برنامه نویسی با C++

استاد :
احمد رستمی

فهرست مطالب

منابع و مراجعی که در تهیه و تنظیم این جزوه از آنها استفاده شده عبارتند از:

۱) چگونه با C++ برنامه بنویسیم(Deitel & Deitel)
۲) برنامه نویسی به زبان C++ (جعفر نژاد قمی)
۳) **turbo C++ 4.5 HELP**

فصل اول – مفاهیم مقدماتی برنامه نویسی

اسامی انگلیسی کاراکتر های روی صفحه کلید:

Ampersand	&	colon	:	dollar sign	$	number sign	#
Vertical line , pipe	|	semi colon	;	Exclamation point(mark)	!	Star ,asterisk	*
Underline, underscore	_	Minus, dash ,hyphen	-	question mark	؟	Slash ,slant	/
greater than	>	comma	,	at sign	@	hat	^
less than	<	quotation	'	Equal	=	percent	%
back slash ,reverse slant	\	double quotation	"	right parentheses	)	left parentheses	(
tilde	~	dot	.	Plus	+	Apostrophe	`
Square brace (bracket)	[	Square brace (bracket)	]	Curly brace (bracket)	{	Curly brace (bracket)	}

مقدمات زبان C++:

۱-زبان C++ زبانی قدرتمند است برای نوشتن برنامه های سیستمی نیز به کار می رود. این زبان یک زبان سطح میانی است، به این معنی که هم دستورات زبان های سطح بالا و هم دستورات زبان های سطح پایین را دارا می باشد. با آن می توان برنامه نویسی پورت های socket programming را انجام داد. همچنین زبانی انعطاف پذیر است برای نوشتن کامپایلرها،ویراستار ها و سیستم عامل ها.

۲-C++ زبانی قابل حمل است به این معنی که می توان برنامه نوشته شده در آن را به راحتی از سیستم عاملی به سیستم عامل دیگر منتقل کرد.

۳-کلمات کلیدی (Reserved Words) زبان C++ کم هستند.

۴-C++ زبانی شی گرا است(objected oriented).

۵-زبان C++ ، case sensitive است.(یعنی حساس به حروف بزرگ و کوچک). کلمات کلیدی با حرف کوچک نوشته شوند.

۶-در هر سطر می توان چند دستور نوشت.
۷-هر دستور می تواند در چند سطر نوشته شود.
۸-آخر هر دستور کاراکتر **;** قرار می گیرد.
۹-comment ها یا جملات توضیحی بین /* و */ یا بعد از // قرار می گیرند.
مانند:

```
//cho khahi ke namat bovad javedan makon name nike bozorgan nahan
```

یا :

```
/* cho khahi ke namat
 bovad javedan makon
name nike bozorgan nahan   */
```

۱۰-هر برنامه از قطعاتی به نام کلاس(class) یا تابع(function) تشکیل شده است.پس زبان **c++** باید حد اقل یک تابع داشته باشد.(تابع **Main**)
۱۱-زبان **c++** زبانی کامپایلری است به این معنی که دستوراتی که نوشته ایم در آخر با زدن کلید **ctrl + f9** کامپایل می شود. در این صورت تمامی خطاهای برنامه لیست می شود و اگر خطا نداشته باشد،برنامه اجرا می شود.

کامپایل زبان c++ :

برنامه های نوشته شده(کد برنامه ها) با پسوند **cpp.** ذخیره می شوند.(در زبان **c** با پسوند **c.** ذخیره می شوند.**)**
فایل های اجرایی برنامه با پسوند **exe.** ذخیره می شوند.
هنگام کد نویسی در برنامه **turbo c++ 4.5** برای کامپایل کردن یا ترجمه کردن کلید **F9** و برای کامپایل و اجرا کلید **ctrl + F9** را می زنیم.
نکته: همیشه قبل از کامپایل و اجرا برنامه را ذخیره کنید.

نوشتن برنامه به زبان c++ :

ساختار کلی یک برنامه در زبان **c++** به شکل زیر است.در ساختار زیر تنها تابع **main** اجباری است و باقی موارد درصورت نیاز اضافه می شوند و ضروری نیستند.

```
#include < فایل های سرآیند >
تعریف ثوابت
تعریف متغیرهای سراسری
اعلان توابع
تعریف توابع
main( )
{ }
```

فایل های سرآیند (کتابخانه ها):

این فایل ها پس از نصب کامپایلر زبان **c++** روی هارد کامپیوتر قرار می گیرند یک برنامه **c++** ممکن است از دستوراتی (توابع) استفاده کند که از پیش نوشته شده اند،این دستورات از پیش نوشته شده در داخل فایل های سرآیند(**header**) قرار دارند.پسوند این فایل ها **h.** است.در یک برنامه ممکن است از صفر و یک و دو و بیشتر از این فایل ها استفاده شود.برای استفاده از فایل های سرآیند در یک برنامه باید به شکل زیر عمل کرد:

#include < فایل های سرآیند **>**

نکته:دستوراتی که قبل از آنها کاراکتر # قرار می گیرد دستورات پیش پردازنده نام دارند،که اینگونه دستورات به **;** نیاز ندارند.

