامه‌ای است که به کاربر اجازه دهد بر روی جاده‌ها از یک نقطهٔ آغازین حرکت کند تا از یک ورودی به خروجی این شبکهٔ پر پیچ و خم برسد. در ابتدا مسئله را برای یک مسیر ثابت حل کنید سپس برنامه را طوری گسترش دهید که این مسیر پر پیچ‌وخم به‌طور تصادفی ایجاد گردد و هر بار که دکمهٔ شروع (به‌عنوان مثال **F1**) فشرده می‌شود مسیر جدیدی ترسیم گردد. البته با این بازی در اسباب بازی ها که یک گوی در داخل یک میدان حرکت داده می‌شود تا از داخل مسیرها بیرون آید آشنا شده‌اید. البته ممکن است شما تمایل داشته باشید به جای یک مربع از یک دایره به‌عنوان میدان مسیرهای پر پیچ و خم استفاده کنید.

۸. مسئلهٔ قبل را به‌گونه‌ای دیگر تغییر می‌دهیم. این بار برنامه را طوری طرح‌ریزی کنید که کاربر مسیر موردنظر خود را با ورود دو علامت موردنظر در صفحه به‌صورت خانه به خانه ترسیم کند، و سپس کامپیوتر کلیهٔ مسیرهای ممکن را بیابد. برای حل این مسئله باید از تکنیک توابع بازگشتی استفاده کنید. البته برنامه باید به‌صورتی طرح‌ریزی شود که از ورود مسیرهای غلط جلوگیری کند، مثلاً قرار دادن دو ردیف حصار به‌صورت زیر در مسیر اشتباه است و حتماً باید بین هر دو حصاری یک جاده قرار گیرد:

```
############
############
```

حالت‌های دیگر خطا را خودتان بررسی کنید.

۹. خروجی برنامهٔ زیر چیست؟ آیا می‌توانید نکات جدید برنامه‌نویسی از آن استخراج کنید؟

```
#include <iostream>
using namespace std;
int main()
{
    int* ptr = new int(1);
    int& ref = *ptr;  // How to refer to heap space
    cout << ref << '\n';
    ptr = new int(2);
    cout << ref << '\n';
    delete &ref;
    delete ptr;
    return 0;
}
```

۱۰.خروجی برنامهٔ زیر چیست؟ آیا می‌توانید نکات جدید برنامه‌نویسی از آن استخراج کنید؟

```
#include <iostream>
using namespace std;
int main()
{
     int* ptr = new int(1);
     int*& ref = ptr;
     cout << *ref << '\n';
     delete ref;
     ptr = new int(2);
     cout << *ref << '\n';
     delete ref;
     //delete ptr;
     //make error because ptr is deleted in 11th line.
     return 0;
}
```

۱۱.خروجی برنامهٔ زیر چیست؟ آیا می‌توانید نکات جدید برنامه‌نویسی از آن استخراج کنید؟

```
#include <iostream>
using namespace std;
int& input(int& value)
{
    cout << "Enter an integer: ";
     cin >> value;
    return value;
}
int main()
{
    int* ptr = new int(); //initialed variable with zero
     cout << "Value = " << input(*ptr) << '\n';
      delete ptr;
    return 0;
}
```

۱۲.فرض کنید تعاریف زیر را برای x و y داریم:

```
int x = 0;
const int y = 0;
```

به نظر شما کدام یک از دستورات زیر با خطا همراه است و کدام یک خیر؟

```
1.  int& ref = y;
2.  int& ref = 0;
3.  int& ref = x;
4.  const int& ref = x;
5.  const int& ref = 0;
6.  const int& ref = y;
7.  int& const ref = y;
8.  int& const ref = 0;
9.  int& const ref = x;
```

۱۳.خروجی برنامهٔ زیر چیست؟

```
1.  #include <iostream>
2.  using namespace std;
3.  int& foo(int& number)
4.  {
5.     ++number;
6.     return number;
7.  }
8.  main()
9.  {
10.    int num = 0;
11.     cout<< "num = " << ++foo(num)-- << '\n';
12.    return 0;
13. }
```

۱۴. فرض کنید تعاریف زیر را داریم:

```
int dim = 10;
int** ptr;
typedef int (* ptrf)();
```

به نظر شما کدام یک از دستورات زیر صحیح است و کدام یک خیر؟ سعی کنید برای هر یک از موارد بالا که صحیح نیست دلیل بیاورید و صحیح آن را بنویسید. همچنین در مواردی که دستور را درست تشخیص می‌دهید، بگویید این اشاره‌گر به چه نوع متغیر یا تابعی قادر است اشاره کند.

```
1.  ptr = new (int*[dim]);
2.  ptr = new (int*)[dim];
3.  int (** ptr_func)() = new (int(*)());
4.  ptrf* ptr_func = new ptrf;
5.  int (** ptr_func)() = new (int(*[dim])());
6.  ptrf* ptr_func = new ptrf[dim];
```

۱۵.برنامه‌ای بنویسید که با دریافت ضرایب یک هر چندجمله‌ای نزولی دلخواه برحسب x به‌عنوان ضابطهٔ تابع f (x) =0 مقدار مساحت زیر منحنی f (x) را از a تا b محسابه کرده و بازگرداند.

۱۱-۶: موارد مطالعاتی

۱. (پروژهٔ برنامه‌نویسی) : احتمالاً دیگر باید بتوانید پروژه مربوط به پردازشگر متن را که پیشتر مطرح شده بود تکمیل کنید پس در ابتدا به این امر مبادرت ورزید.

۲. (پروژهٔ برنامه‌نویسی) : با مراجعه به ماشین‌حساب‌های مهندسی و بررسی چگونگی عملکرد آنها، برنامه‌ای بنویسید که عملکرد یک ماشین‌حساب مهندسی را شبیه سازی کند. سپس با استفاده از آنچه که در مورد نگاره سازی فراگرفتید قسمتی به برنامهٔ خود اضافه کنید که برخی از توابع سادهٔ ریاضی را دریافت و آنها را رسم کند.

۳. (پروژهٔ برنامه‌نویسی) : یک فرهنگ لغت ایجاد کنید که کاربر بتواند با جستجوی یک کلمه معانی مختلف مرتبط با آن را بیابد. برنامه باید دارای دو صفحهٔ مجزا باشد که با فشردن یک دکمه بین آنها سوئیچ کند. یک صفحه برای جستجو در فرهنگ لغت و یک صفحه برای افزودن کلمه به فرهنگ لغت. احتمالاً بهتر است از بردارها در این پروژه بهره بگیرید.

۴. (پروژهٔ برنامه‌نویسی) : برنامه‌ای بنویسید که با دریافت تاریخ میلادی، تقویمی برحسب هجری شمسی به‌صورت ماه به ماه به‌دست دهد. کاربر باید بتواند با زدن یک دکمه به تقویم مربوط به ماه قبل، و با زدن یک دکمه به تقویم ماه بعد مراجعه کند. احتمالاً کمی نیازمند تحقیق در خصوص تقویم میلادی هستید.

۵. (پروژهٔ برنامه‌نویسی) : برنامه‌ای بنویسید که عمل بر زدن و کشیدن ورق‌ها را بین چهار بازی‌کن در یک بازی حکم، شبیه‌سازی کند.

۶. (پروژهٔ برنامه‌نویسی) : با استفاده از آرایه‌های دندانه‌ای برنامه‌ای بنویسید که تعداد خطوط مشخصی از مثلث خیام-پاسکال را در حافظه ذخیره و چاپ کند. تعداد خطوط مورد نظر باید از کاربر پرسیده شود.

۷. در مورد کاربرد تابع **strdup** تحقیق کنید.

۸. به کمک استاد خود در مورد روش جستجوی **quick sort** تحقیق کنید و سپس برنامه‌ای برای پیاده‌سازی آن برروی یک آرایهٔ دو بعدی تلاش کنید.

۹. مراقب باشید هیچگاه یک متغیر محلی مستقر درپشتهٔ برنامه را از یک تابع به واسطهٔ مرجع یا اشاره‌گر برنگردانید. چرا که اگر چنین عملی را انجام دهید شما به متغیری اشاره یا ارجاع کرده‌اید که برای مدت زمان اندکی صاحب آن نبوده و نیستید. حال تحقیق کنید در C++ کدام متغیرها از فضای **Stack** و کدام متغیرها از فضای **heap** تخصیص داده می‌شوند.

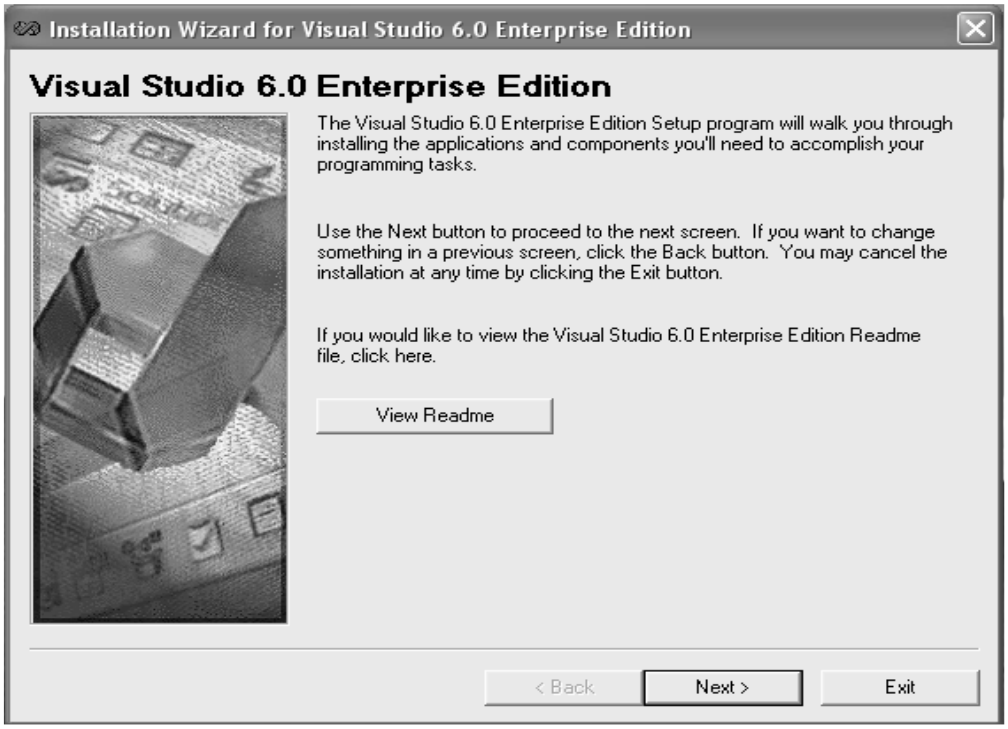
۱۰. تحقیق کنید چگونه می‌توانیم در حین اجرا از مقدار فضای باقی‌مانده در **heap** سیستم با خبر بشویم، و چگونه می‌توانیم به این وسیله برکمبود حافظه و احتمالاً بر حافظه‌ای که بر اساس تراوش و نشت از دست رفته مدیریت کنیم. منظور از تراوش آن است که برنامه‌نویس فراموش کند حافظه‌ای را که از سیستم گرفته به آن بازگرداند، در چنین حالتی احتمال آنکه با کمبود حافظه روبرو شویم کم نیست.

