جزوه برنامه سازی پیشرفته ۲

(زبان C++ و مقدمه ای بر C#)

نویسنده: مهندس حمیدرضا ابراهیمی

عضو هیات علمی دانشگاه آزاد اسلامی واحد فراشبند فارس

فهرست مطالب

1 مقدمه

در این درس شما با اصول برنامه نویسی شئ گرا بکمک زبان C++ بطور فشرده آشنا خواهید شد. سَبک برنامه نویسی که دانشجویان رشته کامپیوتر معمولاً در درس برنامه سازی پیشرفته ۱ (یا همان مبانی کامپیوتر و برنامه سازی) بطور ضمنی با آن آشنا میشوند سبک تابعی یا روالی (Functional , Procedural) نامیده میشود.در زبانهای تابعی مانند C , Pascal , FORTRAN هر دستور برنامه صراحتاً از مترجم (Compiler) میخواهد که کاری نظیر: دریافت ورودی، جمع کردن اعداد و چاپ اطلاعات را انجام دهد. هر برنامه در چنین زبانی متشکل از تعدادی دستور متوالی است.در این نوع زبانها برای سازماندهی و کنترل پیچیدگی برنامه های بزرگ و همچنین سهولت درک برنامه ، دستورات را در قالب مجموعه هایی به نام تابع (Function) طبقه بندی میکنند. همچنین این ایده را میتوان با دسته بندی چندین تابع در قالب یک پیمانه یا Module (که معمولاً در یک فایل نگهداری میشود) توسعه داد.

دسته بندی دستورات یک برنامه در قالب تابع و پیمانه و نیز پیروی از اصول سه گانه توالی ، گزینش و تکرار و نیز عدم استفاده از دستور goto اصول یک **مکتب برنامه نویسی** (Programming Paradigm) به نام **برنامه نویسی ساختیافته** (Structured Programming) را تشکیل میدهند که قبل از ظهور سبک شئ گرا برای چندین دهه سبک غالب و برتر در تولید نرم افزارهای بزرگ محسوب میشد.

با رشد قدرت پردازشی سخت افزار و نیز پیچیده تر شدن روز افزون نرم افزارها ، حتی سبک ساختیافته هم جوابگوی نیازهای تیمهای برنامه نویسی و کنترل هزینه و زمان توسعه نرم افزار نبود. از اینرو دانشمندان در دهه ۸۰ میلادی سبک شئ گرا را طراحی و پیشنهاد نمودند که با الهام از مسائل دنیای واقعی ابداع شده بود.

دو دلیل عمده شکست سبک ساختیافته در پیاده سازی برنامه های رایانه ای بسیار بزرگ عبارتند از:

- دسترسی نامحدود توابع به متغیرها و داده های سراسری
- ضعف مدلسازی دنیای واقعی بکمک توابع و داده های نامرتبط

2 آشنایی با سبک برنامه نویسی شئ گرا

سه اصل اساسی در برنامه سازی شئ گرا عبارتند از:**محصور سازی ، وراثت و چند ریختی**.

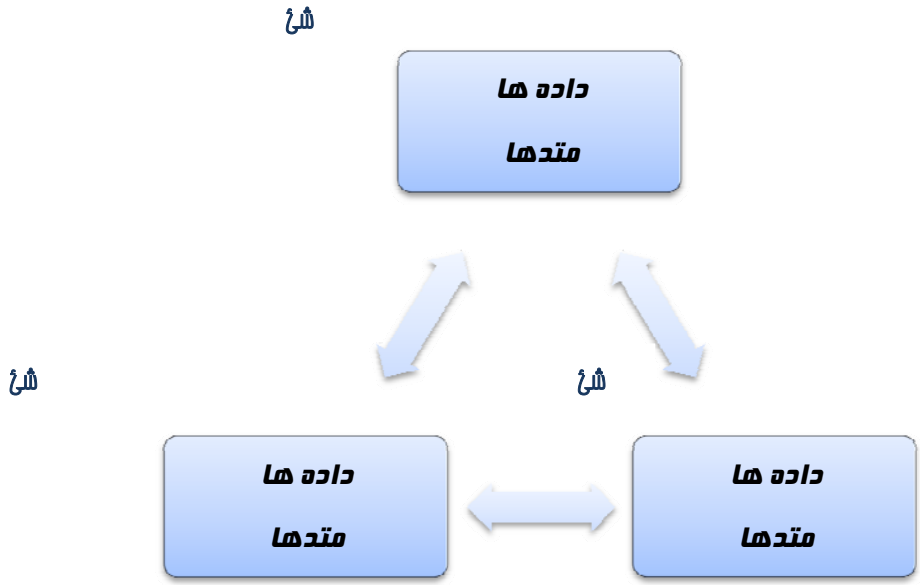
ایده اصلی در برنامه نویسی به سبک شئ گرا (Programming , OOPObject-Oriented) آنستکه داده ها و توابعی که روی آن داده ها عملیات انجام میدهند بایستی در یک واحد ویژه به نام **شئ** (Object) جاسازی و نگهداری شوند.

توابع مربوط به یک شئ که توابع عضو یا متُد (method) نامیده میشوند معمولاً تنها راه دسترسی به داده های آن شئ هستند. بنابراین در دسترسی به اطلاعات یک برنامه نوعی محدودیت ایجاد شده و داده ها به گونه ای پنهان شده اند. بدین ترتیب داده و عملیات مربوط به آن در یک موجودیت واحد محصور میشوند که **اصل اول در برنامه نویسیشئ گرا میباشد (محصور سازی:Encapsulation)**.

البته ایجاد تغییر در داده های یک شئ تنها از طریق متدهای بخصوصی امکانپذیر است و این امر فرآیند اشکال زدایی برنامه ها را به شدت آسان میکند.به عمل فراخوانی متدهای یک شئ ، *ارسال پیام* برای آن شئ هم گفته میشود.

با مطرح شدن سبک شئ گرا و استقبال مهندسان از آن، زبانهای جدیدی مبتنی بر این سبک طراحی و تولید شدند از جمله: C++ , Java , Smalltalk , Object Pascal در دهه ۹۰ و C# در دهه اول قرن ۲۱.

یک برنامه C++ شامل تعدادی شئ است که با فراخوانی متدهای یکدیگر با هم ارتباط برقرار میکنند:



در برنامه نویسی به سبک شئ گرا، برنامه نویس ابتدا به ساکن در گیر جزییات نمیشود بلکه به سازماندهی کل برنامه فکر میکند.

اغلب دستورات برنامه نویسی در C++ مشابه دستورات زبان C هستند و تفاوت اصلی وقتی نمایان میشود که برنامه را از دیدگاه و سطح بالاتری مد نظر قرار دهیم(سازماندهی در قالب اشیاء). برنامه نویس در برخورد با یک مسئله برنامه نویسی متوسط یا بزرگ به سبک شئ گرا ابتدا این سوال را مطرح میکند*!اشیاء تشکیل دهنده این برنامه کدامند؟* بدین ترتیب فرآیند طراحی بطور شگفت انگیزی ساده میشود. این حقیقت ناشی از رابطه نزدیک بین اشیاء برنامه نویسی و اشیاء دنیای واقعی است. نمونه هایی از اشیاء در محیط برنامه نویسی که اغلب از دنیای واقعی الهام گرفته شده اند عبارتند از:

خودرو در مسئله شبیه سازی رایانه ای ترافیک شهری ، قطعات الکترونیکی در برنامه طراحی مدارات الکترونیکی، کشورها در یک برنامه مدلسازی دنیای اقتصاد ، پنجره و منو و صفحه کلید و ماوس در طراحی واسط کاربری یک نرم افزار ، آرایه های سفارشی ، کارمند ، دانشجو ، مشتری ، جدول مختصات جغرافیایی شهرهای دنیا و بسیاری موارد نامحدود دیگر...

در برنامه نویسی شئ گرا اشیاء را بکمک مفهوم رده یا کلاس (class) دسته بندی میکنند .بعنوان مثال، یک شئ به نام پیکان متعلق به یک رده یا کلاس از اشیاء به نام کلاس خودرو محسوب میشود. از اینرو میتوان گفت که: **یک کلاس ، توصیفی از تعدادی شئ مشابه است.**

مفهوم کلاس منجر به توسعه مفهوم دیگری به نام **وراثت** (Inheritance) گردیده است که میتوان آنرا اصل دوم شئ گرایی نامید. برای درک این مفهوم توجه کنید که انسانها در زندگی روزمره از مفهوم کلاس در قالب کلاسهای فرعی کوچکتر یا **زیر کلاس** (subclass) استفاده میکنند. مثلاً گفته میشود موجودات عالم به دو دسته جاندار و بی جان تقسیم میشوند و خود دسته جانداران نیز به دسته های گیاهان و حیوانات تجزیه میشود. در این شیوه طبقه بندی اصل بر آنستکه هر کلاس فرعی ویژگیها و رفتارهای کلاس اصلی را به ارث میبرد و البته میتواند ویژگیهای خاص خود را نیز داشته باشد: گیاهان همانند سایر جانداران به اکسیژن نیاز دارند اما بر خلاف برخی جانداران ، برای ادامه حیات خود عملیات فتو سَنتز انجام میدهند. در این نوع رده بندی ، کلاسی اصلی را **کلاس پایه** (base class) یا **کلاس پدر** مینامند. همچنین کلاسهای فرعی **کلاسهای مشتق** (derived class) نیز نامیده میشوند.

₩تذکر مهم: *رابطه میان اشیاء و کلاسهایشان را با رابطه میان کلاسهای مشتق و کلاسهای پدر اشتباه نگیرید!! اشیاء ، که در حافظه نگهداری میشوند ، دقیقاً در برگیرنده ویژگیها و متدهای کلاس خود هستند ، حال آنکه کلاسهای مشتق ممکن است ویژگیها و متدهای دیگری نیز علاوه بر ویژگیهای کلاس پدر داشته باشند:*

پس از ایجاد و اشکالزدایی یک کلاس میتوان آنرا در اختیار دیگر برنامه نویسان قرار داد تا در برنامه های خودشان از آن استفاده کنند. این قابلیت را قابلیت استفاده مجدد (reusability) میگویند. البته قبلاً مشابه این قابلیت را در استفاده از توابع کتابخانه ای دیده ایم با این تفاوت که دستکاری کردن توابع کتابخانه ای اگر غیر ممکن نباشد بسیار پیچیده است حال آنکه بکمک اصل وراثت براحتی میتوان از کلاسهای موجود کلاسهای دلخواه دیگری ایجاد کرد.