برخی از فایل های سرآیند و توابعی که در آنها قرار دارند:

نام تابع	کاربرد	نام کتابخانه(فایل سرآیند)
clrscr( )	پاک کردن صفحه نمایش	conio.h
cin	گرفتن اطلاعات	iostream.h
cout	چاپ اطلاعات	iostream.h
gotoxy(x,y)	بردن مکان نما به سطر**y** و ستون**x**	iostream.h
rand( )	تولید اعداد تصادفی	stdlib.h
strcmp( )	مقایسه رشته ای	string.h
strcpy( )	کپی رشته	string.h
sqrt( )	جذر	math.h

انواع داده ای:(**Data Types**)

	مثال	اندازه	نوع	متغیر
۱	int a=2;	۲ بایت	اعداد صحیح	int
۲	long int a=45999;	۱۰ بایت	اعداد صحیح	long int
۳	float a=17.35;	۴ بایت	اعداد اعشاری	float
۴	double a=17.35;	۸ بایت	اعداد اعشاری (دقت مضاعف)	double
۵	char c;	۱ بایت	نوع کاراکتری	char

نکته : هر بایت ۸ بیت است و در هر بیت می توان دو مقدار ۰ و یک را ذخیره کرد پس در یک بایت می توان $۲^{۸}$ مقدار را ذخیره کرد. در متغیری از جنس **integer** می توان $۲^{۱۶}$ مقدار یعنی ۶۵۵۳۶ را ذخیره کرد و چون اعداد منفی را هم شامل می شود پس این مقدار نصف می شود.(۳۲۷۶۸)

تعریف ثوابت (**Const**):

اگر یک ثابت تعریف کنیم ،مقدارش در کل برنامه بدون تغییر است. ثوابت فقط یکبار مقدار اولیه می گیرند. از ثابت ها برای خواناتر شدن برنامه و سهولت در تغییر برنامه استفاده می شود.

در **c++** می توانیم به سه شکل زیر ثابت تعریف کنیم:

```
1)# define  name  value
2)const  name=value;
3)const  type name=value;
```

مثال)

```
# define s1 "Ali"
و
const m=3;
و
const float f=12.56;
```

تعریف متغیرها(**Variable**):

نحوه یا **syntax** تعریف متغیرها به صورت زیر است:

نام متغیر نوع متغیر **;**

می توان در حین تعریف متغیرها ، به آنها مقدار دهی اولیه کرد.

```
int a;
float ahmad;
char m,n,p='u';
int b=4,c=b,a=c;
double a=2.5,k=3.002,r;
```

خصوصیت **c++** این است که در هر جای برنامه می توان متغیر تعریف کرد.

طریقه زیر در تعریف و مقدار اولیه دهی به متغیرها غلط است:

int a=b=6;

نام متغیر میتواند حاوی حروف و اعداد و کاراکتر **underline (_)** باشد و بایستی ابتدا با یک حرف یا **underline** آغاز شود. در یک تابع نمی توان دو متغیر همنام بکار برد مگر آنکه در حروف بزرگ و کوچک با هم اختلاف داشته باشند.

تابع **main**:

صورت کلی تابع **main**:

```
void main( )
{
دستورات
}
یا
```

```
int main( )
{
دستورات
return 0;
}
```

دستور **cout** :

این دستور برای چاپ متغیر ها و عبارات به کار می رود. نحوه ی استفاده از این دستور به شکل زیر است:

cout<<value1<<value2<<...;

مثال)

دستور	خروجی
cout<<"Liver Pool";	**Liver Pool**
cout<<a;	مقدار داخل متغیر **a** را چاپ می کند.
cout<<5;	**5**
cout<<"5+6"<<3;	**5+63**
cout<<5+6<<3;	**113**

عبارت **endl** : اگر این عبارت در جلوی دستور **cout** بیاید باعث می شود که مکان نما به خط پایین برود.

عبارت **\n** (**New Line**): اگر این عبارت بعد از **cout** بیاید باعث می شود که مکان نما به خط پایین برود.