بخش

ضمیمه ها

پیوست ۱: نصب VS98

در صورتی که سی‌دی Visual Studio 98 را از قبل در اختیار داشته‌اید و یا آنکه از طریق وب سایت کتاب تهیه کرده اید، جهت نصب محیط Visual Studio لازم است سی‌دی مذکور را در داخل سی‌دی‌رام خود قرار دهید. پس از قرار دادن این سی‌دی داخل سی‌دی‌رام، برنامه نصب به‌طور auto run اجرا شده و پنجره‌ای مطابق شکل ۱۲-۱ ظاهر می‌شود.

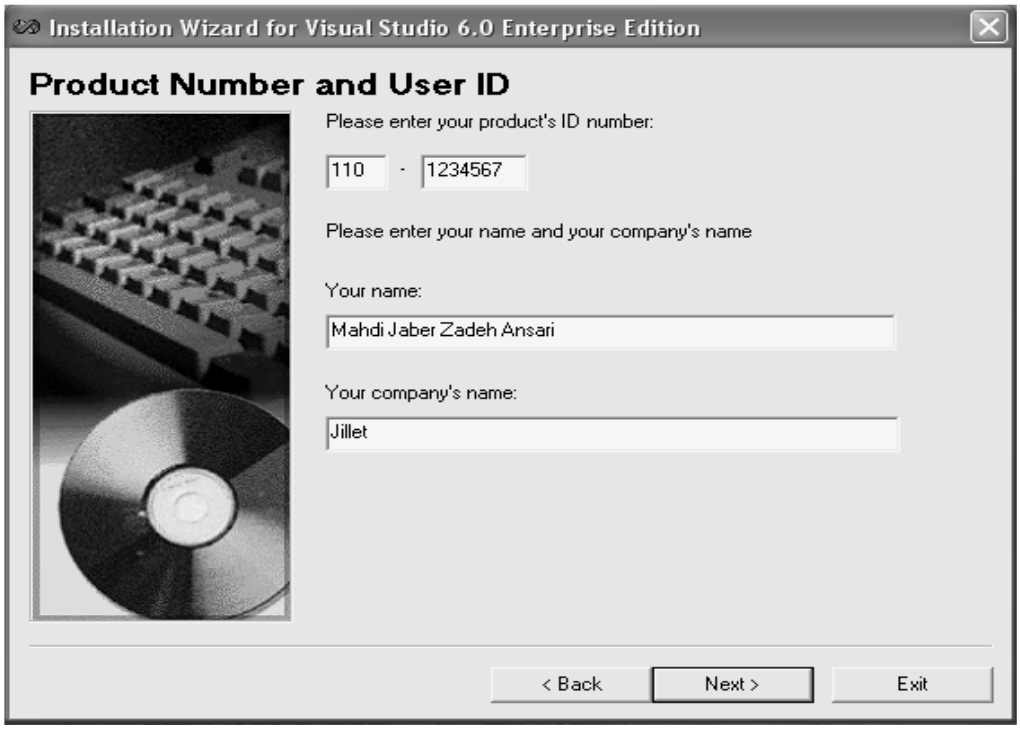


شکل : ۱۲- ۱

پس از آن مراحل نصب به‌صورت زیر خواهد بود:

۱. در این پنجره کلید Next را بفشارید تا پنجره‌ای با عنوان End User License Agreement باز شود در این پنجره پس از انتخاب گزینهٔ I accept دکمهٔ Next فعال می‌گردد، این کلید را بفشارید تا به مرحلهٔ بعد بروید.

۲. در این مرحله پنجره‌ای با عنوان Product Number and User ID ظاهر می‌شود. در قسمت ID Number مطابق شکل ۱۲-۲ اعداد 1234567 – 110 را وارد کنید و پس از آن نام خود و نام مؤسسه خود را وارد کرده و کلید Next را بفشارید تا به مرحلهٔ بعد بروید.



شکل : ۲۱-۲

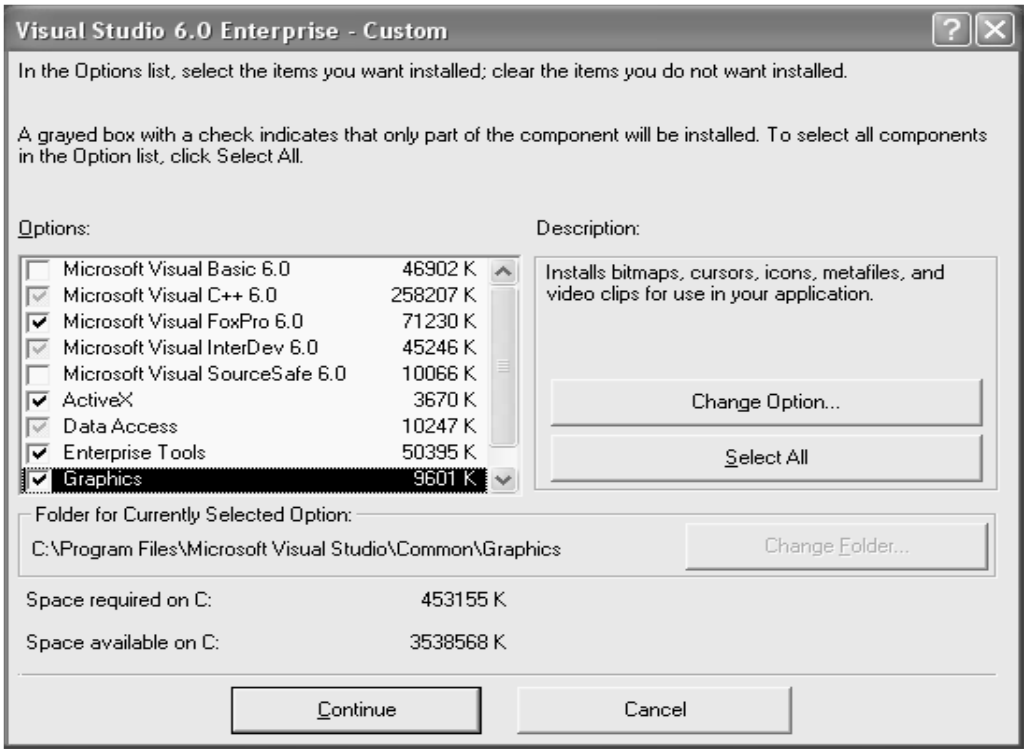
۳. پس از آن در صورتی که ماشین مجازی جاوا برروی سیستم شما نصب نباشد، از شما سؤال می‌شود که آیا این ماشین Update بشود یا نه؟ که باید در گزینهٔ مربوطه علامت تیک بزنید و دکمهٔ Next را بفشارید.

۴. پس از آنکه ماشین مجازی جاوا برروی سیستم شما نصب گردید، پیغامی ظاهر می‌شود که برنامهٔ نصب باید کامپیوتر شما را مجدداً راه‌اندازی کند. لذا با فشردن دکمهٔ ok اجازه دهید که کامپیوتر reboot گردد.

۵. پس از آنکه کامپیوتر مجدداً راه‌اندازی شد، پنجره‌ای با عنوان Visual Studio 6.0 Enterprise Edition ظاهر می‌گردد. که در آن باید نوع نصب را انتخاب کنید برای این منظور نوع Custom را انتخاب کنید و دکمهٔ Next را بفشارید.

۶. پس از آن در پنجرهٔ Choose Common Install Folder شاخهٔ موردنظر برای نصب را انتخاب نمایید. البته بهتر است مسیر پیش‌فرض را قبول کنید و کلید Next را بفشارید.

۷. پس از آن پنجره‌هایی باز می‌شود که مقادیر پیش‌فرض آنها را قبول کنید تا آنکه به پنجره‌ای مطابق با شکل ۱۲-۳ برخورد کنید. در این حالت گزینه‌های موجود را مطابق با این شکل تنظیم کنید و دکمهٔ continue را بفشارید.



شکل ۱۲-۳

۸. سپس پنجرهٔ دیگری باز می‌شود که با انتخاب گزینهٔ Register Environment Variable و فشردن دکمهٔ OK مراحل نصب آغاز می‌گردد.

۹. پس از اتمام مراحل نصب از شما خواسته می‌شود که کامپیوتر خود را یکبار دیگر راه‌اندازی مجدد کنید. در این حالت گزینه‌ای با نام Microsoft Visual Studio 6.0 در قسمت All Program منوی Start ظاهر می‌شود که می‌توانید از این قسمت مطابق روش مطرح شده در قسمت ۳-۴ پروژه‌ها و برنامه‌های موردنظر خود را ایجاد کنید.

۱۰. پس از راه‌اندازی مجدد کامپیوتر ویزارد جدیدی ظاهر می‌شود که از شما در مورد نصب محیط MSDN می‌پرسد. در صورتی که MSDN سی‌دی‌های مربوطه را تهیه کرده‌اید، حتماً MSDN را برروی کامپیوتر خود نصب کنید زیرا بسیاری از مشکلات شما را در مورد کد نویسی در VC6 پاسخگو خواهد بود.

پیوست ۲: جدول کدهای اسکی

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI
^P	16	10		DLE
^Q	17	11		DC1
^R	18	12		DC2
^S	19	13		DC3
^T	20	14		DC4
^U	21	15		NAK
^V	22	16		SYN
^W	23	17		ETB
^X	24	18		CAN
^Y	25	19		EM
^Z	26	1A		SUB
^[	27	1B		ESC
^\	28	1C		FS
^]	29	1D		GS
^^	30	1E	▲	RS
^-	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

Dec	Hex	Char
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

Dec	Hex	Char
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	░
177	B1	▒
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Char
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

Dec	Hex	Char
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	φ
238	EE	ε
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	n
253	FD	2
254	FE	■
255	FF	

تذکر : عدد 1– در مبنای ۱۶ برابر FF و عدد 128– برابر 80 در مبنای هگزا دسیمال (مبنای ۱۶) خواهند بود لذا در کامپیوترهایی که کاراکترها از ۱۲۸– تا ۱۲۷ تغییر می‌کنند جدول فوق با کمی دگرگونی روبرو خواهد شد. به‌طوری که خانهٔ 255 برابر 1– و خانهٔ 128 برابر 128– خواهد بود.