اصل سوم شئ گرایی اصل **چند ریختی** (Polymorphism) نام دارد که بدلیل پیچیدگی نسبی شرح مفصل آنرا به مبحثی اختصاصی در برنامه سازی شئ گرا موکول مینماییم اما برای آشنایی در اینجا به یکی از جنبه های این اصل اشاره میشود:

به بیان ساده ، چند ریختی عبارتست از امکان ایجاد یک متغیر ، تابع یا شئ که دارای شکلهای متعدد باشد.

همچنین بکمک این اصل، میتوان هنگام مشتق نمودن یک کلاس از روی یک کلاس دیگر ، امکان عملکرد متفاوت کلاس مشتق شده روی برخی متدهای کلاس پدر را به وجود آورد.مثلاً اگر کلاس Shape یک کلاس پدر برای توصیف یک شکل هندسی و دارای متدی به نام Draw برای ترسیم آن روی صفحه نمایش باشد، و همچنین کلاسهای Circle و Rectangle دو کلاس مشتق شده از این کلاس باشند ، اصل چند ریختی به برنامه نویس اجازه میدهد متد Draw را در هر کلاس مشتق شده بطور جداگانه و اختصاصی مجدداً تعریف کند چرا که عمل ترسیم برای دایره با عمل ترسیم برای مستطیل تفاوت دارد.اما نحوه فراخوانی این متد مشابه فراخوانی همین متد در کلاس پدر است.

3 مقدمه ای بر C++

در اوایل دهه ۱۹۸۰ میلادی ، شرکت AT &T Bell آمریکا بر آن شد تا نقیصه اصلی زبان پر طرفدار C یعنی عدم پشتیبانی از سبک شئ گرا را بر طرف کند و در این راستا یکی از مهندسان این شرکت به نام Bjarne Stroustrup اقدام به ابداع زبانی به نام C++ نمود که مبتنی بر زبان C اما دارای قابلیت شئ گرایی بود.

از آن زمان به بعد شرکتهای زیادی کامپایلرهایی برای C++ طراحی نمودند (از جمله شرکتهای: Borland, Microsoft) و این امر سبب شد تفاوتهایی بین نسخه های مختلف این زبان بوجود آید و از قابلیت حمل و سازگاری آن کاسته شود. به همین دلیل در سال ۱۹۹۸ زبان C++ توسط موسسه استانداردهای ملی آمریکا (ANSI) به شکل یکپارچه و استاندارد در آمد.

البته متاسفانه با وجود استاندارد سازی این زبان، همچنان برنامه نویسی این زبان در محیطهای مختلف از تفاوتهای نسبتاً قابل توجهی حکایت دارد!!

برنامه های نمونه ارائه شده در این جزوه بدلیل در دسترس بودن و سهولت نصب محیط Turbo C++ 4.5 در این محیط تست شده اند و دانشجویان عزیز میبایست اقدام به تهیه و نصب این مترجم بنمایند.

قبل از ارائه اولین برنامه، شایان ذکر است که برای مقاصد آموزشی معمولاً از برنامه های کنسولی استفاده میشود. این برنامه ها دانشجو را از درگیر شدن با جنبه های غیر ضروری برنامه نویسی تحت ویندوز معاف کرده و در یک محیط متنی ساده اجرا میگردند.بدیهی است دانشجو پس از تسلط به برنامه های کنسولی با کمی مطالعه و تمرین میتواند به سمت برنامه نویسی تحت ویندوز یا هر سیستم عامل گرافیکی دیگر حرکت کند.

مطابق سنت بنیانگذاران زبان C ، برای ورود به دنیای C++ بدون هیچ توضیحی اولین برنامه (و شاید ساده ترین برنامه ممکن) در این زبان برای چاپ پیام *سلام دنیا* را ارائه میدهیم:

```
#include<iostream.h>
void main()
{
     cout<<"Salaam Donya!";
}
```

اکنون به توضیح اجزای این برنامه میپردازیم:

پیش از هر چیز باید گفت از دیدگاه زبان C++ همانند زبان C حروف کوچک و بزرگ با یکدیگر تفاوت دارند. برنامه نویسان معمولاً از حروف کوچک استفاده میکنند مگر آنکه دلیل خوبی برای استفاده از حروف بزرگ داشته باشند.

اولین خط از کد بالا یک راهنمای پیش پردازنده است (پیش پردازنده بخشی از کامپایلر است که ابتدا یک بررسی مقدماتی روی متن برنامه انجام میدهد). علامت مشخصه این خطوط کاراکتر # است. این راهنماها جنبه اجرایی ندارند و جزو دستورات برنامه محسوب نمیشوند. راهنمای پیش پردازنده include به پیش پردازنده دستور میدهد که محتوای یک فایل بخصوص را در متن برنامه فعلی جایگزین کند. این فایلها که فایل سربرگ (Header File) نامیده میشوند معمولاً حاوی یک سری تعاریف ضروری برای استفاده امکانات کتابخانه ای و پیش ساخته محیط برنامه نویسی میباشند.

فایل سربرگiostream.h حاوی تعاریف لازم برای استفاده از توابع ورودی و خروجی و سایر امکانات مربوطه است.*(در برخی محیطهای برنامه نویسیC++ (مانند Visual C++) نیازی به ذکر پسوند فایلهای سربرگ شناخته شده نیست.)*

خط بعدی برنامه اقدام به تعریف تابع اصلی برنامه مینماید. توابع یکی از اجزای اصلی برنامه های C++ هستند و هر برنامه باید شامل یک تابع به نام main() باشد. این تابع حاوی اولین دستور اجرایی برنامه میباشد. یادآور میشود که توابعی که بخشی از یک کلاس باشند متد نامیده میشوند.در زبان C++ بلافاصله پس از نام هر تابع اجباراً یک جفت پرانتز قرار داده میشود در غیر اینصورت کامپایلر آنرا یک تابع تلقی نخواهد کرد.

بدنه یک تابع که شامل تعریف متغیرها و دستورات اجرایی برنامه است در یک جفت علامت آکولاد{ } قرار داده میشود. دستورات بدنه تابع اصلی علاوه بر انجام محاسبات معمولی میتوانند به فراخوانی توابع دیگر یا متدهای اشیاء برای انجام وظیفه مورد نظر بپردازند.

تنها دستور موجود در تابع اصلی برنامه فوق دستور cout است. این دستور از کامپیوتر میخواهد که رشته متنی محصور در زوج علامت " " را روی صفحه نمایش چاپ نماید.در انتهای هر دستور در زبان C++ باید علامت ; قرار داده شود. بطور دقیق تر باید گفت cout یک دستور نیست بلکه یک شئ از پیش تعریف شده است که ما بکمک عملگر یا علامت >> از آن میخواهیم کار بخصوصی برایمان انجام دهد. این عملگر در زبان Cبمعنای انتقال بیتی به سمت چپ است اما در اینجا اصطلاحاً سربارگذاری (overloading) شده و معنای دیگری پیدا کرده است. پس از عملگر>> عبارتی که قصد چاپ آنرا داریم آورده میشود. مثالهای دیگری از این دستور را در زیر مشاهده میکنید:

```
cout<<"Salaam"<<" Donya!!"<<endl; // چاپ همان پیغام قبلی و بردن مکان نما به سر خط بعدی

cout<<" sum is: "<<s;   // چاپ یک پیام و سپس مقدار متغیر s
```

در مثال های بالا، لغت endl بدین معناست که پس از چاپ اطلاعات ، مکان نما به سر خط بعدی برود.

همانطور که میدانید درج توضیحات در متن برنامه ها کمک شایانی به خواناتر شدن و درک معانی تک تک دستورات مینماید. در C++ دو نوع توضیحات وجود دارد: توضیحات تک خطی که با علامت // شروع شده و تا انتهای همان خط ادامه می یابند، و توضیحات چند خطی که در میان یک زوج علامت /* */ ذکر میگردند:

```
// comments
/* another
comment */
```

مثال: برنامه ای برای محاسبه و چاپ حاصلجمع دو عدد صحیح:

```
#include <iostream.h>
void main()
{
     int  a=5, b;
     b=19;
     cout<<"a+b= "<<a+b;
}
```

در مثال فوق دو متغیر از نوع اعداد صحیح به نامهای a و b تعریف شده و اولی با عدد ۵ مقدار دهی شده است. سپس مقدار ۱۹ در متغیر b ریخته شده و حاصلجمع این دو متغیر چاپ گردیده است.علامت انتساب = است.

اسامی متغیرها در زبان C++ میتواند از حروف کوچک و بزرگ زبان انگلیسی و ارقام ۰ تا ۹ و همچنین علامت _ تشکیل شده باشد اما اولین کاراکتر از اسم یک متغیر حتماً باید یک حرف یا علامت _باشد.حداکثر طول اسم یک متغیر حدوداً ۲۵۰ کاراکتر میباشد. ضمناً اسامی متغیرها شناسه (identifier) نیز نامیده میشوند.

مثال: برنامه ای برای تبدیل دما برحسب فارنهایت به سانتی گراد:

```
#include <iostream.h>
void main()
{
     int  ftemp; // درجه حرارت بر حسب فارنهایت
     cout<<"Please enter temperature in fahrenheit:";
     cin>>ftemp;
     int ctemp=(ftemp - 32) * 5 / 9;
     cout<<"Equivalent in Celsius is:"<<ctemp<<endl;
}
```

نکته قابل توجه در مثال فوق، نحوه دریافت اطلاعات از کاربر است. برای این منظور از شئ خروجی و از پیش تعریف شده cin و عملگر >> استفاده شده است.اگر بخواهیم بیش از یک متغیر را با این شئ بخوانیم (مثلاً متغیرهای a , b) بصورت زیر عمل میکنیم:

```
cin>>a>>b;
```

ویژگی دیگر مثال فوق اینستکه متغیر ctemp در همان نقطه ای که به آن احتیاج بوده تعریف شده که این امر در زبان C غیر قانونی بوده و تمامی متغیرهای محلی میبایست در ابتدای تابع تعریف گردند.

مثال: برنامه ای برای محاسبه و چاپ مساحت دایره:

```
#include <iostream.h>
void main()
{
float rad;
constfloat pi = 3.1415F;
   cout<<"Enter radius of circle:";
   cin>>rad;
float area = pi * rad * rad;
   cout<<"Area is: "<<area<<endl;
}
```

همانطور که از متن برنامه بالا حدس میزنید، یکی از راههای نمایش متغیرهای اعشاری در برنامه های C++ معرفی آنها بصورت float است. ضمناً متغیر pi در واقع مقداری ثابت (بدلیل وجود کلمه کلیدی const) و از نوع اعشاری (بدلیل پسوند F بعد از مقدار عددی) میباشد. بنابراین هر تلاشی برای تغییر مقدار این متغیر منجر به تولید خطای کامپایلری خواهد شد.

راه دیگری برای تعریف مقادیر ثابت سراسری در برنامه استفاده از راهنمای پیش پردازنده #define است که یادگاری از زبان C بوده و در C++ توصیه نمیشود.مثلاً برای تعریف مقدار ثابت ۳.۱۴۱۵:

```
#define pi 3.1415
```

دستور فوق باید در اوائل برنامه و قبل از تعریف توابع قرار داده شود.

جدول زیر مشخصات کلی انواع متغیرهای زبان C++ را ارائه میدهد:

نوع متغیر	تعداد بایت در حافظه و محدوده	شرح
unsigned char	۱ بایت- ۰ تا ۲۵۵	کاراکترهای بدون علامت
char	۱ بایت- از ۱۲۸- تا ۱۲۷+	کاراکتر
unsigned int	۲ بایت- ۰ تا ۶۵۵۳۵	اعداد صحیح بدون علامت
int	۲ بایت – از ۳۲۷۶۸- تا ۳۲۷۶۷+	اعداد صحیح
long	۴ بایت- از ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷	اعداد صحیح بزرگ
unsigned long	۴ بایت- از ۰ تا ۴۲۹۴۹۶۷۲۹۵	اعداد صحیح بزرگ بدون علامت
float	۴ بایت- از 3.4E-38 تا 3.4E+38	اعداد اعشاری
double	۸ بایت- از 1.7E-308 تا 1.7E+308	اعدا اعشاری با دقت مضاعف
long double	۱۰ بایت- از 3.4E-4932 تا 1.1E+4932	اعداد اعشاری با دقت مضاعف بسیار بزرگ

قابل ذکر است که در محیطهای برنامه نویسی جدیدتر حجم و گنجایش متغیرها افزایش یافته است.

تبدیل نوع متغیرها و عبارتها:

زبان C++ همانند C در برخورد با عبارتهایی که از چندین متغیر متفاوت به لحاظ نوع تشکیل شده اند سخت گیری کمتری به خرج میدهد.مثلاً دستورات زیر را در نظر بگیرید:

```
int count = 7;
float avgWeight = 155.5F;
double totalWeight = count * avgWeight;
cout<<totalWeight;
```

در اینجا یک متغیر از نوع اعداد صحیح در یک متغیر اعشاری ضرب شده است.کامپایلر هیچ اشکالی در دستورات فوق نمیبیند اما همیشه به متغیرهای با گنجایش (به لحاظ تعداد بایت اشغالی در حافظه) تمایل دارد.پس در خطوط بالا متغیر نوع صحیح ابتدا به نوع اعشاری تبدیل شده و سپس عمل ضرب صورت میگیرد و نهایتاً نتیجه به نوع اعشاری با دقت مضاعف تبدیل میگردد تا قابل ذخیره در متغیر نوع double باشد.ضمناً نوع float به نوع long ترجیح داده میشود.

چنانچه برنامه نویس بخواهد صراحتاً عمل تبدیل نوع متغیرها را انجام دهد از ابزاری به نام Cast استفاده میکند.انواع رایج Cast در C++ عبارتند از **نوع ایستا، نوع پویا و نوع ثابت**. در اینجا نوع ایستا را بررسی میکنیم.برای مثال اگر بخواهیم یک متغیر از نوع اعداد صحیح را به نوع کاراکتری تبدیل نموده و نتیجه را در یک متغیر کاراکتری ذخیره نماییم از دستور زیر استفاده میکنیم:

```
aCharVar = static_cast<char>(anIntVar);
```

متغیر در حال تبدیل در پرانتز و نوع مقصد در داخل یک جفت < > ذکر شده اند.

مثال دیگر: تبدیل به نوع اعشاری با دقت مضاعف `static_cast<double>(intVar)`

عملگرهای محاسباتی:

عملگرهای محاسباتی در جدول زیر خلاصه شده اند:

عملگر	شرح
+, -, *, /	چهار عمل اصلی
%	محاسبه باقیمانده برای تقسیم اعداد صحیح
++ و --	افزایش واحد و کاهش واحد(ویژه متغیرهای صحیح)

زبان C++ (به پیروی از زبان C) برای مختصر نویسی عملگرهای انتساب مرکب را ارائه نموده است. دستور زیر را در نظر بگیرید:

```
x = x + 1;
```

در این دستور متغیر x دو بار ظاهر شده است و اگر طول نام آن چندین کاراکتر باشد مطمئناً تایپ آن برای برنامه نویس بطور چشم گیری وقت گیری وقت گیر خواهد بود. در C++ میتوان دستور فوق را بصورت زیر نوشت:

```
x += 1;
```

عملگرهای انتساب مرکب عبارتند از:

+= , -= , *= , /= , %= , <<= , >>= , |= , &= , ^=

عملگرهای افزایش واحد و کاهش واحد: عبارتند از ++ و --. این عملگرها به دو شکل پیشوندی و پسوندی قابل استفاده هستند و در هر یک از حالات پسوندی و پیشوندی عملکردشان کمی تفاوت دارد. *در شکل پیشوندی (مانند ++i) ابتدا متغیر افزایش یا کاهش می یابد و پس از آن از مقدار متغیر برای سایر محاسبات استفاده میشود ولی در شکل پسوندی (مانند i++) ابتدا از مقدار فعلی متغیر برای محاسبات استفاده میگردد و سپس مقدار متغیر افزایش یا کاهش پیدا میکند.*

مثال: خروجی برنامه زیر را تعیین کنید:

```
#include<iostream.h>
void main()
{
int m=15,n,p;
     n = m++;
     p = ++m;
     cout<<m<<" "<<n<<" "<<p;
}
```

در برنامه بالا ابتدا m++ به داخل n ریخته میشود.بر طبق قانون پسوندی ابتدا مقدار قبلی m یعنی ۱۵ در n ریخته میشود و سپس m یک واحد افزایش میابد(۱۶). پس از آن شکل پیشوندی افزایش واحد m یعنی ++m به p منسوب میشود و در این حالت عدد ۱۷ (یعنی ۱ واحد بیش از مقدار قبلی m) در p ریخته میشود. بنابراین خروجی برنامه فوق بصورت زیر است :

17 15 17

توابع کتابخانه ای: رایج ترین و پر کاربرد ترین توابع کتابخانه ای در اکثر زبانهای برنامه نویسی توابع ریاضی و توابع ورودی و خروجی استاندارد هستند. این گروهها از توابع کتابخانه ای در C++ در فایل سربرگ<iostream.h> و <math.h> تعریف شده اند.

مثال: برنامه ای برای محاسبه و چاپ جذر یک عدد بکمک توابع کتابخانه ای:

```
#include <iostream.h>
#include <math.h>
void main()
{
float x,jx;
     cin>>x;
     jx=sqrt(x);
     cout<<"Jazr(x): "<<jx;
}
```

تابع کتابخانه ای sqrt مقدار جذر پارامتر خود را محاسبه و بعنوان خروجی برمیگرداند.

به خاطر داشته باشید که ممکن است پیاده سازی برخی عملیاتها بدون استفاده از توابع کتابخانه ای پیچیده و پر زحمت باشد!

دستورات گزینش یا تصمیم گیری:

عملگرهای مقایسه ای زبان C++ عبارتند از: < , <= , > , >= , == , !=

بنابراین در این زبان برای بررسی برابری از عملگر== و نابرابری از عملگر=! استفاده میشود.

همچنین عملگرهای منطقی(مورد استفاده در شرطها) عبارتند از:

عملگر	معنا
&&	And
||	Or
!	Not

کلی ترین شکل دستور گزینش دستور if با قالب کلی زیر است:

```
if ( شرط ) {
//دستورات
}
else if( شرط ) {
//دستورات
}
.
.
.
else {
//دستورات
}
```

منطق اجرای این نردبان مشابه سایر زبانهای برنامه سازی است: ابتدا شرط اولین if بررسی شده و اگر درست باشد دستورات زیر آن if اجرا شده و اجرای کل نردبان به پایان میرسد. در غیر اینصورت شرط if بعدی بررسی میگردد و ... نهایتاً اگر همه شرطها نادرست باشند دستورات زیر else اجرا میشوند.ناگفته نماند وجود بخش else اختیاری است و اگر زیر هر قسمت فقط یک دستور وجود داشته باشد میتوان از درج علامت آکولاد خودداری کرد.

مثال: برنامه ای برای تشخیص زوج یا فرد بودن عدد صحیح ورودی

```
#include <iostream.h>
void main()
{
	int n;
	cin>>n;
	if( n % 2 == 0 )
		cout<<"zoj";
	else
		cout<<"fard";
}
```

مثال: برنامه ای برای توضیح نوع رابطه دو عدد ورودی(به لحاظ کوچکتر و بزرگتر بودن)

```
#include <iostream.h>
void main()
{
	inta,b;
	cin>>a>>b;
	if( a>b )
		cout<<a<<" ><<b;
	else if( a == b )
		cout<<a<<" == "<<b;
else
cout<<a<<" < "<<b;

}
```

تذکر مهم:چنانچه از دستورات if بصورت تو در تو استفاده مینمایید بخاطر داشته باشید قانون کامپایلر برای تطبیق if ها با else اینستکه هر else با نزدیکترین if قبل از خود در بلوک جاری جفت میگردد.

مثال: یک برنامه نویس تازه کار ممکن است تصور درستی از نحوه اجرای برنامه زیر نداشته باشد. شما سعی کنید عملکرد این برنامه را توضیح دهید:

```
#include<iostream.h>
void main()
{
	int a, b, c;
	cout <<"Enter three numbers, a, b, and c:\n";
	cin >> a >> b >> c;
	if( a==b )
if( b==c )
		cout <<"a, b, and c are the same\n";
	else
	   cout <<"a and b are different\n";
```

}

حلقه ها:

حلقه های زبان C++ عبارتند از: while { } do , while , for . ساده ترین نوع حلقه به لحاظ یادگیری در این زبان حلقه for است که با قالب کلی زیر استفاده میگردد:

for(تنظیم مقادیر اولیه شمارنده ها ; شرط ادامه حلقه ; بهنگام سازی شمارنده ها) {

// دستورات بدنه حلقه

}

در بسیاری از زبانهای برنامه سازی حلقه for یک حلقه تکرار معین محسوب میشود ولی در C++ این یک حلقه تمام عیار با قابلیت تکرار نامعین است.روند اجرای حلقه فوق بدین ترتیب است که ابتدا بخش تنظیم مقادیر اولیه اجرا شده و سپس شرط حلقه بررسی میشود.در صورت درست بودن شرط یکبار بدنه حلقه اجرا شده ، بخش بهنگام سازی شمارنده ها انجام گرفته و دوباره شرط ادامه بررسی میشود و الی آخر.چنانچه شرط ادامه حلقه ذکر نشود عملاً یک حلقه بی پایان در اختیار خواهید داشت.