پیوست ۳ : سرفایل Limits.h

```
/***
*limits.h - implementation dependent values
*
*       Copyright (c) 1985-1997, Microsoft Corporation.  All
rights reserved.
*
*Purpose:
*       Contains defines for a number of implementation
dependent values
*       which are commonly used in C programs.
*       [ANSI]
*
*       [Public]
****/
#if     _MSC_VER > 1000
#pragma once
#endif

#ifndef _INC_LIMITS
#define _INC_LIMITS

#if     !defined(_WIN32) && !defined(_MAC)
#error ERROR: Only Mac or Win32 targets supported!
#endif

#define CHAR_BIT      8         /* number of bits in a char
*/
#define SCHAR_MIN   (-128)      /* minimum signed char value
*/
#define SCHAR_MAX     127       /* maximum signed char value
*/
#define UCHAR_MAX     0xff      /* maximum unsigned char
value */
#ifndef _CHAR_UNSIGNED
#define CHAR_MIN    SCHAR_MIN   /* mimimum char value */
#define CHAR_MAX    SCHAR_MAX   /* maximum char value */
#else
#define CHAR_MIN      0
#define CHAR_MAX    UCHAR_MAX
#endif  /* _CHAR_UNSIGNED */

#define MB_LEN_MAX    2             /* max. # bytes in
multibyte char */
#define SHRT_MIN    (-32768)        /* minimum (signed)
short value */
#define SHRT_MAX      32767         /* maximum (signed)
short value */
#define  USHRT_MAX        0xffff            /* maximum unsigned
short value */
```

```
#define  INT_MIN         (-2147483647  -  1)  /* minimum  (signed)
int value */
#define  INT_MAX          2147483647      /* maximum  (signed)  int
value */
#define  UINT_MAX         0xffffffff      /* maximum unsigned int
value */
#define LONG_MIN     (-2147483647L - 1) /* minimum (signed)
long value */
#define  LONG_MAX         2147483647L     /* maximum  (signed)long
value */
#define  ULONG_MAX       0xffffffffUL   /* maximum unsigned long
value */

#if      _INTEGRAL_MAX_BITS >= 8
#define  _I8_MIN       (-127i8  -  1)      /* minimum signed 8 bit
value */
#define  _I8_MAX         127i8            /* maximum signed 8 bit
value */
#define  _UI8_MAX      0xffui8        /* maximum unsigned 8
bit value */
#endif

#if      _INTEGRAL_MAX_BITS >= 16
#define  _I16_MIN      (-32767i16  -  1) /* minimum signed 16 bit
value */
#define  _I16_MAX        32767i16        /* maximum signed 16 bit
value */
#define  _UI16_MAX     0xffffui16     /* maximum unsigned 16
bit value */
#endif

#if      _INTEGRAL_MAX_BITS >= 32
#define  _I32_MIN    (-2147483647i32 - 1) /* minimum signed
32 bit value */
#define  _I32_MAX        2147483647i32 /* maximum signed 32 bit
value */
#define  _UI32_MAX     0xffffffffui32 /* maximum unsigned 32
bit value */
#endif

#if      _INTEGRAL_MAX_BITS >= 64
/* minimum signed 64 bit value */
#define  _I64_MIN    (-9223372036854775807i64 - 1)
/* maximum signed 64 bit value */
#define  _I64_MAX      9223372036854775807i64
/* maximum unsigned 64 bit value */



#define  _UI64_MAX     0xffffffffffffffffui64
#endif
```

```
#if     _INTEGRAL_MAX_BITS >= 128
/* minimum signed 128 bit value */
#define _I128_MIN      (-
170141183460469231731687303715884105727i128 - 1)
/* maximum signed 128 bit value */
#define _I128_MAX
170141183460469231731687303715884105727i128
/* maximum unsigned 128 bit value */
#define _UI128_MAX
0xffffffffffffffffffffffffffffffffui128
#endif

#ifdef  _POSIX_

#define _POSIX_ARG_MAX      4096
#define _POSIX_CHILD_MAX    6
#define _POSIX_LINK_MAX     8
#define _POSIX_MAX_CANON    255
#define _POSIX_MAX_INPUT    255
#define _POSIX_NAME_MAX     14
#define _POSIX_NGROUPS_MAX  0
#define _POSIX_OPEN_MAX     16
#define _POSIX_PATH_MAX     255
#define _POSIX_PIPE_BUF     512
#define _POSIX_SSIZE_MAX    32767
#define _POSIX_STREAM_MAX   8
#define _POSIX_TZNAME_MAX   3

#define ARG_MAX                14500        /* 16k heap, minus
overhead */
#define LINK_MAX            1024
#define MAX_CANON           _POSIX_MAX_CANON
#define MAX_INPUT           _POSIX_MAX_INPUT
#define NAME_MAX            255
#define NGROUPS_MAX         16
#define OPEN_MAX            32
#define PATH_MAX            512
#define PIPE_BUF            _POSIX_PIPE_BUF
#define SSIZE_MAX           _POSIX_SSIZE_MAX
#define STREAM_MAX          20
#define TZNAME_MAX          10

#endif  /* POSIX */

#endif  /* _INC_LIMITS */
```