مثال: محاسبه و چاپ تعداد شمارنده ها یا همان مقسومٌ علیه های عدد صحیح ورودی

```
#include<iostream.h>
void main()
{
      int n,count=0;
      cin>>n;
      for(int i=1 ; i<=n ; i++)
            if( n % i == 0 ) count++;
      cout<<count;
}
```

در برنامه فوق متغیر شمارنده حلقه در خود حلقه تعریف شده و فقط در داخل بلوک حلقه معنی دارد که این هم یکی دیگر از ویژگیهای جدید C++ محسوب میشود.

در بخشهای تنظیم مقادیر اولیه و بهنگام سازی شمارنده ها میتوان بیش از یک دستور قرار داد.مثلاً:

```
      for(i=0,j=n ; i<j ; i++,j--)
            ...
```

حلقه while دارای قالب زیر است:

```
while ( شرط ادامه حلقه ) {

        دستورات بدنه حلقه //

    }
```

مثال: تشخیص اینکه آیا عدد صحیح ورودی مربع کامل هست یا نه

```
#include <iostream.h>
void main()
{
     int n,i=1;
     cin>>n;
     while( i*i < n )
          i++;
     if( i*i == n )
          cout<<"Perfect square:"<<i<<"*"<<i<<"=="<<n;
     else
          cout<<"Not perfect square";
}
```

بلوک بندی و دسته بندی دستورات:

برای دسته بندی دستورات در C++ از کاراکترهای { } استفاده میگردد.

مثال: برنامه ای برای چاپ اعداد دنباله فیبوناچی کوچکتر از ۱۰۰۰

```
#include<iostream.h>
void main()
{
     unsignedlongint next=0; //جمله ماقبل آخر دنباله فیبوناچی
     unsignedlongint last=1;//جمله آخر دنباله
     while( last < 1000 ) {
     cout<<last<<" ";  // چاپ جمله آخر دنباله
     long sum = next + last;   // محاسبه حاصلجمع دو جمله آخر دنباله
     next = last;  // یک گام پیشروی متغیرها
     last = sum;
     }
}
```

حلقه do { } while نوع سوم دستورات حلقه است.تفاوت اساسی این حلقه با دیگر دستورات حلقه اینستکه شرط این حلقه پس از اجرای بدنه بررسی میشود. بنابراین بدنه این حلقه دست کم یکبار اجرا میشود.قالب این دستور در زیر دیده میشود:

```
do {
```

```
        دستورات بدنه حلقه //

} while ( شرط ادامه حلقه ) ;
```

مثال: اگر برای عدد طبیعی و دلخواه k عدد صحیح دیگری مانند n وجود داشته باشد بطوریکه داشته باشیم: k=n! گوییم عدد k یک عدد فاکتوریل است.برنامه ای بنویسید که اعداد فاکتوریل کوچکتر از عدد ورودی و دلخواه m را چاپ نماید

```
#include <iostream.h>
void main()
{
      int m;
      cout<<"Enter a positive integer:";
      cin>>m;
      cout<<"Factorial numbers less than "<<m<<" are as follows: \n";
      long f=1,i=1;
      do {
            cout<<f<<" , ";
            f *= ++i;
      }
      while( f < m );
}
```

در برنامه فوق دو نکته قابل توجه وجود دارد:

اولاً رشته کاراکتر دو حرفی '\n' نشانه یک کاراکتر ویژه به نام newline است که نتیجه چاپ آن بردن مکان نما به سر خط بعدی است. سایر کاراکترهای ویژه رایج و عملکردشان در C++ در جدول زیر خلاصه شده اند:

کاراکتر ویژه	عملکرد هنگام چاپ
\a	تولید صدای بوق(beep)
\b	Backspace
\t	Tab
\\	چاپ خود کاراکتر \
\'	چاپ کاراکتر '
\"	چاپ کاراکتر "

دستور switch:

این دستور راه دیگری برای پیاده سازی گزینش های چند حالته یا چند راهه است.اما باید دانست که قدرت نردبان if-else-if

در حالت کلی بسیار بیشتر از این دستور میباشد. شکل کلی این دستور بصورت زیر است:

```
switch ( عبارت از جنس صحیح یا کاراکتری ) {
case مقدار ثابت صحیح یا کاراکتری: دستورات اجرایی ; break;
case مقدار ثابت صحیح یا کاراکتری: دستورات اجرایی ; break;
   .
   .
default:   دستورات اجرایی
}
```

این دستور ابتدا مقدار عبارت را محاسبه میکند و سپس میان مقادیر ثابت مقابل هر case از بالا به پایین به دنبال آن مقدار میگردد.اگر مقدار مربوطه پیدا شد دستورات مقابل case مربوطه اجرا شده و کل دستور switch خاتمه یافته تلقی میگردد.اگر مقدار عبارت در میان مقادیر ثابت موجود نباشد دستورات بخش default اجرا میشوند و اگر در این حالت بخش default هم موجود نباشد هیچ عملی صورت نمیگیرد.

در دستور switch نقش دستور break شایان توجه است. این دستور بلوک دستوری جاری را متوقف نموده و کنترل اجرا را به اولین دستور پس از بلوک جاری منتقل میکند.**بنابراین دستور break میتواند برای شکستن و پایان دادن زود هنگام حلقه ها نیز بکار رود.**

بعنوان یک قاعده تجربی در برنامه نویسی، توصیه میشود در صورت امکان در تصمیم گیریهای چند حالته از دستور switch استفاده نمایید مگر آنکه بدلیل مستقل بودن حالات تصمیم گیری از یکدیگر ناچار به استفاده از نردبان if-else-if باشید.

مثال: برنامه ای برای تبدیل یک نمره عددی در مقیاس ۱ تا ۱۰۰ به نمره حرفی معادل(A تا F)

```
#include <iostream.h>
void main()
{
int score;
cout<<"enter your test score:";
cin>>score;
switch( score / 10 ) {
     case 10:
```

```
      case 9:          cout<<"your grade is A";  break;
      case 8:          cout<<"your grade is B";  break;
      case 7:          cout<<"your grade is C";  break;
      case 6:
      case 5:          cout<<"your grade is D";  break;
      case 4: case 3: case 2:
case 1: case 0: cout<<"your grade is F";  break;
      default:         cout<<"Error: score is out of range";
}

}
```

در دستور switch موجود در برنامه فوق، چنانچه ورودی برابر ۱۰ یا ۹ باشد نمره A چاپ میشود.اگر ورودی ۸ باشد نمره B چاپ میشود.اگر ۷ باشد نمره C و اگر ۶ یا ۵ باشد نمره D نمایش داده میشود.در هر یک از حالات ۴ تا ۰ نمره F چاپ میگردد و در غیر اینصورت پیغامی مبنی بر عدم اعتبار ورودی به کاربر نشان داده میشود.

مثال: حلقه for زیر بمحض مشاهده یک عدد بخش پذیر بر ۱۷ متوقف میشود

```
for( i=5000 ; i<7000 ; i++ )
     if( i%17 == 0 )  break
```

عملگر شرطی:

یکی از ویژگیهای زبانهای خانواده C مختصر نویسی است. بسیاری اوقات در برنامه نویسی با شرائطی مواجه میشویم که اگر یک شرط خاص برقرار باشد یک مقدار معین به یک متغیر نسبت میدهیم و در غیر اینصورت مقدار دیگری به آن نسبت میدهیم. مسلماً دستور if به راحتی میتواند این وضعیت را بصورت زیر پیاده سازی کند:

```
if  ( a < b )
min = a;
else
min = b;
```

اما طراحان زبان C برای مختصر نویسی عملگر ? را پیشنهاد کرده اند که دستور بالا را بصورت زیر معادلسازی میکند:

```
min = ( a < b ) ? a : b;
```

با توجه به اینکه اغلب عملگرهای پر کاربرد زبان C++ تا اینجا معرفی شده اند لازم است **جدول اولویت بندی یا تقدم عملگرها** ارائه شود:

اولویت	نوع عملگر	عملگرها
↑افزایش اولویت	یکانی	!,++,--,+,-
	محاسباتی	* , / , %
	محاسباتی	+ , -
	مقایسه ای	< , > , <= , >=
	مقایسه ای	== , !=
	منطقی	&&
	منطقی	||
	شرطی	? :
	انتساب	= , += , -= , *= , /= , %=

با دانستن قوانین مطرح شده در جدول فوق ، در عبارات بدون پرانتز ابهامی در ارزیابی مقدار عبارت بوجود نخواهد آمد.

دستور continue:

برخلاف دستور break در حلقه ها ، این دستور اجرای برنامه را به ابتدای حلقه منتقل میکند(مانند یک پرش به اول حلقه).

بنابراین در صورت اجرای این دستور در داخل یک حلقه ، تکمیل دور فعلی حلقه ناتمام میماند و اجرای دور جدید آغاز میشود.

تمرین: عملکرد برنامه زیر را تجزیه و تحلیل نموده و خروجی آنرا بدست آورید

```
#include<iostream.h>
void main()
{
int i = 0;
do
    {
        i++;
        cout<<"before the continue\n";
continue;
        cout<<"after the continue, should never print\n";
     } while (i < 3);
     cout<<"after the do loop\n";

}
```

سازه ها (structures) : یک سازه عبارتست از مجموعه ای از متغیرهای مرتبط به هم با انواع دلخواه (البته خود این متغیرها نیز میتوانند به نوبه خود سازه باشند).

اجزاء تشکیل دهنده یک سازه را **عضو یا فیلد** مینامند.بعنوان مثال فرض کنید در یک برنامه گرافیکی قصد داریم اطلاعات مربوط به نقاط رنگی صفحه نمایش کامپیوتر را بصورت متمرکز نگهداری و پردازش کنیم. در اینصورت بهتر است برای انسجام برنامه بجای استفاده از متغیرهای پراکنده از سازه ها و بشکل زیر کمک بگیریم:

```
struct point {// تعریف ساختار یک سازه (فضایی در حافظه اشغال نمیکند)
      int x,y; // مختصات نقطه
      int color;// رنگ نقطه
};
void main()
{
      point  p1,p2,p3;//point  تعریف دو متغیر از نوع سازه
      p1.x = 120;
      p1.y = 250;
      p1.color = 0x2AE538;
      p2.x = p1.x -17;
      p2.y = p1.y -29;
      p2.color = 0xFFFFFF;
      p3 = p2;
}
```

کلمه کلیدی struct برای تعریف ساختار یک سازه بکار میرود. در برنامه بالا دو متغیر سازه ای برای نمایش اطلاعات دو نقطه رنگی تعریف شده است و مشخصات این دو نقطه مقدار دهی شده اند. برای دسترسی به اعضای یک سازه نام متغیر سازه ای با نقطه از نام عضو مربوطه جدا میشود.