پیوست ۴: سرفایل Math.h

```
/***
*math.h - definitions and declarations for math library
*
*          Copyright (c) 1985-1997, Microsoft Corporation. All
rights reserved.
*
*Purpose:
*         This file contains constant definitions and external
subroutine
*       declarations for the math subroutine library.
*       [ANSI/System V]
*
*       [Public]
*
****/

#if     _MSC_VER > 1000
#pragma once
#endif

#ifndef _INC_MATH
#define _INC_MATH

#if     !defined(_WIN32) && !defined(_MAC)
#error ERROR: Only Mac or Win32 targets supported!
#endif


#ifdef  _MSC_VER
/*
 * Currently, all MS C compilers for Win32 platforms default
to 8 byte
 * alignment.
 */
#pragma pack(push,8)
#endif  /* _MSC_VER */

#ifdef __cplusplus
extern "C" {
#endif

#ifndef __assembler /* Protect from assembler */

/* Define _CRTIMP */
#ifndef _CRTIMP


#ifdef  _DLL
#define _CRTIMP __declspec(dllimport)
#else   /* ndef _DLL */
```

```
#define _CRTIMP
#endif  /* _DLL */
#endif  /* _CRTIMP */


/* Define __cdecl for non-Microsoft compilers */

#if     ( !defined(_MSC_VER) && !defined(__cdecl) )
#define __cdecl
#endif

/* Define _CRTAPI1 (for compatibility with the NT SDK) */

#ifndef _CRTAPI1
#if   _MSC_VER >= 800 && _M_IX86 >= 300
#define _CRTAPI1 __cdecl
#else
#define _CRTAPI1
#endif
#endif


/* Definition of _exception struct - this struct is passed
to the matherr
 * routine when a floating point exception is detected
 */

#ifndef _EXCEPTION_DEFINED
struct _exception {
        int type;       /* exception type - see below */
        char *name;     /* name of function where error
occured */
        double arg1;    /* first argument to function */
        double arg2;    /* second argument (if any) to
 function */
        double retval;  /* value to be returned by function
*/
        } ;

#define _EXCEPTION_DEFINED
#endif


/* Definition of a _complex struct to be used by those who
use cabs and


 * want type checking on their argument
 */

#ifndef _COMPLEX_DEFINED
```

```
struct _complex {
        double x,y; /* real and imaginary parts */
        } ;

#if     !__STDC__ && !defined (__cplusplus)
/* Non-ANSI name for compatibility */
#define complex _complex
#endif

#define _COMPLEX_DEFINED
#endif
#endif  /* __assembler */


/* Constant definitions for the exception type passed in the
_exception struct
 */

#define _DOMAIN     1   /* argument domain error */
#define _SING       2   /* argument singularity */
#define _OVERFLOW   3   /* overflow range error */
#define _UNDERFLOW  4   /* underflow range error */
#define _TLOSS      5   /* total loss of precision */
#define _PLOSS      6   /* partial loss of precision */

#define EDOM        33
#define ERANGE      34


/* Definitions of _HUGE and HUGE_VAL - respectively the
XENIX and ANSI names
 * for a value returned in case of error by a number of the
floating point
 * math routines
 */
#ifndef __assembler /* Protect from assembler */
_CRTIMP extern double _HUGE;
#endif  /* __assembler */

#define HUGE_VAL _HUGE


/* Function prototypes */




#if     !defined(__assembler)   /* Protect from assembler */
#if     _M_MRX000
_CRTIMP int     __cdecl abs(int);
_CRTIMP double  __cdecl acos(double);
```

```
_CRTIMP double  __cdecl asin(double);
_CRTIMP double  __cdecl atan(double);
_CRTIMP double  __cdecl atan2(double, double);
_CRTIMP double  __cdecl cos(double);
_CRTIMP double  __cdecl cosh(double);
_CRTIMP double  __cdecl exp(double);
_CRTIMP double  __cdecl fabs(double);
_CRTIMP double  __cdecl fmod(double, double);
_CRTIMP long    __cdecl labs(long);
_CRTIMP double  __cdecl log(double);
_CRTIMP double  __cdecl log10(double);
_CRTIMP double  __cdecl pow(double, double);
_CRTIMP double  __cdecl sin(double);
_CRTIMP double  __cdecl sinh(double);
_CRTIMP double  __cdecl tan(double);
_CRTIMP double  __cdecl tanh(double);
_CRTIMP double  __cdecl sqrt(double);
#else
        int     __cdecl abs(int);
        double  __cdecl acos(double);
        double  __cdecl asin(double);
        double  __cdecl atan(double);
        double  __cdecl atan2(double, double);
        double  __cdecl cos(double);
        double  __cdecl cosh(double);
        double  __cdecl exp(double);
        double  __cdecl fabs(double);
        double  __cdecl fmod(double, double);
        long    __cdecl labs(long);
        double  __cdecl log(double);
        double  __cdecl log10(double);
        double  __cdecl pow(double, double);
        double  __cdecl sin(double);
        double  __cdecl sinh(double);
        double  __cdecl tan(double);
        double  __cdecl tanh(double);
        double  __cdecl sqrt(double);
#endif
_CRTIMP double  __cdecl atof(const char *);
_CRTIMP double  __cdecl _cabs(struct _complex);
#if     defined(_M_ALPHA)
        double  __cdecl ceil(double);
        double  __cdecl floor(double);

#else


_CRTIMP double  __cdecl ceil(double);
_CRTIMP double  __cdecl floor(double);
#endif
_CRTIMP double  __cdecl frexp(double, int *);
```

```
_CRTIMP double  __cdecl _hypot(double, double);
_CRTIMP double  __cdecl _j0(double);
_CRTIMP double  __cdecl _j1(double);
_CRTIMP double  __cdecl _jn(int, double);
_CRTIMP double  __cdecl ldexp(double, int);
        int     __cdecl _matherr(struct _exception *);
_CRTIMP double  __cdecl modf(double, double *);

_CRTIMP double  __cdecl _y0(double);
_CRTIMP double  __cdecl _y1(double);
_CRTIMP double  __cdecl _yn(int, double);


#if     defined(_M_MRX000)

/* MIPS fast prototypes for float */
/* ANSI C, 4.5 Mathematics        */

/* 4.5.2 Trigonometric functions */

_CRTIMP float  __cdecl acosf( float );
_CRTIMP float  __cdecl asinf( float );
_CRTIMP float  __cdecl atanf( float );
_CRTIMP float  __cdecl atan2f( float , float );
_CRTIMP float  __cdecl cosf( float );
_CRTIMP float  __cdecl sinf( float );
_CRTIMP float  __cdecl tanf( float );

/* 4.5.3 Hyperbolic functions */
_CRTIMP float  __cdecl coshf( float );
_CRTIMP float  __cdecl sinhf( float );
_CRTIMP float  __cdecl tanhf( float );

/* 4.5.4 Exponential and logarithmic functions */
_CRTIMP float  __cdecl expf( float );
_CRTIMP float  __cdecl logf( float );
_CRTIMP float  __cdecl log10f( float );
_CRTIMP float  __cdecl modff( float , float* );

/* 4.5.5 Power functions */
_CRTIMP float  __cdecl powf( float , float );
        float  __cdecl sqrtf( float );




/*  4.5.6  Nearest  integer,  absolute  value,  and  remainder
functions */
        float  __cdecl ceilf( float );
        float  __cdecl fabsf( float );
```

```
        float  __cdecl floorf( float );
_CRTIMP float  __cdecl fmodf( float , float );

_CRTIMP float  __cdecl hypotf(float, float);

#endif  /* _M_MRX000 */

#if     defined(_M_ALPHA)

/* ALPHA fast prototypes for float */
/* ANSI C, 4.5 Mathematics        */

/* 4.5.2 Trigonometric functions */

        float  __cdecl acosf( float );
        float  __cdecl asinf( float );
        float  __cdecl atanf( float );
        float  __cdecl atan2f( float , float );
        float  __cdecl cosf( float );
        float  __cdecl sinf( float );
        float  __cdecl tanf( float );

/* 4.5.3 Hyperbolic functions */
        float  __cdecl coshf( float );
        float  __cdecl sinhf( float );
        float  __cdecl tanhf( float );

/* 4.5.4 Exponential and logarithmic functions */
        float  __cdecl expf( float );
        float  __cdecl logf( float );
        float  __cdecl log10f( float );
_CRTIMP float  __cdecl modff( float , float* );

/* 4.5.5 Power functions */
        float  __cdecl powf( float , float );
        float  __cdecl sqrtf( float );

/*  4.5.6  Nearest  integer,  absolute  value,  and  remainder
functions */
        float  __cdecl ceilf( float );
        float  __cdecl fabsf( float );
        float  __cdecl floorf( float );
        float  __cdecl fmodf( float , float );




_CRTIMP float  __cdecl _hypotf(float, float);

#endif  /* _M_ALPHA */
```

```
#if     !defined(_M_M68K)
/* Macros defining long double functions to be their double
counterparts
 *   (long   double   is   synonymous   with   double   in   this
implementation).
 */


#ifndef __cplusplus
#define acosl(x)     ((long double)acos((double)(x)))
#define asinl(x)     ((long double)asin((double)(x)))
#define atanl(x)     ((long double)atan((double)(x)))
#define     atan2l(x,y)     ((long     double)atan2((double)(x),
(double)(y)))
#define _cabsl       _cabs
#define ceill(x)     ((long double)ceil((double)(x)))
#define cosl(x)      ((long double)cos((double)(x)))
#define coshl(x)     ((long double)cosh((double)(x)))
#define expl(x)      ((long double)exp((double)(x)))
#define fabsl(x)     ((long double)fabs((double)(x)))
#define floorl(x)    ((long double)floor((double)(x)))
#define     fmodl(x,y)         ((long     double)fmod((double)(x),
(double)(y)))
#define frexpl(x,y)  ((long double)frexp((double)(x), (y)))
#define  _hypotl(x,y)         ((long  double)_hypot((double)(x),
(double)(y)))
#define ldexpl(x,y)  ((long double)ldexp((double)(x), (y)))
#define logl(x)      ((long double)log((double)(x)))
#define log10l(x)    ((long double)log10((double)(x)))
#define _matherrl    _matherr
#define  modfl(x,y)     ((long  double)modf((double)(x),  (double
*)(y)))
#define    powl(x,y)            ((long    double)pow((double)(x),
(double)(y)))
#define sinl(x)      ((long double)sin((double)(x)))
#define sinhl(x)     ((long double)sinh((double)(x)))
#define sqrtl(x)     ((long double)sqrt((double)(x)))
#define tanl(x)      ((long double)tan((double)(x)))
#define tanhl(x)     ((long double)tanh((double)(x)))
#else   /* __cplusplus */
inline long double acosl(long double _X)
        {return (acos((double)_X)); }
inline long double asinl(long double _X)
        {return (asin((double)_X)); }



inline long double atanl(long double _X)
        {return (atan((double)_X)); }
inline long double atan2l(long double _X, long double _Y)
```

```
        {return (atan2((double)_X, (double)_Y)); }
inline long double ceill(long double _X)
        {return (ceil((double)_X)); }
inline long double cosl(long double _X)
        {return (cos((double)_X)); }
inline long double coshl(long double _X)
        {return (cosh((double)_X)); }
inline long double expl(long double _X)
        {return (exp((double)_X)); }
inline long double fabsl(long double _X)
        {return (fabs((double)_X)); }
inline long double floorl(long double _X)
        {return (floor((double)_X)); }
inline long double fmodl(long double _X, long double _Y)
        {return (fmod((double)_X, (double)_Y)); }
inline long double frexpl(long double _X, int *_Y)
        {return (frexp((double)_X, _Y)); }
inline long double ldexpl(long double _X, int _Y)
        {return (ldexp((double)_X, _Y)); }
inline long double logl(long double _X)
        {return (log((double)_X)); }
inline long double log10l(long double _X)
        {return (log10((double)_X)); }
inline long double modfl(long double _X, long double *_Y)
        {double _Di, _Df = modf((double)_X, &_Di);
        *_Y = (long double)_Di;
        return (_Df); }
inline long double powl(long double _X, long double _Y)
        {return (pow((double)_X, (double)_Y)); }
inline long double sinl(long double _X)
        {return (sin((double)_X)); }
inline long double sinhl(long double _X)
        {return (sinh((double)_X)); }
inline long double sqrtl(long double _X)
        {return (sqrt((double)_X)); }
inline long double tanl(long double _X)
        {return (tan((double)_X)); }
inline long double tanhl(long double _X)
        {return (tanh((double)_X)); }

inline float frexpf(float _X, int *_Y)
        {return ((float)frexp((double)_X, _Y)); }
inline float ldexpf(float _X, int _Y)
        {return ((float)ldexp((double)_X, _Y)); }
#if     !defined(_M_MRX000) && !defined(_M_ALPHA)



inline float acosf(float _X)
        {return ((float)acos((double)_X)); }
inline float asinf(float _X)
```

```
        {return ((float)asin((double)_X)); }
inline float atanf(float _X)
        {return ((float)atan((double)_X)); }
inline float atan2f(float _X, float _Y)
        {return ((float)atan2((double)_X, (double)_Y)); }
inline float ceilf(float _X)
        {return ((float)ceil((double)_X)); }
inline float cosf(float _X)
        {return ((float)cos((double)_X)); }
inline float coshf(float _X)
        {return ((float)cosh((double)_X)); }
inline float expf(float _X)
        {return ((float)exp((double)_X)); }
inline float fabsf(float _X)
        {return ((float)fabs((double)_X)); }
inline float floorf(float _X)
        {return ((float)floor((double)_X)); }
inline float fmodf(float _X, float _Y)
        {return ((float)fmod((double)_X, (double)_Y)); }
inline float logf(float _X)
        {return ((float)log((double)_X)); }
inline float log10f(float _X)
        {return ((float)log10((double)_X)); }
inline float modff(float _X, float *_Y)
        { double _Di, _Df = modf((double)_X, &_Di);
        *_Y = (float)_Di;
        return ((float)_Df); }
inline float powf(float _X, float _Y)
        {return ((float)pow((double)_X, (double)_Y)); }
inline float sinf(float _X)
        {return ((float)sin((double)_X)); }
inline float sinhf(float _X)
        {return ((float)sinh((double)_X)); }
inline float sqrtf(float _X)
        {return ((float)sqrt((double)_X)); }
inline float tanf(float _X)
        {return ((float)tan((double)_X)); }
inline float tanhf(float _X)
        {return ((float)tanh((double)_X)); }
#endif  /* !defined(_M_MRX000) && !defined(_M_ALPHA) */
#endif  /* __cplusplus */
#endif  /* _M_M68K */
#endif  /* __assembler */

#if     !__STDC__


/* Non-ANSI names for compatibility */

#define DOMAIN      _DOMAIN
```

```
#define SING            _SING
#define OVERFLOW        _OVERFLOW
#define UNDERFLOW       _UNDERFLOW
#define TLOSS           _TLOSS
#define PLOSS           _PLOSS

#ifndef _MAC
#define matherr         _matherr
#endif  /* ndef _MAC */

#ifndef __assembler /* Protect from assembler */

_CRTIMP extern double HUGE;

_CRTIMP double  __cdecl cabs(struct _complex);
_CRTIMP double  __cdecl hypot(double, double);
_CRTIMP double  __cdecl j0(double);
_CRTIMP double  __cdecl j1(double);
_CRTIMP double  __cdecl jn(int, double);
        int     __cdecl matherr(struct _exception *);
_CRTIMP double  __cdecl y0(double);
_CRTIMP double  __cdecl y1(double);
_CRTIMP double  __cdecl yn(int, double);

#endif  /* __assembler */

#endif  /* __STDC__ */

#ifdef  _M_M68K
/* definition of _exceptionl struct - this struct is passed
to the _matherrl
 * routine when a floating point exception is detected in a
long double routine
 */

#ifndef _LD_EXCEPTION_DEFINED

struct _exceptionl {
        int type;           /* exception type - see below */
        char *name;            /* name of function where error
occured */
        long double arg1;   /* first argument to function */
        long double arg2;     /* second argument (if any) to
function */




        long double retval; /* value to be returned by
function */
} ;
```

```
#define _LD_EXCEPTION_DEFINED
#endif


/* definition of a _complexl struct to be used by those who
use _cabsl and
 * want type checking on their argument
 */

#ifndef _LD_COMPLEX_DEFINED
struct _complexl {
        long double x,y;    /* real and imaginary parts */
} ;
#define _LD_COMPLEX_DEFINED
#endif


long double  __cdecl acosl(long double);
long double  __cdecl asinl(long double);
long double  __cdecl atanl(long double);
long double  __cdecl atan2l(long double, long double);
long double  __cdecl _atold(const char  *);
long double  __cdecl _cabsl(struct _complexl);
long double  __cdecl ceill(long double);
long double  __cdecl cosl(long double);
long double  __cdecl coshl(long double);
long double  __cdecl expl(long double);
long double  __cdecl fabsl(long double);
long double  __cdecl floorl(long double);
long double  __cdecl fmodl(long double, long double);
long double  __cdecl frexpl(long double, int  *);
long double  __cdecl _hypotl(long double, long double);
long double  __cdecl _j0l(long double);
long double  __cdecl _j1l(long double);
long double  __cdecl _jnl(int, long double);
long double  __cdecl ldexpl(long double, int);
long double  __cdecl logl(long double);
long double  __cdecl log10l(long double);
int          __cdecl _matherrl(struct _exceptionl  *);
long double  __cdecl modfl(long double, long double  *);
long double  __cdecl powl(long double, long double);
long double  __cdecl sinl(long double);
long double  __cdecl sinhl(long double);
long double  __cdecl sqrtl(long double);
long double  __cdecl tanl(long double);



long double  __cdecl tanhl(long double);
long double  __cdecl _y0l(long double);
long double  __cdecl _y1l(long double);
```

```
long double  __cdecl _ynl(int, long double);

#endif  /* _M_M68K */


#ifdef  __cplusplus
}

#if     !defined(_M_M68K)


template<class _Ty> inline
        _Ty _Pow_int(_Ty _X, int _Y)
        {unsigned int _N;
        if (_Y >= 0)
                _N = _Y;
        else
                _N = -_Y;
        for (_Ty _Z = _Ty(1); ; _X *= _X)
                {if ((_N & 1) != 0)
                        _Z *= _X;
                if ((_N >>= 1) == 0)
                        return (_Y < 0 ? _Ty(1) / _Z : _Z); }}

#ifndef _MSC_EXTENSIONS

inline long __cdecl abs(long _X)
        {return (labs(_X)); }
inline double __cdecl abs(double _X)
        {return (fabs(_X)); }
inline double __cdecl pow(double _X, int _Y)
        {return (_Pow_int(_X, _Y)); }
inline double __cdecl pow(int _X, int _Y)
        {return (_Pow_int(_X, _Y)); }
inline float __cdecl abs(float _X)
        {return (fabsf(_X)); }
inline float __cdecl acos(float _X)
        {return (acosf(_X)); }
inline float __cdecl asin(float _X)
        {return (asinf(_X)); }
inline float __cdecl atan(float _X)
        {return (atanf(_X)); }
inline float __cdecl atan2(float _Y, float _X)
        {return (atan2f(_Y, _X)); }



inline float __cdecl ceil(float _X)
        {return (ceilf(_X)); }
inline float __cdecl cos(float _X)
```

```
        {return (cosf(_X)); }
inline float __cdecl cosh(float _X)
        {return (coshf(_X)); }
inline float __cdecl exp(float _X)
        {return (expf(_X)); }
inline float __cdecl fabs(float _X)
        {return (fabsf(_X)); }
inline float __cdecl floor(float _X)
        {return (floorf(_X)); }
inline float __cdecl fmod(float _X, float _Y)
        {return (fmodf(_X, _Y)); }
inline float __cdecl frexp(float _X, int * _Y)
        {return (frexpf(_X, _Y)); }
inline float __cdecl ldexp(float _X, int _Y)
        {return (ldexpf(_X, _Y)); }
inline float __cdecl log(float _X)
        {return (logf(_X)); }
inline float __cdecl log10(float _X)
        {return (log10f(_X)); }
inline float __cdecl modf(float _X, float * _Y)
        {return (modff(_X, _Y)); }
inline float __cdecl pow(float _X, float _Y)
        {return (powf(_X, _Y)); }
inline float __cdecl pow(float _X, int _Y)
        {return (_Pow_int(_X, _Y)); }
inline float __cdecl sin(float _X)
        {return (sinf(_X)); }
inline float __cdecl sinh(float _X)
        {return (sinhf(_X)); }
inline float __cdecl sqrt(float _X)
        {return (sqrtf(_X)); }
inline float __cdecl tan(float _X)
        {return (tanf(_X)); }
inline float __cdecl tanh(float _X)
        {return (tanhf(_X)); }
inline long double __cdecl abs(long double _X)
        {return (fabsl(_X)); }
inline long double __cdecl acos(long double _X)
        {return (acosl(_X)); }
inline long double __cdecl asin(long double _X)
        {return (asinl(_X)); }
inline long double __cdecl atan(long double _X)
        {return (atanl(_X)); }
inline long double __cdecl atan2(long double _Y, long double
_X)


        {return (atan2l(_Y, _X)); }
inline long double __cdecl ceil(long double _X)
        {return (ceill(_X)); }
```

```
inline long double __cdecl cos(long double _X)
        {return (cosl(_X)); }
inline long double __cdecl cosh(long double _X)
        {return (coshl(_X)); }
inline long double __cdecl exp(long double _X)
        {return (expl(_X)); }
inline long double __cdecl fabs(long double _X)
        {return (fabsl(_X)); }
inline long double __cdecl floor(long double _X)
        {return (floorl(_X)); }
inline long double __cdecl fmod(long double _X, long double
 _Y)
        {return (fmodl(_X, _Y)); }
inline long double __cdecl frexp(long double _X, int * _Y)
        {return (frexpl(_X, _Y)); }
inline long double __cdecl ldexp(long double _X, int _Y)
        {return (ldexpl(_X, _Y)); }
inline long double __cdecl log(long double _X)
        {return (logl(_X)); }
inline long double __cdecl log10(long double _X)
        {return (log10l(_X)); }
inline long double __cdecl modf(long double _X, long double
* _Y)
        {return (modfl(_X, _Y)); }
inline long double __cdecl pow(long double _X, long double _Y)
        {return (powl(_X, _Y)); }
inline long double __cdecl pow(long double _X, int _Y)
        {return (_Pow_int(_X, _Y)); }
inline long double __cdecl sin(long double _X)
        {return (sinl(_X)); }
inline long double __cdecl sinh(long double _X)
        {return (sinhl(_X)); }
inline long double __cdecl sqrt(long double _X)
        {return (sqrtl(_X)); }
inline long double __cdecl tan(long double _X)
        {return (tanl(_X)); }
inline long double __cdecl tanh(long double _X)
        {return (tanhl(_X)); }

#endif  /* _MSC_EXTENSIONS */
#endif  /* _M_M68K */
#endif  /* __cplusplus */

#ifdef  _MSC_VER



#pragma pack(pop)
#endif  /* _MSC_VER */

#endif  /* _INC_MATH */
```