4 توابع (functions)

یک تابع تعدادی دستور برنامه نویسی را بصورت یک گروه مجزا دسته بندی کرده و یک نام دلخواه به آن گروه نسبت میدهد. از این پس این گروه از دستورات میتواند از سایر قسمتهای برنامه برای اجرا فراخوانی شود.

مهمترین دلیل استفاده از توابع کمک به سازماندهی مفهومی یک برنامه است.تقسیم بندی یک برنامه به تعدادی تابع تصویر کلی تری از برنامه در اختیار قرار میدهد و فرآیند طراحی را آسان میکند. دلیل دیگر برای استفاده از توابع، کاهش تعداد دستورات برنامه است زیرا با این روش مجموعه دستورات پُر تکرار را میتوان فقط یک بار در برنامه ذکر نمود.

مثال: برنامه زیر از یک تابع برای چاپ خطی از علامتهای * در خروجی و سپس نمایش محدوده متغیرها استفاده میکند

```
#include<iostream.h>
void starline();  // الگوی تابع ) اعلام وجود تابع
void main()
{
     starline();//فراخوانی (اجرا) تابع
     cout<<"Data type range"<<endl;
     starline();//فراخوانی (اجرا) تابع
     cout <<"char -128 to 127"<< endl
<<"short -32,768 to 32,767"<< endl
<<"int System dependent"<< endl
<<"long -2,147,483,648 to 2,147,483,647"<< endl;
    starline();//فراخوانی (اجرا) تابع
}

void starline() //تعریف تابع
{
     for(int j=0; j<45; j++)
     cout <<'*';
     cout << endl;
}
```

در هنگام اعلام وجود توابع قبل از تعریف توابع بایستی نوع مقدار برگشتی و پارامترهای تابع را مشخص نمایید.چنانچه از اعلام وجود تابع صرفنظر کنید با خطای کامپایلری مواجه خواهید شد.

برنامه فوق از دو تابع main() و starline() تشکیل شده است.

الگوی تعریف توابع بصورت زیر است:

نام تابع(مشخصات پارامترها) نوع مقدار برگشتی یا همان خروجی {

بدنه تابع(شامل تعریف متغیرهای محلی و دستورات اجرایی)//

}

توابع معمولاً اطلاعات اولیه مورد نیاز برای انجام وظائفشان را از طریق پارامترها دریافت میکنند. مثلاً اگر بخواهیم به جای تابع starline() در مثال قبلی تابعی به نام repchar بنویسیم که بتواند یک کاراکتر دلخواه را بجای علامت * چاپ کند بصورت زیر عمل میکنیم:

```
#include<iostream.h>
void repchar(char, int); // الگوی تابع
void main()
{
     repchar('-', 43); / - فراخوانی تابع با کاراکتر/
```

```
    cout <<"Data type Range"<< endl;
    repchar('=', 23); // = فراخوانی تابع با کاراکتر
    cout <<"char -128 to 127"<< endl
    <<"short -32,768 to 32,767"<< endl
    <<"int System dependent"<< endl
    <<"double -2,147,483,648 to 2,147,483,647"<< endl;
    repchar('-', 43);
}
//-------------------------------------------------------------
// تابع زیر کاراکتر داده شده در پارامتر اول را به تعداد داده شده در
// پارامتر دوم چاپ میکند
void repchar(char ch, int n)
{
    for(int j=0; j<n; j++)
        cout << ch;
    cout << endl;
}
```

در برنامه فوق میتوان به جای ارسال مقادیر ثابت به تابع ، پارامترهای n و ch را به دلخواه کاربر در تابع اصلی از ورودی دریافت نموده و به تابع ارسال کرد.

تذکر مهم:در برنامه فوق تابع پس از دریافت پارامترها بازای هر کدام یک متغیر موقتی محلی (همنام با اسم ذکر شده در تعریف تابع) برای نگهداری مقدار آنها ایجاد میکند و این روش پیش فرض را ارسال پارامتر به شیوه مقداری مینامند.

نکته: بطور پیش فرض ، سازه ها هم در صورت ارسال بعنوان پارامتر برای توابع بصورت مقداری فرستاده میشوند.

دستور return:این دستور میتواند همزمان دو عمل انجام دهد:

۱. *خروج از تابع و رفتن به نقطه بعد از فراخوانی آن در تابع فرا خواننده*
۲. *بازگرداندن مقدار به تابع فراخواننده*

مثال: تابع زیر پارامتری را بعنوان وزن بر حسب واحد پوند دریافت نموده و معادل آن بر حسب کیلوگرم را بعنوان خروجی تابع برمیگرداند

```
float lbstokg(float pounds)
{
    float kilograms = 0.453592 * pounds;
    return kilograms;
}
```

پارامترهای ارجاعی (reference arguments):

منظور از ارجاع ، یک نام مستعار برای یک متغیر است. مهمترین کاربرد ارجاعها ارسال پارامترها به شیوه ارجاعی میباشد.ارجاعها در حقیقت به جای مقدار، آدرس متغیرها را در اختیار قرار میدهند.بدین ترتیب اگر یک پارامتر به شیوه ارجاعی به یک تابع فرستاده شود، تابع فراخوانده شده به مقدار آن متغیر در تابع فراخواننده دسترسی نامحدود خواهد داشت(و باید در این زمینه بسیار با احتیاط عمل کرد).

مثال: برنامه زیر بکمک یک تابع یک عدد اعشاری دریافت نموده و جزء صحیح و جزء اعشاری آنرا چاپ میکند

```
#include <iostream.h>
void main()
{
     void intfrac(float, float&, float&); // اعلام وجود تابع
     float number, intpart, fracpart;
     do {
     cout <<"\nEnter a real number: ";
     cin >> number;
     intfrac(number, intpart, fracpart); //یافتن جزء صحیح و اعشار عدد
     cout <<"Integer part is "<< intpart
     <<", fraction part is "<< fracpart << endl;
     } while( number != 0.0 ); //حلقه با ورودی ۰.۰ خاتمه میابد
}
//---------------------------------------------------
// intfrac()
// این تابع جزء صحیح و جزء اعشار اعداد حقیقی را محاسبه میکند
void intfrac(float n, float& intp, float& fracp)
{
     long temp = static_cast<long>(n); //تبدیل به اعداد صحیح بزرگ
     intp = static_cast<float>(temp); //تبدیل به اعداد ممیز شناور
     fracp = n - intp;
}
```

اولین نکته قابل ملاحظه در برنامه فوق، ذکر الگوی تابع intfrac در داخل تابع اصلی است. بدین ترتیب کامپایلر کل فایل متن برنامه را برای پیدا کردن تعریف این تابع جستجو خواهد کرد.

نکته دوم اینکه پارامترهای دوم و سوم تابع intfrac با چسباندن یک علامت & به کلمه float (یا هر نوع مورد نظر دیگر) به شیوه ارجاعی ارسال شده اند.پس این تابع بدون استفاده از دستور return میتواند از طریق این پارامترها مقدار مورد نظر را به تابع فراخواننده باز گرداند.در فراخوانی تابع علامت & ذکر نمیشود.

نکته سوم اینستکه اسامی intp و intpart عملاً دو نام برای یک متغیر هستند و بطریق مشابه اسامی fracp و fracpart.

سربارگذاری توابع (Overloaded Functions):

منظور از سربار گذاری یک تابع تعریف مجدد آن با پارامترهای متفاوت(تعداد یا نوع یا هر دو) برای داشتن عملکردی متفاوت میباشد. *این ویژگی در زبان C وجود ندارد* .یکی از مزایای این قابلیت، رفع نیاز به چندین نام متفاوت برای یک عملیات مشترک است.

(نکته بسیار مهم: در سربار گذاری، تفاوت تعاریف مجدد توابع نمیتواند صرفاً در نوع خروجی تابع باشد)

مثال: تابع چاپ خط کاراکتری با سه الگوی کاربری متفاوت

```
#include<iostream.h>
//معرفی یک تابع با ۳ الگوی متفاوت
void repchar();
void repchar(char);
void repchar(char, int);
void main()
{
     repchar();
     repchar('=');
     repchar('+', 30);
}
//---------------------------------------------------------------
// repchar()
// نمایش ٤٥ علامت ستاره
void repchar()
{
     for(int j=0; j<45; j++)
          cout <<'*';
     cout << endl;
}
//---------------------------------------------------------------
// repchar()
// نمایش کاراکتر ارسالی به تعداد ٤٥ بار
void repchar(char ch)
{
     for(int j=0; j<45; j++)
          cout << ch;
     cout << endl;
}
//---------------------------------------------------------------
// repchar()
// نمایش کاراکتر ارسالی به تعداد دفعات مورد نیاز( پارامتر دوم)
void repchar(char ch, int n)
```

```
{
     for(int j=0; j<n; j++)
          cout << ch;
     cout << endl;
}
```

توابع بازگشتی (recursive functions):

تابع بازگشتی تابعی است که اقدام به فراخوانی و اجرای خودش مینماید. تکنیک قدرتمند بازگشتی نقش بزرگی در حل مسائل پیچیده برنامه نویسی و مدلسازی بسیاری از مفاهیم ریاضی بکمک رایانه داشته و دارد (هر چند در کاربردهای با مقیاس واقعی برنامه های بازگشتی بعضاً ناکارآمد هستند).

ابداع روش بازگشتی در حقیقت ریشه در برخی مفاهیم ریاضی مانند فاکتوریل دارد. در کتب ریاضی معمولاً فاکتوریل یک عدد بصورت زیر تعریف میشود:

$$n! = \begin{cases} 1 & \text{اگر} \quad n = 0 \\ n * (n-1)! & \text{برای} \quad n > 0 \end{cases}$$

میبینید که در بیان مفهوم فاکتوریل دوباره از مفهوم فاکتوریل استفاده شده است که به نظر بی معنی است اما نکته کلیدی در اینجا وجود شرطی برای پایان خودفراخوانی است: شرط n=0.