پیوست ۵ : جدول کلیدهای ترکیبی

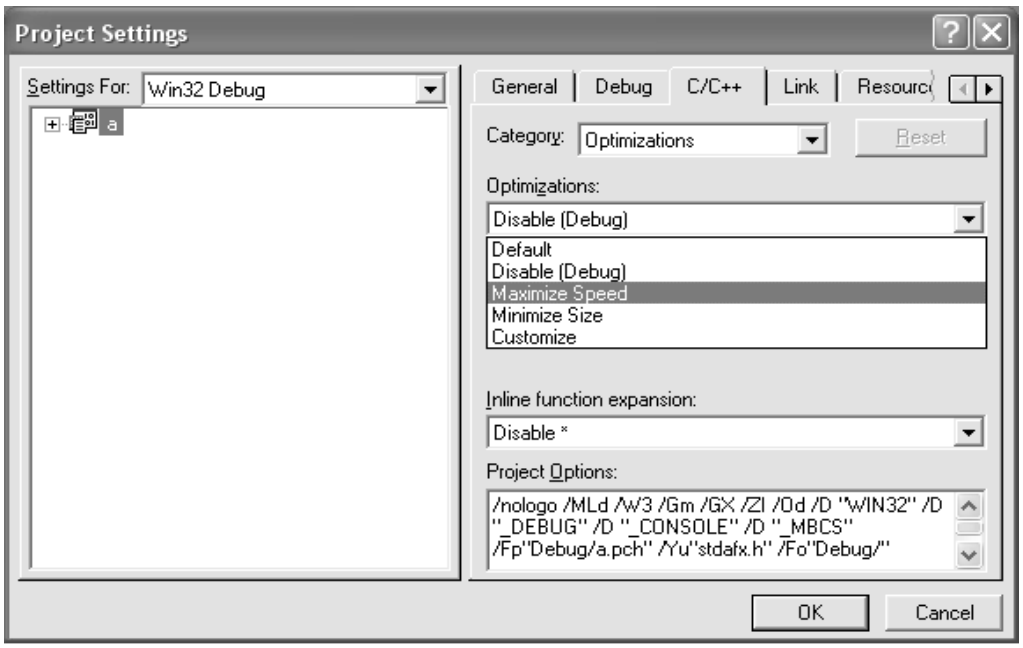
Key	Scan Code		ASCII or Extended†			ASCII or Extended† with SHIFT			ASCII or Extended† with CTRL			ASCII or Extended† with ALT		
	Dec	Hex	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
ESC	1	01	27	1B	ESC	27	1B	ESC	27	1B	ESC	1	01	NUL§
1!	2	02	49	31	1	33	21	!				120	78	NUL
2@	3	03	50	32	2	64	40	@	3	03	NUL	121	79	NUL
3#	4	04	51	33	3	35	23	#				122	7A	NUL
4$	5	05	52	34	4	36	24	$				123	7B	NUL
5%	6	06	53	35	5	37	25	%				124	7C	NUL
6^	7	07	54	36	6	94	5E	^	30	1E	RS	125	7D	NUL
7&	8	08	55	37	7	38	26	&				126	7E	NUL
8*	9	09	56	38	8	42	2A	*				127	7F	NUL
9(	10	0A	57	39	9	40	28	(				128	80	NUL
0)	11	0B	48	30	0	41	29	)				129	81	NUL
-_	12	0C	45	2D	-	95	5F	_	31	1F	US	130	82	NUL
=+	13	0D	61	3D	=	43	2B	+				131	83	NUL
BKSP	14	0E	8	08		8	08		127	7F		14	0E	NUL§
TAB	15	0F	9	09		15	0F	NUL	148	94	NUL§	15	A5	NUL§
Q	16	10	113	71	q	81	51	Q	17	11	DC1	16	10	NUL
W	17	11	119	77	w	87	57	W	23	17	ETB	17	11	NUL
E	18	12	101	65	e	69	45	E	5	05	ENQ	18	12	NUL
R	19	13	114	72	r	82	52	R	18	12	DC2	19	13	NUL
T	20	14	116	74	t	84	54	T	20	14	SC	20	14	NUL
Y	21	15	121	79	y	89	59	Y	25	19	EM	21	15	NUL
U	22	16	117	75	u	85	55	U	21	15	NAK	22	16	NUL
I	23	17	105	69	i	73	49	I	9	09	TAB	23	17	NUL
O	24	18	111	6F	o	79	4F	O	15	0F	SI	24	18	NUL
P	25	19	112	70	p	80	50	P	16	10	DLE	25	19	NUL
[{	26	1A	91	5B	[	123	7B	{	27	1B	ESC	26	1A	NUL§
]}	27	1B	93	5D	]	125	7D	}	29	1D	GS	27	1B	NUL§
ENTER	28	1C	13	0D	CR	13	0D	CR	10	0A	LF	28	1C	NUL§
ENTER♪	28	1C	13	0D	CR	13	0D	CR	10	0A	LF	166	A6	NUL§
LCTRL	29	1D												
RCTRL♪	29	1D												
A	30	1E	97	61	a	65	41	A	1	01	SOH	30	1E	NUL
S	31	1F	115	73	s	83	53	S	19	13	DC3	31	1F	NUL
D	32	20	100	64	d	68	44	D	4	04	EOT	32	20	NUL
F	33	21	102	66	f	70	46	F	6	06	ACK	33	21	NUL
G	34	22	103	67	g	71	47	G	7	07	BEL	34	22	NUL
H	35	23	104	68	h	72	48	H	8	08	BS	35	23	NUL
J	36	24	106	6A	j	74	4A	J	10	0A	LF	36	24	NUL
K	37	25	107	6B	k	75	4B	K	11	0B	VT	37	25	NUL
L	38	26	108	6C	l	76	4C	L	12	0C	FF	38	26	NUL
;:	39	27	59	3B	;	58	3A	:				39	27	NUL§
'"	40	28	39	27	'	34	22	"				40	28	NUL§
`~	41	29	96	60	`	126	7E	~				41	29	NUL§
LSHIFT	42	2A												
|	43	2B	92	5C	\	124	7C	|	28	1C	FS			
Z	44	2C	122	7A	z	90	5A	Z	26	1A	SUB	44	2C	NUL
X	45	2D	120	78	x	88	58	X	24	18	CAN	45	2D	NUL
C	46	2E	99	63	c	67	43	C	3	03	EIX	46	2E	NUL
V	47	2F	118	76	v	86	56	V	22	16	SYN	47	2F	NUL
B	48	30	98	62	b	66	42	B	2	02	STX	48	30	NUL
N	49	31	110	6E	n	78	4E	N	14	0E	SC	49	31	NUL
M	50	32	109	6D	m	77	4D	M	13	0D	CR	50	32	NUL
,<	51	33	44	2C	,	60	3C	<				51	33	NUL§
.>	52	34	46	2E	.	62	3E	>				52	34	NUL§

Key	Scan Code		ASCII or Extended†			ASCII or Extended† with SHIFT			ASCII or Extended† with CTRL			ASCII or Extended† with ALT		
	Dec	Hex	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
/?	53	35	47	2F	/	63	3F	?				53	34	NUL§
GRAY /¢	53	35	47	2F	/	63	3F	?	149	95	NUL	164	A5	NUL
R SHIFT	54	36												
*PRTSC	55	37	42	2A	*	PRTSC		↑↑	16	10				
L ALT	56	38												
R ALT¢	56	38												
SPACE	57	39	32	20	SPC	32	20	SPC	32	20	SPC	32	20	SPC
CAPS	58	3A												
F1	59	3B	59	3B	NUL	84	54	NUL	94	5E	NUL	104	68	NUL
F2	60	3C	60	3C	NUL	85	55	NUL	95	5F	NUL	105	69	NUL
F3	61	3D	61	3D	NUL	86	56	NUL	96	60	NUL	106	6A	NUL
F4	62	3E	62	3E	NUL	87	57	NUL	97	61	NUL	107	6B	NUL
F5	63	3F	63	3F	NUL	88	58	NUL	98	62	NUL	108	6C	NUL
F6	64	40	64	40	NUL	89	59	NUL	99	63	NUL	109	6D	NUL
F7	65	41	65	41	NUL	90	5A	NUL	100	64	NUL	110	6E	NUL
F8	66	42	66	42	NUL	91	5B	NUL	101	65	NUL	111	6F	NUL
F9	67	43	67	43	NUL	92	5C	NUL	102	66	NUL	112	70	NUL
F10	68	44	68	44	NUL	93	5D	NUL	103	67	NUL	113	71	NUL
F11¢	87	57	133	85	E0	135	87	E0	137	89	E0	139	8B	E0
F12¢	88	58	134	86	E0	136	88	E0	138	8A	E0	140	8C	E0
NUM	69	45												
SCROLL	70	46												
HOME	71	47	71	47	NUL	55	37	7	119	77	NUL			
HOME¢	71	47	71	47	E0	71	47	E0	119	77	E0	151	97	NUL
UP	72	48	72	48	NUL	56	38	8	141	8D	NUL§			
UP¢	72	48	72	48	E0	72	48	E0	141	8D	E0	152	98	NUL
PGUP	73	49	73	49	NUL	57	39	9	132	84	NUL			
PGUP¢	73	49	73	49	E0	73	49	E0	132	84	E0	153	99	NUL
GRAY-	74	4A				45	2D	–						
LEFT	75	4B	75	4B	NUL	52	34	4	115	73	NUL			
LEFT¢	75	4B	75	4B	E0	75	4B	E0	115	73	E0	155	9B	NUL
CENTER	76	4C				53	35	5						
RIGHT	77	4D	77	4D	NUL	54	36	6	116	74	NUL			
RIGHT¢	77	4D	77	4D	E0	77	4D	E0	116	74	E0	157	9D	NUL
GRAY+	78	4E				43	2B	+						
END	79	4F	79	4F	NUL	49	31	1	117	75	NUL			
END¢	79	4F	79	4F	E0	79	4F	E0	117	75	E0	159	9F	NUL
DOWN	80	50	80	50	NUL	50	32	2	145	91	NUL§			
DOWN¢	80	50	80	50	E0	80	50	E0	145	91	E0	160	A0	NUL
PGDN	81	51	81	51	NUL	51	33	3	118	76	NUL			
PGDN¢	81	51	81	51	E0	81	51	E0	118	76	E0	161	A1	NUL
INS	82	52	82	52	NUL	48	30	0	146	92	NUL§			
INS¢	82	52	82	52	E0	82	52	E0	146	92	E0	162	A2	NUL
DEL	83	53	83	53	NUL	46	2E	.	147	93	NUL§			
DEL¢	83	53	83	53	E0	83	53	E0	147	93	E0	163	A3	NUL

جدول فوق از مستندات **MSDN 98** استخراج شده است.

پیوست ۶ : بهینه‌سازی در کامپایلرها

در کامپایلرهای مختلف، عملیات متفاوتی برروی کد منبع برنامه صورت می‌گیرد تا برنامه اجرایی حاصل، از نظر پارامترهایی همچون سرعت کامپایل، سرعت اجرا و حافظهٔ مورد استعمال و یا ترکیبی از آنها بهینه باشد. لذا برای یک برنامه‌نویس سودمند خواهد بود که بداند یک کامپایلر نوعی چه بهینه سازی هایی را می تواند انجام دهد و کدام ها را نمی‌تواند. لذا در این قسمت به معرفی برخی از انواع بهینه‌سازی‌هایی که یک کامپایلر می تواند انجام دهد می‌پردازیم. اما قبل از آن باید این نکته را خاطر نشان سازیم که برای آنکه کامپایلر قادر به انجام این بهینه‌سازی‌ها باشد باید تنظیماتی را برروی آن انجام دهید، به‌عنوان مثال در کامپایلر VC6 می‌بایستی از طریق گزینهٔ Setting در منوی Project عمل کرد، به طوری که با انتخاب این گزینه پنجره‌ای مطابق شکل زیر باز می‌شود که می‌بایست در برگهٔ C/C++ گزینهٔ Optimizations را از قسمت Category انتخاب کنید و تنظیمات موردنظر خود را انجام دهید:



حال در ادامه به معرفی انواع بهینه‌سازی‌های ممکن در کامپایلرها می‌پردازیم، اما قبل از آن باید این نکته را متذکر سازیم که برخی از این بهینه‌سازی‌ها در راستای افزایش سرعت و برخی دیگر در راستای کاهش حافظه است.

۱. Function In Lining

کامپایلر می‌تواند فراخوانی یک تابع را با بدنه آن جایگزین نماید. به مثال زیر دقت نمایید:

```
1.  float Square(float a)
2.  {
3.     return a*a;
4.  }
5.  //********************
6.  float parabola(float x)
7.  {
8.     return Square(x) + 1.0f;
9.  }
```

کامپایلر می‌تواند دستور فراخوانی تابع را با بدنه آن جایگزین نماید به‌صورت زیر :

```
1.  float parabola(float x)
2.  {
3.     return x*x + 1.0f;
4.  }
```

مزایای این بهینه‌سازی عبارتند از:

۱. بالاسری ناشی از فراخوانی و بازگشت و انتقال پارامتر به تابع حذف می‌شود.

۲. درصد محلی بودن ارجاعات در کد بیشتر شده و در نتیجه ضریب برخورد در حافظهٔ کش (Cache) بیشتر می‌شود. به‌طوری که اگر زمان دسترسی به Cache برابر T_1 و زمان دسترسی به Ram برابر T_2 و درصد یافتن داده‌ها در Cache برابر H=90% باشد آنگاه زمان کل دسترسی به داده‌ها و کد برنامه به‌طور قابل ملاحظه‌ای مطابق رابطهٔ زیر کاهش می‌یابد:

$$T_s = H.T_1 + (1-H).T_2$$

۳. اگر فقط یک فراخوانی روی تابع inline شده وجود داشته باشد کد کوچکتر می‌شود.

۴. inline نمودن یک تابع فرصتی را برای اعمال سایر بهینه‌سازی‌ها روی کد به‌وجود می‌آورد که در ادامه توضیح داده خواهد شد.

۵. اما عیب inline نمودن یک تابع این است که اگر در صد فراخوانی تابع در کد زیاد باشد و به خصوص اگر که تابع بزرگ باشد باعث افزایش طول کد برنامه می‌شود. به هر حال اگر تابع کوچک باشد و اگر تعداد فراخوانی‌های آن کم باشد کامپایلر می‌تواند بدون ذکر برنامه‌نویس آن تابع را inline کند.

۲. Constant Folding and Constant Propagation

در بهینه سازی Constant Folding یک عبارت یا زیر عبارت که فقط شامل ثوابت است با نتیجه عبارت جایگزین می‌شود. به مثال زیر دقت کنید:

```
1.  double a , b;
2.  a = b + 2.0/3.0;
```

کامپایلر این کد را با کد زیر جایگزین می کند:

```
1.  double a , b;
2.  a = b + 0.666666666666667;
```

به این نکته دقت نمایید که در عبارتی مثل (b*2.0 / 3.0) این نوع بهینه‌سازی اعمال نمی‌شود، البته دلیل آن نیز روشن است، چرا که اولویت‌ها موجب می‌شوند که ابتدا عمل b*2.0 صورت گیرد. پس بهتر است این عبارت توسط خود برنامه‌نویس به‌صورت b*(2.0 / 3.0) نوشته شود.

در بهینه‌سازی Constant Propagation یک ثابت در صورت امکان در بین یک سری از عبارات منتشر می‌شود. به مثال زیر توجه نمایید:

```
1.  float parabola(float x)
2.  {
3.     return x*x + 1.0f;
4.  }
5.  float a, b;
6.    a = parabola(2.0f); //it is equal: a=2*2+1;
7.     b = a + 1.0f;
```

کامپایلر این کد را با کد زیر جایگزین می‌نماید:

```
6.  a = 5.0f;
7.    b = 6.0f;
```

البته اگر تابع مذکور عبارتی باشد که نتواند inline شود یا نتواند در زمان کامپایل محاسبه گردد نمی‌توان از این نوع بهینه‌سازی استفاده کرد. به‌عنوان مثال اگر در عبارتی از تابع کتابخانه‌ای sin استفاده گردد که در یک کتابخانه جدا تعریف شده نمی‌توان از کامپایلر انتظار داشت که فراخوانی این تابع را حذف کند چرا که همچنان که در انتهای فصل سوم مطرح شد زمان ترجمهٔ این توابع تا مرحلهٔ پیوند به طول می‌انجامد، زیرا در این مرحله است که توابع کتابخانه‌ای به کد ما پیوند می‌خورند.

۳. Pointer Elimination

چنانکه پیشتر مطرح شد برای دستیابی به مقدار یک متغیر توسط یک اشاره‌گر به دو مرحله decode کردن توسط cpu نیاز است این در حالی است که برای دستیابی به مقدار یک متغیر توسط نام آن تنها به یک مرحله decode کردن نیاز است، لذا اگر در زمان کامپایل معلوم باشد که یک اشاره‌گر به چه خانه‌ای از حافظه اشاره می‌کند می‌توان آن اشاره‌گر را حذف نمود. به مثال زیر دقت کنید:

```
1.  void plus(int* p)
2.  {
3.    *p = *p + 2;
4.  }
5.  int a;
6.   plus(&a);
```

کامپایلر خط ۶ این کد را با کد زیر جایگزین می‌کند:

```
6.  a += 2;
```

به نظر شما آیا این نوع از بهینه‌سازی در آرایه‌های پویا نیز قابل اعمال است!؟

۴. Common Sub Expression Elimination

اگر در یک عبارت ریاضی چندین بار یک زیر عبارت تکرار شده باشد آنگاه کامپایلر فقط یکبار آن زیر عبارت را محاسبه می‌نماید. به مثال زیر دقت نمایید:

```
1.  int a, b, c;
2.  b = (a+1) * (a+1);
3.  c = (a+1) / 4;
```

کامپایلر این کد را با کد زیر جایگزین می‌کند:

```
1.  int a, b, c, temp;
2.  temp = a+1;
3.  b = temp * temp;
4.  c = temp / 4;
```

۵. Register Variables

در این نوع از بهینه‌سازی کامپایلر سعی می‌کند که متغیرهایی را که بیشتر از همه مورد استفاده قرار می‌گیرند را در ثبات‌های cpu ذخیره کند. از جمله این متغیرها می‌توان به شمارنده‌های حلقه‌ها، اشاره‌گرها و پارامترهای تابع و... اشاره کرد.

به خاطر دارید که اگر در کد یک برنامه با آدرس یک متغیر کار می‌کنیم آن متغیر نمی‌تواند در یک ثبات قرار گیرد چرا که ثبات آدرس ندارد.