تابع بازگشتی فاکتوریل را در زیر ملاحظه میفرمایید:

```
long fact(int n){
     if( n==0 )
          return 1;
     else
          return n * fact(n-1);
}
```

تمرین: تابعی بازگشتی برای محاسبه ب.م.م دو عدد صحیح بنویسید

توابع در جا (inline functions):

میدانیم که یکی از مزایای توابع ، صرفه جویی در مصرف حافظه برای مجموعه دستورات تکراری است، اما بهای پرداختی در این زمینه ، نیاز به اجرای چند دستور خاص برای پرش به محل اولین دستور تابع و نیز برگشت از تابع میباشد. بنابراین میتوان به این نتیجه رسید که **تعریف توابع با تعداد دستورات کم به لحاظ وقت تلف شده از پردازنده مقرون به صرفه نیست!**

ایده توابع در جا بر این اساس میگوید برای توابع کوتاه(که حافظه چندانی هم مصرف نمیکنند) بهتر است دستورات تابع را درست در همان مکانی که فراخوانی میگردد جاسازی (و احتمالاً تکرار) کنیم تا نیازی به انجام عملیات اضافی و پرش نباشد.

مثال: تابعی در جا برای تبدیل وزن از واحد پوند به واحد کیلوگرم

```
#include <iostream.h>
// lbstokg()
// تابعی درجا برای تبدیل واحد پوند به کیلوگرم
inlinefloat lbstokg(float pounds)
{
     return 0.453592 * pounds;
}
//-------------------------------------------------------------
void main()
{
     float lbs;
     cout <<"\nEnter your weight in pounds: ";
     cin >> lbs;
     cout <<"Your weight in kilograms is "<< lbstokg(lbs)
     << endl;
}
```

در برنامه فوق از کلمه کلیدی inline برای معرفی این تابع بصورت در جا استفاده شده و هر گاه این تابع فراخوانی شود کامپایلر بدنه آنرا در نقطه فراخوانی کپی میکند تا نیازی به پرش و عملیات برگشت نباشد.

پارامترها یا آرگومانهای با مقدار پیش فرض:

در تعریف یک تابع چنانچه برای برخی پارامترهای آن مقدار پیش فرض اعلام نماییم آزادی عمل بیشتری در فراخوانی آن تابع بدست خواهیم آورد.یعنی در تابع فراخواننده میتوانیم مقدار برخی از پارامترها را ذکر نکنیم.

مثال: تابع repchar با مقادیر '*' و ۴۵ برای پارامترهایش (جایگزینی برای تابع سربارگذاری شده در مثالهای قبلی!)

```
#include<iostream.h>
void repchar(char='*',int=45);//معرفی تابع بامقادیرپیش فرض برای پارامترها
void main()
{
```

```
        repchar(); //چاپ ٤٥ ستاره
        repchar('='); // چاپ ٤٥ علامت =
        repchar('+', 30); // چاپ ٣٠ علامت +
}
//-------------------------------------------------------------
// repchar()
void repchar(char ch, int n)
{
        for(int j=0; j<n; j++)
        cout << ch;
        cout << endl;
}
```

تذکر مهم:پارامترهای دارای مقادیر پیش فرض ، بایستی جزو پارامترهای انتهایی در فهرست پارامترها باشند، مثلاً اگر در یک فهرست پارامتر ۵ تایی پارامتر سوم دارای مقدار پیش فرض باشد پارامترهای چهارم و پنجم نیز باید اینگونه باشند و گرنه با پیغام خطای کامپایلری مواجه خواهید شد.

کلاس حافظه (storage class):

طبق تعریف، طول عمر یک متغیر عبارتست از مدت زمان سپری شده از لحظه ایجاد آن تا لحظه نابودی آن. اصطلاح کلاس حافظه برای بیان طول عمر و زمان ایجاد و نابودی متغیرهاست.متغیرهای محلی در لحظه فراخوانی و ورود به تابع مربوطه ایجاد شده و پس از خروج از تابع از بین میروند.از اینرو کلاس حافظه یا کلاس ذخیره سازی این متغیرها را خودکار یا **automatic** مینامند.

در مقابل، متغیرهای سراسری، یعنی آنهایی که در فضای بین توابع تعریف شده اند دارای کلاس حافظه **static** هستند بدین معنا که طول عمر آنها برابر زمان اجرای کل برنامه است و نیز از نقطه تعریف تا انتهای فایل برنامه شناخته شده هستند (بر خلاف متغیرهای محلی که فقط در تابع خودشان قابل استفاده هستند).

تنها تفاوت متغیرهای محلی **static** با متغیرهای سراسری اینستکه متغیرهای محلی ایستا از لحظه فراخوانی تابع مربوطه ایجاد میشوند.

برای تعریف متغیرهای محلی بصورت **static** کافیست کلمه کلیدی static را قبل از ذکر نوع داده یی آنها قید کنیم:

```
staticint  m; // a  static  local  variable
```

مثال: محاسبه میانگین تعدادی عدد بطوریکه با ورود هر عدد جدید میانگین جدید محاسبه و چاپ گردد

```
#include <iostream.h>
```

```
float getavg(float);
void main()
{
     float data=1, avg;
     while( data != 0 )
     {
          cout <<"Enter a number: ";
          cin >> data;
          avg = getavg(data);
          cout <<"New average is "<< avg << endl;
     }
}
//---------------------------------------------------------------
// getavg()
// محاسبه و برگرداندن میانگین اعداد قبلی همراه با عدد جدید
float getavg(float newdata)
{
staticfloat total = 0;// متغیرهای محلی ایستا فقط یکبار مقداردهی اولیه
staticint count = 0; (در اولین فراخوانی تابع) //میشوند
count++; //نگهداشتن تعداد اعداد ورودی تا بحال
total += newdata;
return total / count; //برگرداندن میانگین جدید
}
```

مقدار برگشتی ارجاعی (return by Reference):

میدانیم که یک تابع در حالت عادی در صورت نیاز مقداری را بعنوان خروجی برمیگرداند. اما این امکان وجود دارد که از تابع بخواهیم آدرس مقدار خروجی خود را به تابع فراخواننده ارسال کند و این ویژگی را مقدار برگشتی ارجاعی مینامند.

یکی از انگیزه های ابداع این قابلیت، عدم نیاز به کپی کردن یک متغیر خروجی پرحجم (مانند یک شئ بزرگ) بوده است.

مثال: خروجی برنامه زیر را تعیین کنید

```
#include <iostream.h>
int x; // متغیر سراسری
int& setx(); //int  معرفی تابع با خروجی ارجاعی از نوع
void main()
{ // !!!جدید و عجیب:از یک تابع در سمت چپ دستور انتساب استفاده شده
//:هم تابع اجرا میشود هم از آدرس خروجی آن بعنوان متغیر استفاده شده
     setx() = 92;
     cout <<"x="<< x << endl;
}
//---------------------------------------------------------------
int& setx()
{
```

```
    return x; // آدرس متغیر مربوطه برگردانده میشود
}
```

دیده میشود که تابع setx() در حقیقت آدرس متغیر سراسری x را بعنوان خروجی برمیگرداند،پس با اجرای اولین دستور تابع اصلی عدد ۹۲ در x ریخته میشود! و خروجی بصورت x=92 خواهد بود.

5 شئ گرایی در C++ و سربارگذاری عملگرها

در فصل دوم مفاهیم مقدماتی در برنامه سازی شئ گرا ذکر شد.در این فصل مفاهیم شئ گرایی را در برنامه نویسی بکار میگیریم.نخست چگونگی تعریف کلاس :

```
class نام کلاس  {
    private:
        //بخش خصوصی کلاس
    public:
        //بخش عمومی کلاس
};
```

وجود علامت ; در انتهای تعریف کلاس اجباری است.

در تعریف کلاس به تجربه رسم بر اینستکه داده های یک کلاس در بخش خصوصی و عملیاتهای مربوط به کلاس در بخش عمومی تعریف گردند(البته برنامه نویسان حرفه ای ممکن است به صلاحدید خود گاهی اوقات دسته بندی خصوصی و عمومی را بگونه دیگری انجام دهند) .**اطلاعات تعریف شده در بخش خصوصی محرمانه تلقی شده و از خارج از کلاس قابل مشاهده و استفاده نیستند.**

مثال: تعریف کلاسی برای نمایش یک مستطیل و عملیات مربوطه

```
#include<iostream.h>
class rectangle {
private:
    int  length,width;// اطلاعات حساس در بخش خصوصی ذکر میشوند
```

```
public:
// متد سازنده
     rectangle(int l=10,int w=5)  { length = l; width = w; }
// متدی برای برگرداندن طول مستطیل
     int get_length() {  return length; }
     int get_width(){  return width; }// متدی برای برگرداندن عرض مستطیل
     int area() {  return length*width; } // متد محاسبه مساحت مستطیل
// متدچاپ مقادیر طول و عرض مستطیل
     void print() {  cout<<"Length: "<<length<<" Width:"<<width; }
};
void  main(){
     int c;
     rectangle r1(17,6); // ایجاد یک شئ مستطیل با طول ۱۷ و عرض ٦
     rectangle r2;// ایجاد یک مستطیل با مقادیر طول و عرض پیش فرض
     r1.print(); // چاپ مقادیر طول و عرض مستطیل
     cout<<"\nRectangle r1 area:"<<r1.area();
}
```

در ابتدای برنامه فوق یک کلاس تعریف شده و در تابع اصلی از این کلاس نمونه برداری شده و ۲ شئ از روی آن ساخته شده است. شئr1 یک مستطیل با طول و عرض ۱۷ و ۶ میباشد و شئr2 دارای طول و عرض پیش فرض کلاس خواهد بود(چون برنامه نویس در اینجا طول و عرض را قید نکرده است). بنابراین اگر شئr3 را بصورت;rectangle r3(9) تعریف نماییم طول آن برابر ۹ میشود اما عرض آن مقدار پیش فرض ۵ خواهد بود.سپس تابع عضو یا همان متد print() از کلاس برای چاپ مشخصات مستطیل و متد area() برای محاسبه مساحت آن فراخوانی گردیده اند.

منظور از متّد سازنده (**constructor**) که همنام کلاس تعریف میشود و خروجی برای آن تعریف نمیگردد متدی است که بمحض نمونه برداری از کلاس (یعنی تعریف اشیاء جدید از روی کلاس) اجرا میشود و کاربرد اصلی آن تنظیم مقادیر اولیه داده های کلاس(بخصوص داده های خصوصی) به میل برنامه نویس و زمینه سازی استفاده از شئ است.ضمناً متد سازنده میتواند سربارگذاری شود(دارای شکلهای متعدد باشد).*ساده ترین متد سازنده یک کلاس که هیچ پارامتری ندارد متد سازنده پیش فرض نامیده میشود.*

از آنجا که دسترسی مستقیم به اعضای خصوصی یک کلاس امکانپذیر نیست برای امکان آگاهی دیگر اجزاء برنامه از مشخصات یک شئ مستطیل دو متد get_length() و get_width() تعریف شده اند.این دو تابع را **توابع دستیابی** مینامند.