۶. Join Identical Branches

در این نوع از بهینه‌سازی قسمت‌های مشترک کد مربوط به دو پرش if-then-else به منظور صرفه‌جویی در کد یکی می‌شوند. به مثال زیر دقت نمایید:

```
1.  double x, y, z;
2.  bool b;
3.  if (b)
4.  {
5.    y = sin(x);
6.    z = y + 1.0;
7.  }
8.  else
9.  {
10.   y = cos(x);
11.   z = y + 1.0;
12. }
```

کامپایلر این کد را با کد زیر جایگزین می‌کند:

```
1.  double x, y, z;
2.  bool b;
3.  if (b)
4.  {
5.     y = sin(x);
6.  }
7.  else
8.  {
9.     y = cos(x);
10. }
11.  z = y + 1.0;
```

۷. Eliminate Jump

با کپی کردن کدی که به آن پرش می‌شود می‌توان از انجام پرش جلوگیری کرد. به مثال زیر دقت کنید:

```
1.  int SomeFunction(int a, bool b)
2.  {
3.    if(b)
4.    {
5.      a = a*2;
6.    }
7.   else
8.    {
9.      a = a * 3;
10.   }
11.  return a + 1;
12. }
```

کامپایلر این کد را با کد زیر جایگزین می‌کند:

```
1.  int SomeFunction(int a, bool b)
2.  {
3.    if(b)
4.    {
5.      a = a*2;
6.      return a + 1;
7.    }
8.   else
9.    {
10.     a = a * 3;
11.     return a + 1;
12.   }
13. }
```

همچنین اگر درست یا غلط بودن نتیجه یک عبارت شرطی مربوط به یک دستور شرطی در زمان کامپایل مشخص باشد می‌توان پرش را حذف نمود به مثال زیر دقت نمایید:

```
1.  if(true)
2.  {
3.     a=b;
4.  }
5.  else
6.  {
7.     a=c;
8.  }
```

کامپایلر این کد را با کد زیر جایگزین می‌کند:

```
1.  a=b;
```

همچنین اگر دو دستور شرطی با عبارات شرطی یکسان به‌طور متوالی آمده باشند و برای کامپایلر واضح باشد که نتیجه این دو عبارت شرطی یکسان است در این صورت دستور شرطی دوم می‌تواند حذف شود و دستورات موجود در بدنه آن به بدنه دستور شرطی اول اضافه شود. به مثال زیر توجه فرمایید:

```
1.  int SomeFunction(int a, bool b)
2.  {
3.    if(b)
4.     {
5.        a = a*2;
6.     }
7.   else
8.     {
9.        a = a * 3;
10.    }
11. //***********************
12.   if(b)
13.    {
14.      return a + 1;
15.    }
16.  else
17.   {
18.     return a - 1;
19.   }
20. }
```

کامپایلر این کد را با کد زیر جایگزین می‌کند:

```
1.  int SomeFunction(int a, bool b)
2.  {
3.    if(b)
4.     {
5.        a = a*2;
6.        return a + 1;
7.     }
8.    else
9.     {
10.      a = a * 3;
11.      return a - 1;
12.    }
13. }
```

۸. Loop Unrolling

بعضی از کامپایلرها حلقه را باز می‌کنند، معمولاً این کار زمانی انجام می‌شود که بدنه حلقه خیلی کوچک باشد و باز کردن آن راه را برای انجام بهینه‌سازی‌های بیشتر باز کند. از طرف دیگر حلقه‌هایی نیز که تعداد تکرار آنها خیلی کوچک است به منظور حذف نمودن بالاسری حلقه‌سازی باز می‌شوند. به مثالی که در ادامه آمده است توجه کنید:

```
1.  int i , a[2];
2.   for(i=0;i<2;i++)
3.       a[i]=i+1;
```

کامپایلر این کد را با کد زیر جایگزین می‌کند:

```
1.  int a[2];
2.   a[0]=1;
3.    a[1]=2;
```

۹. Loop Invariant Code Motion

در این بهینه‌سازی، کامپایلر عبارت یا دستوری را که بود و نبود آن در حلقه، هیچ فرقی نداشته باشد از حلقه خارج می‌کند. به مثال زیر دقت کنید:

```
1.  int i, a[100], b;
2.  for(i=0;i<100;i++)
3.  {
4.     a[i] = b*b + 1;
5.  }
```

کامپایلر این کد را با کد زیر جایگزین می‌کند:

```
1.  int i, a[100], b, temp;
2.  temp = b*b + 1;
3.  for(i=0;i<100;i++)
4.  {
5.     a[i] = temp;
6.  }
```

۱۰. Algebraic Reduction

در بسیاری از کامپایلرها عبارات ساده جبری را می‌توان با استفاده از قوانین پایه‌ای جبر ساده‌تر نمود. به‌عنوان مثال یک کامپایلر می‌تواند عبارت (a–)– را به عبارت a ساده نماید. البته معمولاً برنامه‌نویسان چنین عبارتی را به‌کار نمی‌برد ولی چنین عبارتی ممکن است حاصل سایر بهینه‌سازی‌ها مثل Function InLining باشد. اما معمولاً برنامه‌نویسان مبتدی از عبارات جبری در برنامه‌های خود استفاده می‌کنند که ممکن است بتوان ساده‌تر هم نوشته شوند. به‌عنوان مثال برنامه‌نویس ممکن است از دستور if(!a && !b) به جای دستور if(!(a || b)) استفاده کرده باشد، این در حالی است که دستور دوم یک عملگر کمتر دارد. خوشبختانه کامپایلر می‌تواند این دسته از عبارات را بهینه کند.

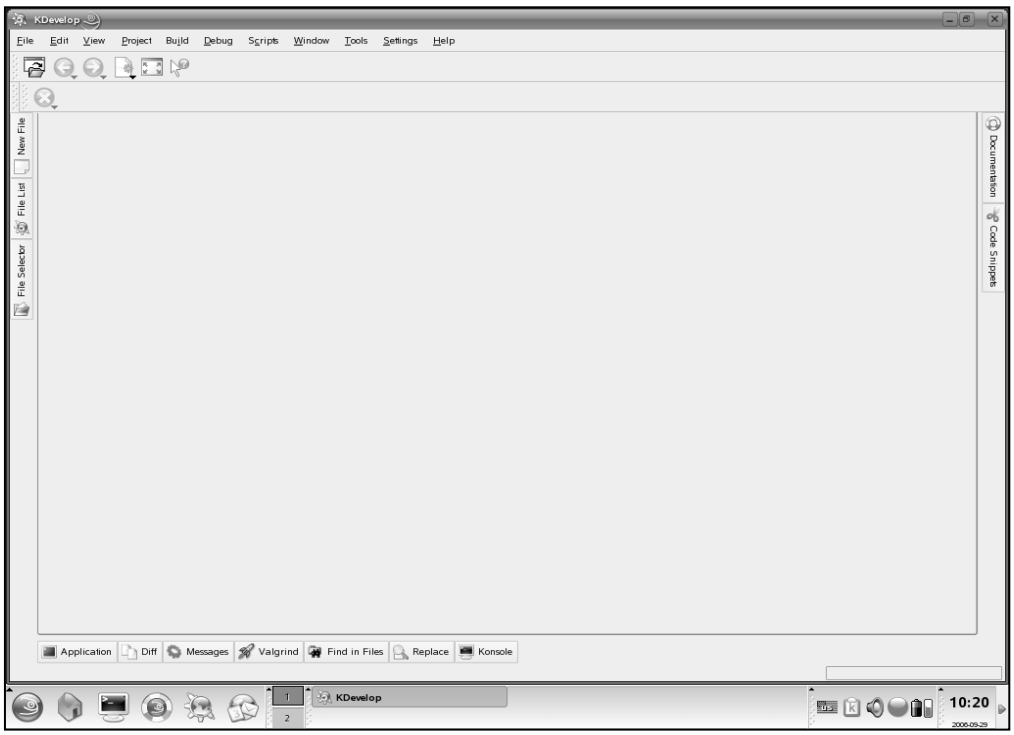
البته از کامپایلر انتظار نداشته باشید که عبارات پیچیده جبری را ساده نماید. به‌عنوان مثال به عبارت جبری a*b*c+c*a*b دقت کنید. این عبارت در فرم ساده شده به‌صورت 2*a*b*c می‌باشد. ولی کامپایلر ممکن است نتواند این بهینه‌سازی را انجام دهد. به هر حال کامپایلری وجود ندارد که کلیه قوانین پایه‌ای جبر را در بهینه‌سازی عبارات جبری به‌کار ببرد. چرا که ممکن است برای انجام چنین بهینه‌سازی که شاید تأثیر چشمگیری هم برروی سرعت اجرای برنامه نداشته باشد مجبور به صرف ساعت‌ها زمان در مرحلهٔ کامپایل برنامه باشیم.

پیوست ۷ : کامپایل برنامه در لینوکس

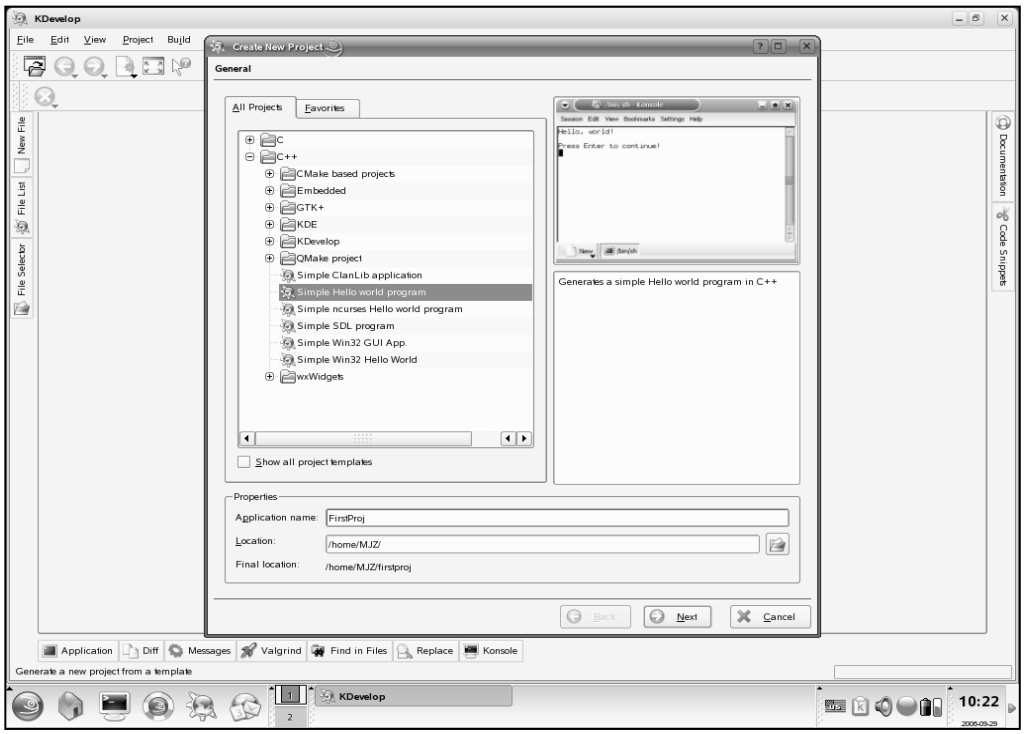
در سیستم‌عامل لینوکس جهت ایجاد و کامپایل برنامه‌های C++ از برنامهٔ KDevelop استفاده می‌شود. اگر کامپایلر C++ و همچنین محیط KDevelop را برروی سیستم‌عامل لینوکس خود نصب کرده باشید قادر به اجرای این برنامه خواهید بود. برای این منظور ما مراحل ایجاد و کامپایل یک برنامهٔ سادهٔ C++ را تحت سیستم‌عامل Suse 10.0 بیان می‌کنیم. این مراحل در دیگر سیستم‌عامل‌های لینوکس نیز به همین صورت است.