(نکته : اگر پیاده سازی متدهای یک کلاس در خارج از کلاس انجام شود کلاس مربوطه را رابط (interface) نیز مینامند)

متد سازنده کپی:

چنانچه بخواهیم یک شئ را از روی یک شئ دیگر تعریف نماییم بایستی در تعریف کلاس مربوطه یک متد ویژه را از قبل طراحی نموده باشیم. دستور rectangle r2(r1); بر طبق قرارداد یک شئ جدید به نام r2 با مشخصات شئ موجود r1 میسازد.برای اِعمال کنترل روی این فرآیند برنامه نویس باید متد ویژه ای به نام متد سازنده کپی را در تعریف کلاس بگنجاند(در غیر اینصورت کامپایلر از یک سازنده کپی پیش فرض استفاده خواهد نمود).

متد سازنده کپی متدی است که هنگام کپی برداری از روی یک شئ بطور اتوماتیک فراخوانی میشود و میتواند اجزاء اطلاعاتی شئ جدید را تنظیم نماید.این متد نیز باید همنام کلاس و بدون خروجی تعریف شود اما لازم است یک پارامتر از نوع ارجاع ثابت و همجنس کلاس داشته باشد.

مثال: سازنده کپی برای کلاس rectangle

```
class rectangle {
private:
     int  length,width;
public:
rectangle(int l=10,int w=5)  { length = l; width = w; } // متد سازنده
// متد سازنده کپی
rectangle(const rectangle& r){ length = r.length; width = r.width; }
int get_length() {  return length; }
int get_width(){  return width; }
int area() {  return length*width; }
void print() {  cout<<"Length: "<<length<<" Width:"<<width; }
};
```

نکته: دستوری مانند **rectangle r2=r1;** نیز منجر به فراخوانی متد سازنده کپی خواهد شد.

متد نابودگر:

پس از پایان عمر یک شئ بایستی عملیاتی برای آزاد سازی منابع مورد استفاده شئ (از جمله حافظه) انجام بگیرد.انجام این عملیات بر عهده یک متد ویژه به نام نابودگر میباشد. تعریف **متد نابودگر** اختیاری است و همنام کلاس میباشد اما قبل از نام آن کاراکتر ~ قرار داده میشود.هر کلاس فقط یک نابودگر میتواند داشته باشد.

پیاده سازی متدها در خارج از بدنه کلاس:

چنانچه بخواهیم جزییات پیاده سازی و عملکرد متدهای یک کلاس را در خارج از بلوک تعریف کلاس انجام دهیم بایستی از عملگر تعیین قلمرو یعنی :: برای اعلام رابطه بین متد و کلاس استفاده نماییم.

مثال: تعریف متد area() در خارج از کلاس rectangle

```
class rectangle {
private:
     int  length,width;
public:
rectangle(int l=10,int w=5)  { length = l; width = w; } // متد سازنده
// متد سازنده کپی
rectangle(const rectangle& r){ length = r.length; width = r.width; }
int get_length() {  return length; }
int get_width(){  return width; }
int area();
void print() {  cout<<"Length: "<<length<<" Width:"<<width; }
};

rectangle::area() {  return length*width; }
```

اعضای ایستا برای کلاسها:

چنانچه یک عضو داده یی از یک کلاس بصورت ایستا یا static تعریف شود، در زمان اجرا فقط یک نمونه از آن عضو برای آن کلاس و تمامی اعضاء آن ساخته خواهد شد و این عضو بین تمامی نمونه های کلاس در دسترس و مشترک خواهد بود.

طول عمر اعضای ایستای کلاسها برابر زمان اجرای کل برنامه است.

مثال: شمارش تعداد اشیاء ساخته شده از کلاس

```
#include <iostream.h>
class myclass {
private:
     staticint  objCount;
public:
     myclass() {   objCount++; }
     int  getcount(){  return objCount; }
};
int myclass::objCount=0;// بسیار مهم: تعریف و مقداردهی عضو ایستا
void  main(){
     myclass m1,m2;
     cout<<m1.getcount();
}
```

متد getcount() در کلاس فوق تعداد اشیایی که تا کنون از کلاس فوق نمونه برداری شده را باز میگرداند. باید دانست که تعریف و مقدار دهی عضو ایستای یک کلاس در خارج از کلاس اهمیت بسیار زیادی داشته و در غیر اینصورت با خطا مواجه خواهیم شد.

6 آرایه ها، رشته ها و اشاره گرها

برای گروهبندی تعدادی متغیر همنوع در اغلب زبانهای برنامه نویسی از آرایه استفاده میشود.داده های ذخیره شده میتوانند شامل انواع ساده مانند int و float یا اشیاء پیچیده تعریف شده توسط کاربر باشند.آرایه ها در زبان C++ شبیه آرایه های زبان C هستند اما در C++ راه دیگری نیز برای دسته بندی داده های همنوع وجود دارد و آن استفاده از بردارها (Vectors) میباشد که در بخش قالبها بررسی خواهند شد.

نشانه مشخصه یک آرایه در زبان C++عملگر زیرنویس یا همان [] است. الگوی کلی تعریف یک آرایه بصورت زیر است:

نوع داده یی نام آرایه [تعداد عناصر];

مثال:

```
int    a[10]; // یک آرایه ۱۰ تایی از اعداد صحیح
//یک آرایه ۵ تایی از اعداد اعشاری با مقادیر اولیه
float  b[]={2,-1,4.75,3.02,0.25};// یک آرایه ۵ تایی با مقادیر اولیه
int    n=9,c[n];// خطا:تعداد خانه های آرایه هنگام تعریف باید مقداری ثابت باشد
```

بر طبق قرارداد، زیرنویس اولین خانه آرایه در زبان C++همواره صفر است.

مثال: خواندن و پر کردن خانه های اول تا nام یک آرایهa

```
for(int i=0; i<n ; i++)
    cin>>a[i];
```

هشدار: *هنگام کار با آرایه ها مراقب باشید مقادیر زیرنویسها از محدوده قانونی تجاوز نکنند و گرنه با خطای زمان اجرا مواجه خواهید شد!*

مثال: تعریف یک آرایه دو بعدی ۳*۵ (ماتریس ۳*۵) و پر کردن خانه های آن

```
int    mat[5][3];
for(int i=0 ; i<5 ; i++)
for(int j=0; j<3; j++)
         mat[i][j]=i*j;
```

مثال: مقداردهی اولیه به یک آرایه دو بعدی

```
int  mat[][];{  { 2 ,3 ,-1, -5},
                { 4, 5 ,-2, -4},
                { 6, 8, -9, -6} };  //  یک ماتریس ۳*٤
```

مثال: برنامه ای برای خواندن یک ماتریس ۳*۳ و محاسبه و چاپ ترانهاده آن

```
#include <iostream.h>
void  main(){
     int  mat[3][3],i,j,t;
     cout<<"Please enter a 3*3 matrix row by row: \n";
     for(i=0; i<3 ; i++)
          for(j=0; j<3 ; j++)
               cin>>mat[i][j];
     for(i=0; i<3 ; i++)
          for(j=0; j<i ; j++)
          {  // تعویض جای عناصر طرفین قطر اصلی
               t=mat[i][j]; mat[i][j]=mat[j][i]; mat[j][i]=t;          }
     cout<<"Transposed matrix:\n";
     for(i=0; i<3 ; i++){
          for(j=0; j<3 ; j++)
               cout<<mat[i][j]<<" ";
          cout<<endl;
     }
}
```

***نکته:** هنگام ارسال آرایه ها بعنوان پارامتر برای توابع از نام آرایه بدون علامت [] استفاده میشود،همچنین نام آرایه به تنهایی حاوی آدرس اولین خانه آرایه میباشد.*

مثال: آرایه ای از سازه ها

```
#include <iostream.h>
struct point {
     int x,y;
     int color;
};
void  main(){
     point  triangle[3];// آرایه ای از ۳ سازه
     cout<<"Enter coordinates & color of the triangle:\n";
     for(inti=0; i<3 ; i++){
          cout<<"x:";
          cin>>triangle[i].x;
          cout<<"y:";
          cin>>triangle[i].y;
          cout<<"color:";
          cin>>triangle[i].color;
     }
```

```
}
```

مثال: پیاده سازی کلاسی برای ساختمان داده یی پشته یا stack بکمک آرایه ها (پشته نوعی ساختمان داده یی است که عملیات حذف و اضافه کردن عناصر تنها از یک سمت آن امکانپذیر است(مانند خشاب اسلحه))

```
const int  MAX=1000;
class Stack
{
private:
     int st[MAX]; // آرایه نگهدارنده عناصر پشته
     int top; //زیرنویس عنصر بالای پشته
public:
     Stack() { top = 0; }// پشته خالی
     void push(int var) //وارد نمودن اطلاعات به پشته
     { st[++top] = var; }
     int pop() // برداشتن اطلاعات از روی پشته
     { return st[top--]; }
};
```

رشته های کاراکتری:

روش سنتی تعریف رشته های کاراکتری برگرفته از زبان C استفاده از آرایه ای از کاراکترها میباشد با این قرارداد که پس از آخرین کاراکتر ، کاراکتر تهی یعنی '\0' با کد اسکی صفر قرار داده شود:

```
     char  name[10]; // رشته ای بطول حداکثر ۹ کاراکتر
     char  family[]="alavi"; // رشته ای به طول ۵ کاراکتر(۶ بایت نیاز دارد
     cout<<family[5]; //چاپ کاراکتر تهی که در انتهای رشته قرار داده شده
     cin>>name;// خواندن رشته
```

اگر رشته ورودی حاوی کاراکتر بلانک یا جای خالی نیز باشد برای خواندن آن از صفحه کلید بایستی از روش زیر استفاده شود:

```
     char  name[20];
     cin.get(name,20);//حداکثر ۱۹ کاراکتر میخواند(یکی برای کاراکتر تهی)
```

روش بالا برای خواندن کاراکترهای شامل چندین خط متن جوابگو نیست زیرا از کاراکتر '\n'بعنوان علامت پایان رشته استفاده میکند.برای رفع مشکل فوق و خواندن رشته های حاوی متون چند خطی میتوان از یک پارامتر سوم برای متد get() جهت اعلام کاراکتر پایان رشته بصورت زیر استفاده نمود:

```
     char  name[20];
     cin.get(name,20,'$');// کاراکتر پایان ورودی رشته $ اعلام میشود
```

توابع کتابخانه ای ویژه کار با رشته های کاراکتری در فایل سربرگstring.h معرفی شده اند.از جمله این توابع میتوان به تابع strlen() برای محاسبه طول رشته و strcpy() برای کپی کردن یک رشته در رشته دیگر اشاره نمود:

```
char s1[20] , s2[20]=”HamidRezaEbrahimi”;
cout<<strlen(s2);
strcpy(s1,s2); //s1 در s2 کپی نمودن رشته
```

برای نگهداری تعدادی رشته کاراکتری (مثلاً تعدادی اسم) میتوان از آرایه ای از رشته ها که در حقیقت یک آرایه دو بعدی از کاراکترها است استفاده نمود:

```
const int DAYS = 7; //تعدا رشته ها در آرایه
const int MAX = 10; //حداکثر اندازه هر رشته
//آرایه ای از رشته ها
char star[DAYS][MAX] = { "Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday",
"Friday", "Saturday" };
for(int j=0; j<DAYS; j++) //چاپ رشته ها
cout << star[j] << endl;
```

استفاده از آرایه ای از کاراکترها برای نمایش رشته های کاراکتری روشی قدیمی محسوب میشود (فقط در موارد خاص از آن استفاده میگردد) و دارای معایبی از جمله عدم امکان انتساب یک رشته به رشته دیگر است. روش نوین و استاندارد C++ برای کار با رشته ها استفاده از کلاس stringاست که اغلب عملگرهای زبان را برای رشته ها سربارگذاری نموده است.