مراحل کامپایل و اجرای یک برنامهٔ C++ در محیط KDevelop

۱. جهت اجرا کردن برنامهٔ KDevelop ابتدا به قسمت Kmenu رفته و در منوی Development برنامهٔ IDE for C/C++ (KDevelop) : C/C++ را از داخل منوی Integrated Environment اجرا کنید. همچنین می‌توانید به جای این عمل کلمهٔ KDevelop را در منوی Run تایپ کرده و کلید Enter را بفشارید. در هر صورت باید شکلی مطابق شکل زیر برروی صفحه نمایش نمایان گردد:



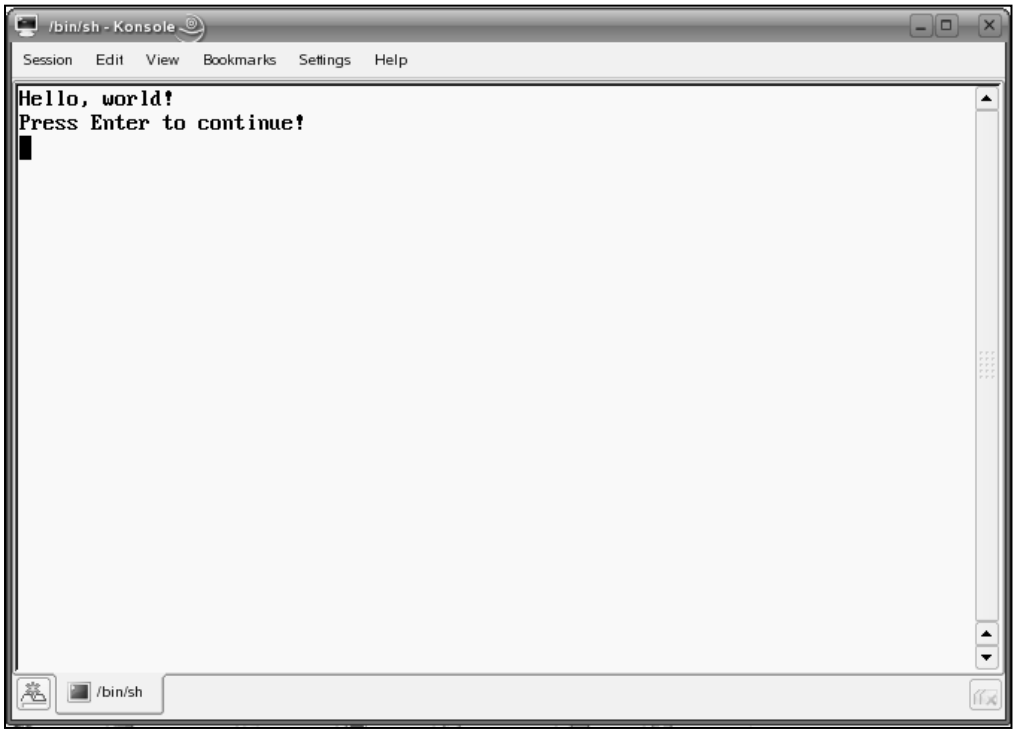
۲. سپس با انتخاب گزینهٔ NewProject از داخل منوی Project صفحه‌ای مطابق شکل زیر در مقابل شما باز خواهد شد:



۳. در این مرحله ضمن انتخاب گزینهٔ Simple Hello word program مطابق شکل فوق، نام پروژه و محل ذخیرهٔ آن را مشخص کنید و کلید Next را بفشارید. به‌عنوان مثال ما نام firstproj را برای این منظور انتخاب کرده‌ایم. پس از آن با زدن مکرر کلید Next پیش‌فرض‌ها را بپذیرید تا در نهایت با ظاهر شدن دکمهٔ Finish مراحل ایجاد پروژه به پایان برسد. در این مرحله قطه کد بسیار کوچکی از طرف کامپایلر به‌عنوان نمونه در فایل اصلی برنامه نوشته شده که می‌توان با کامپایل و اجرای آن، نحوهٔ ایجاد یک پروژه را در لینوکس تجربه کرد.

۴. جهت کامپایل این برنامه می‌توان گزینهٔ Build Project را از داخل منوی Build انتخاب کرد و یا آن دکمهٔ F8 را فشرد. در این صورت پس از یکی دو دقیقه باید پیغام ***Success*** را در پنجرهٔ خروجی دریافت کنید. البته اگر خطای نحوی در کد شما وجود داشته باشد اخطار ناشی از آن را به جای این پیغام خواهید دید.

۵. پس از کامپایل موفقیت‌آمیز برنامه باید آن را اجرا کنید تا خروجی آن را ببینید. برای این منظور می‌توانید دکمه‌های Shift + F9 را بفشارید و یا آنکه گزینهٔ Execute Program را از داخل منوی Build انتخاب کنید. در هر صورت باید خروجی زیر را مشاهده کنید.



۶. کامپایل فایل‌های C++ از طریق خط فرمان نیز امکانپذیر است. برای این منظور می‌توانید از گزینهٔ g++ استفاده کنید. البته جزئیات آن را به‌عنوان تحقیق بر عهدهٔ خود شما قرار می‌دهیم.

۷. در سیستم‌عامل لینوکس تقریباً کد نویسی زبان‌های C و C++ مشابه سیستم‌عامل ویندوز است، و اکثر مطالبی را که تا اینجا فرا گرفتید، می‌توانید در محیط KDevelop استفاده کنید. اما برخی از سرفایل‌ها که مختص محیط MS-Dos و Windows است را نمی‌توانید استفاده کنید، مثل سرفایل conio.h که شامل توابعی است که با وقفه‌های Dos پیاده‌سازی شده است.

واژه‌نامه

آرایه	array
آزمون	test
اتوماتیک	auto
اجتماع	union
ادغام	merge
ارزیابی	evaluate
از بالا به پایین	top-down
اشاره گر	pointer
اشتراک	intersection
اشتراک زمانی	time sharing
اصلی	main
اطلاعات	information
اعلان	declaration
الحاق	concatenation
الگو	pattern
الگوریتم	algorithm
انباره	accumulator
انتساب	assignment
انشعاب	branch
انشعاب چند راهه	multiple branching
بارکردن	load
بازگشت، بازگرداندن	return
بازیابی	retrieve
بدون علامت	unsigned
برد	range
بردار	vector
برنامه‌سازی	programming
برنامه نویسی	coding
بلند	long
به هنگام در آوردن	update
بیت	bit
پایانه	terminal
پاک‌کردن	delete
پردازش	process
پرونده	file
پویش	scan
پیشانه	console
پیمانه‌ای	modular
پیوند	link
تابع	function
تجزیه	decomposition
ترجمه	compile
تصادفی	random
تطابق	matching
تغییر مکان	shift
تکانه	pulse
تو در تو	nested
توزیع‌پذیری	distributive
ثابت	constant , static
ثبات	register
عمل -	operation register

address register	– نشانی
index register	– نمایه
permutation	جایگشت
table	جدول
table look-up	جدول خوانی
decision box	جعبه تصمیم
printer	چاپگر
cycle	چرخه
memory, storage	حافظه
catch	حافظه داخل پردازنده
real	حقیقی
statement	حکم
loop	حلقه
looping	حلقه زنی
field	حوزه
output	خروجی
data	داده
entry	درایه
interpolation	درونیابی
inline	درون خطی
instruction	دستورالعمل
disk	دسته
precision	دقت
sequence	دنباله
disk	دیسک
store	ذخیره کردن
record	رکورد
code	رمز
operation code	– عمل
encode coding	– گذاری
decode	– گشایی
flowchart	روند نما
chain	زنجیره
sub algorithm	زیر الگوریتم
subprogram	زیر برنامه
subroutine	زیر روال
sub expression	زیر عبارت
subtask	زیر وظیفه
structure	ساخت
structured	ساختیافته
overflow	سر ریز
simulator	شبیه‌ساز
simulation	شبیه‌سازی
break	شکستن
counter	شمارنده
instruction counter	شمارنده دستورالعمل
explicit	صریح
formal	صوری
integer	عدد صحیح
prefect number	عدد کامل
signal	علامت
signed	علامت دار
operation	عمل
operator	عملگر

عملوند	operand
فراخوانی	call
فرمان	command
فضای نام	namespace
فورترن	Fortran
قاب	frame
قالب	format , template
قطع کردن	truncate
قلمرو	domain
کاربر	user
کارت	card
کف	floor
کلمه	word
کوبول	Cobol
کوتاه	short
گام به گام	stepwise
گرد کردن	rounding
لفظ	literal
لیست	list
لیست پیوندی	linked list
ماتریس	matrix
ماتریس مربع	square matrix
مترجم	compiler
متغیر	variable
متغیر راهگزین (وضعیت)	switch variable
محفظه ورودی	input hopper
محیط	environment
مرتب کردن	sorting
مرتب کردن حبابی	bubble sorting
مرتب کردن سطلی	bucket sorting
مفسر	interpreter
مقدارخوان	reader
ممیز شناور	floating point
منطقی	Boolean
میانگیر	buffer
میانه	median
نسخه‌برداری	copy
نشانوند	argument
نشانی	address
نگاره‌سازی	graphing
نگاشت	map
نگهبان	sentinel
نمایش	display
اندیس، زیرنویس	index
اندیس‌گذاری	indexing
نویسه، کاراکتر، حرف، نشانه	character
نویسه خالی	blank
واحد	unit
ورودی	input
وظیفه	task
ویرایش	edit
هم عامل	cofactor
همنهشت	congruent

فهرست منابع و مآخذ

1. **C++ How To Program. 2en ed. 1998 . Deitel & Deitel, Harvey M.**
2. **C++ The Complete Reference, 4rd ed, 2002, Herbert schildt.**
3. **MSDN , April 2005.**
4. **MSDN, for Visual studio 6.0 .**
5. **Learning C++ A Hands-On Approach. 2en ed. Eric Nagler.**
6. **Computer Science A First Course. 2nd ed. 1975. John Wiley & Sons. A, I, Forsythe. T, A, Keenan. E, I, Organic. W, Stenberg.**
7. **The Waite Group's Object-Oriented Programming in C++ . 3rd ed. 1995 . Lafore , Robert W.**
8. **Introduction To Algorithms. 2en ed. 2001. Cormen , Leiserson , Rivest , Stein .**
9. **C The Complete reference(C++, ANCI) , Herbert schild.**
10. **Teach Yourself C++ in 24 hours, Jess Liberty, SAMS.(e-book)**
11. **Problem Solving with C++, 2en ed , Walter savitch.**

مخصوص درس مبانی کامپیوتر
دانشجویان کامپیوتر و IT

آموزش مبانی کامپیوتر و برنامه نویسی به زبان C++

Release 2008

Programming

در این کتاب می خوانید:

- آشنایی با علم کامپیوتر و نسل های آن
- آشنایی با الگوریتم و رسم فلوچارت
- تفاوت برنامه نویسی ساخت یافته با برنامه نویسی غیر ساخت یافته
- ساختارهای تکرار در الگوریتم و در C++
- آرایه ها و بردارها در C++
- مفهوم رویه در الگوریتم و مقدمه ای بر رمزنگاری
- توابع و کلاس های حافظه در C++
- آشنایی مقدماتی با روش های طراحی و تحلیل الگوریتم ها
- اشاره گرها و مرجع
- کامپایل برنامه در VS98 و KDevelop در لینوکس
- انواع بهینه سازی در کامپایلرها
- چرا C++ ؟ چرا جاوا و C# برای مبانی مناسب نیستند ؟
- در این کتاب الگوریتم نویسی ، کشیدن فلوچارت و برنامه نویسی به زبان C++ و تحلیل و طراحی الگوریتم ها را در کنار هم و به صورت قدم به قدم و پیوسته بیاموزید .

برای خرید Online
به آدرس زیر مراجعه کنید
www.naghoospress.ir


9789643773199

زانیس