برای استفاده از کلاس string در Turbo C++ بایستی فایل سربرگcstring.h در ابتدای برنامه شامل گردد.

مثال: آشنایی با کلاس استاندارد string

```
#include <iostream.h>
#include <cstring.h>
void  main(){
     string s1("Man"); // طرق مختلف تعریف شئ با مقادیر اولیه
     string s2 = "Woman"; //
     string s3;
     cin>>s3;//( خواندن رشته تا رسیدن به بلانک یا زدن دکمه اینتر
     s3 = s1; // عملگر انتساب سربارگذاری شده
     cout <<"s3 = "<< s3 << endl;
     s3 = "Neither " + s1 + " nor "; // + بهم چسباندن رشته ها با عملگر
     s3 += s2; // باز هم عملگر سربارگذاری شده += برای پیوند دو رشته
     cout <<"s3 = "<< s3 << endl;
     s1.to_lower(); // تبدیل حروف الفبایی رشته به حروف کوچک
}
```

مثال: آشنایی بیشتر با نحوه خواندن و نوشتن رشته های استاندارد در C++

```
string full_name, nickname, address;
string salam("Hello, ");
cout <<"Enter your full name: ";
getline(cin, full_name); // این تابع میتواند کاراکتر بلانک را هم بخواند
```

```
cout <<"Your full name is: "<< full_name << endl;
cout <<"Enter your nickname: ";
cin >> nickname; // خواندن شئ رشته کاراکتری
salam += nickname; // عملگر سربارگذاری شده += برای پیوند رشته ها
cout <<salam<< endl;
cout <<"Enter your address on separate lines\n";
cout <<"Terminate with '$'\n";
//تابع زیر میتواند رشته های شامل چند خط را بخواند
getline(cin, address, '$');// کاراکتر پایان ورودی علامت $ تعریف شده
cout <<"Your address is: "<< address << endl;
```

متدهای کلاس string بسیار متنوع هستند و قابلیتهای کم نظیری از جمله جستجو در رشته و یافتن زیر رشته در اختیار برنامه نویس قرار میدهند و برای آگاهی از آنها میتوان به مراجع کامل C++ مراجعه کرد.ضمناً برای مقایسه رشته ها میتوان از عملگرهای سربارگذاری شده < و > و => و ... استفاده نمود.

اشاره گرها (pointers):

میدانیم که متغیرها در حافظه اصلی کامپیوتر نگهداری میشوند و هر متغیر دارای یک **نشانی یا آدرس** در حافظه میباشد.به بیان دقیقتر هر بایت حافظه دارای یک آدرس یا نشانی است. حال **اگر نشانی یک متغیر را در داخل یک متغیر دیگر قرار دهیم متغیر دوم یک اشاره گر به متغیر اول محسوب میگردد**:

حافظه اصلی رایانه

نشانی	محتوا
۰	۱۷
۱	-۱۲۸
۲	5Ah
۳	'C'

حداکثر مقدار نشانی برای یک خانه حافظه بیانگر ظرفیت حافظه رایانه است.

یکی از کاربردهای عملگر& در C++ بدست آوردن آدرس یا نشانی یک متغیر در حافظه است و برای تعریف یک اشاره گر از کاراکتر * استفاده میشود:

```
int m,*mp; // mp یک اشاره گر به متغیرهای صحیح است
mp = &m; //بدست آوردن نشانی یک متغیر و انتساب آن به یک اشاره گر
*mp=4; // انتساب عدد ٤ به متغیر m از طریق اشاره گر آن
cout<<m; // خروجی برابر ٤ خواهد بود
```

روش دیگر برای تعریف اشاره گر:

```
int*  ip;
char* cp;// اشاره گر به متغیرهای کاراکتری
```

عبارت mp*بمعنای مکانی در حافظه است که اشاره گر mp به آن اشاره میکند.

⚠ هشدار: *قبل از استفاده از اشاره گرها یک آدرس معتبر به آنها نسبت دهید و گرنه برنامه شما در زمان اجرا با مشکلات جدی روبرو خواهد شد. همچنین میتوانید مقدار اولیه اشاره گرها را* **NULL** *قرار دهید که بمعنی اشاره گری است که به هیچ کجا اشاره نمیکند.*

نکته مهم: انتساب آدرس متغیرها باید به اشاره گرهای هم نوع خودشان صورت گیرد و گرنه با پیغام خطا مواجه خواهید شد. مثلاً قطعه برنامه زیر تولید خطا میکند:

```
float flovar = 98.6;
int*  pint = &flovar; //خطا: ناسازگاری اشاره گر و آدرس ارائه شده
```

اما یک استثناء در این زمینه وجود دارد و آن **اشاره گرهای نوع void** میباشد:

```
float flovar = 98.6;
//اشاره گرهای نوع void میتوانند آدرس هر نوع متغیری را در خود نگهداری کنند
void* ptrint = &flovar; //ok!
```

رابطه میان آرایه ها و اشاره گرها :

نام هر آرایه ، یک اشاره گر ثابت به اولین عنصر آن آرایه محسوب میشود.برای درک بهتر مطلب به مثال زیر توجه نمایید که در آن از نام آرایه بعنوان یک اشاره گر برای دسترسی به عناصر یک آرایه استفاده شده است:

```
int intarray[5] = { 31, 54, 77, 52, 93 };
for(int j=0; j<5; j++) // چاپ عناصر آرایه
    cout <<*(intarray+j)<< endl;
```

به خاطر داشته باشید که نام یک آرایه یک ***اشاره گر فقط–خواندنی*** *محسوب میشود و حق تغییر دادن آنرا نداریم!*

مثال: دسترسی خانه به خانه به عناصر آرایه با استفاده از یک اشاره گر

```
int intarray[] = { 31, 54, 77, 52, 93 }; //آرایه
int* pi; //اشاره گر به اعداد صحیح
pi = intarray; //آدرس خانه اول آرایه در اشاره گر ریخته میشود
for(int j=0; j<5; j++)
     cout<<*(pi++)<< endl;
```

عملیات افزایش واحد روی اشاره گر باعث میشود هر بار یک خانه در آرایه پیشروی کرده و به خانه بعدی اشاره کند.دقت داشته باشید که عبارت (pi++)* با عبارت (ip++)* کاملاً تفاوت دارد: *(ip++)* که بصورت *ip++ نیز قابل نوشتن است ابتدا یک واحد به اشاره گر اضافه کرده سپس مقداری را که اشاره گر به آن اشاره دارد برمیگرداند.*

مثال: معکوس کردن یک رشته کاراکتری با استفاده از اشاره گرها

```
#include <iostream.h>
#include <cstring.h>
void  main()
{
     string s1;
     cin>>s1;
     char*  pc1 = &s1[0]; //اشاره به اولین کاراکتر رشته
     char*  pc2 = &s1[s1.length()-1]; //اشاره به آخرین کاراکتر رشته
     char   ch;
     while( pc1 < pc2 ){
          //تعویض جای کاراکترها
          ch = *pc1;
          *pc1 = *pc2;
          *pc2 = ch;
          pc1++; pc2--;//پیشروی اشاره گرها
     }
     cout<<s1;//!اکنون رشته ورودی وارونه شده است
}
```

مثال: چاپ اسامی روزهای هفته با استفاده از آرایه ای از اشاره گرها

```
// تعریف آرایه ای از اشاره گر به کاراکتر
char* arrptrs[7] = { "Sunday", "Monday","Tuesday","Wednesday",Thursday",
"Friday", "Saturday" };
for(int j=0; j<7; j++)
     cout << arrptrs[j] << endl;
```

عملگرهای new و delete:

عملگرnew مقداری حافظه از سیستم عامل درخواست نموده و اشاره گری به اولین بایت ناحیه بدست آمده را باز میگرداند:

```
// رزرو کردن تعداد دلخواهی متغیر متوالی از نوع اعداد صحیح بزرگ
int i;
cin>>i;
long*  lp = newlong[i];//اشاره گر به اولین مورد برگردانده میشود
```

از عملگرdelete برای آزاد سازی بخشی از حافظه که قبلاً با عملگرnew درخواست کرده ایم استفاده میشود:

```
delete[] lp;
```

اشاره گر به اشیاء:

اشاره گرها میتوانند به اشیاء یک کلاس هم اشاره کنند:

```
#include <iostream.h>
class rectangle {
private:
     int  length,width;    // اطلاعات حساس در بخش خصوصی ذکرمیشوند
public:
// متدسازنده
     rectangle(int l=10,int w=5)  { length = l; width = w; }
// متدی برای برگرداندن طول مستطیل
     int get_length() {  return length; }
     int get_width(){  return width; } // متدی برای برگرداندن عرض مستطیل
     int area() {  return length*width; } // متد محاسبه مساحت مستطیل
// متد چاپ مقادیرطول وعرض مستطیل
     void print() {  cout<<"Length: "<<length<<" Width:"<<width; }
};

void  main()
{
     rectangle *rp1;// تعریف یک اشاره گر به شئ
     rp1 = new rectangle;//new ایجاد یک شئ با عملگر
    rp1->print();// فراخوانی یکی از متدهای شئ از طریق اشاره گر آن
}
```

عملگر-> برای اشاره به اجزاء یک شئ از طریق اشاره گر آن کاربرد دارد.