نکته: فرق بین دو عبارت بالا در این است که عبارت **endl** بیرون " " قرار می گیرد ولی عبارت **\n** داخل " " قرار می گیرد.

عبارت **\t** (**Tab**): این عبارت باعث می شود که مکان نما ۸ کاراکتر به جلو حرکت کند.

Example: cout<<23<<"protest"<<endl<<45<<"\n\tForce majeure";

خروجی:

23protest
45
Force majeure

دستور **cin** :

برای گرفتن مقدار از ورودی می باشد و شکل دستور بدین صورت است:

```
cin>> variable1>>variable2>>...;
```

نکات:

۱) بین هر متغیر علامت>> قرار می گیرد.

۲)پس از وارد کردن هر مقدار با کاراکتر **space** یا **enter** مقدار بعدی وارد می شود.

۳)هر تعداد متغیر در دستور **cin** آمده باشد، به همان تعداد بایستی مقدار وارد کرد.

مثال)

عملکرد	دستور
مقداری از ورودی گرفته و آن را در متغیر **a** قرار می دهد	cin>>a;
سه بار از ورودی مقدار گرفته و به ترتیب در متغیرهای **a,b,c** قرار می دهد	cin>>a>>b>>c;

به برنامه های زیر توجه کنید:

```
#include <iostream.h>
void main()
{
int a,b;
char d;
cin>>a>>b;
cout<<endl<<a+b;
cin>>d;
cout<<"\n"<<"d is: "<<d;
}
```

برنامه ای بنویسید که عبارت " this is first program " را چاپ کند.

```
#include <iostream.h>
void main()
{
cout<<"this is first program";
}
```

عملگرهای محاسباتی:

فرض کنید **x=5 , z=2**

	عملگر	عمل	مثال	خروجی
۱	+	جمع	y=x+z	y=7
۲	-	تفریق	y=x-z	y=3
۳	*	ضرب	y=x*z	y=10
۴	/	تقسیم	y=x/z	y=2.5
۵	%	باقیمانده تقسیم	y=x%z	y=1
۶	++	افزایش یک واحد	x++;	x=6
۷	--	کاهش یک واحد	z--;	z=1

عملگرهای مقایسه ای:

فرض کنید **x=2 , y=3**

	علامت عملگر	نام عملگر	مثال
۱	>	بزرگتر	y>x
۲	>=	بزرگتر یا مساوی	y>=x
۳	<	کوچکتر	x<y
۴	<=	کوچکتر یا مساوی	x<=y
۵	!=	نامساوی	x!=y
۶	==	مساوی	p=3; , q=3; p==q

عملگرهای منطقی:

	عملکرد	نام عملگر	علامت عملگر
۱	اگرطرفین درست باشد،حاصل درست است	**and**	&&
۲	اگریکی ازطرفین یا هردو طرف درست باشد،حاصل درست است	**or**	||
۳	اگر عبارت جلوی **!** غلط باشد، حاصل درست است	**not**	!

حاصل عبارتهای زیر را بدست آورید.

!(3>2) || 3	5>3 && 4>8	5>3 || 4>8

تذکرمهم: در زبان c++ عدد صفر False و هر عددی غیر از صفر true محسوب می شود(حتی اعداد منفی).

عملگرهای خاص:

۱) = (**assignment**)

مقدار سمت راستش را داخل متغیر سمت چپ می ریزد.

مثال)**a=4 , b=c**

۲) **?:** (**question mark colon**)

y= condition ? true expression : false expression;

شرط را بررسی می کند،اگر برقرار بود عبارت قبل از : اجرا می شود و اگر شرط برقرار نبود عبارت بعد از : اجرا می شود.

مثال)

```
p=4;
q=15;
y=(p>4/2)?3*p:q;
```

جواب:**y=12**

مثال)

```
x=8;
m=6
y=x*2<m+4?4*m:8*m;
```

جواب:**y=48**

شیوه نوشتن برخی عملگر ها:

```
i+=1; ==> i=i+1;
i-=1;  ==> i=i-1;
i*=5;  ==> i=i*5;
i/=6;  ==> i=i/6;
i%=2;  ==> i=i%2;
i+=x;  ==> i=i+x;
x+=x;  ==> x=x+x;
```

خروجی قطعه برنامه های زیر را مشخص کنید.

```
int m;
m=8;
m++;   //m=9
m+=3;  //m=m+3 => m=12
m%=2;  //m=m%2 => m=0
cout<<m;
```

```
int m;
m=7;
m*=3;  //m=7*3=21
cout<<m;
```

فصل دوم – حلقه ها (LOOPS)

انواع حلقه ها

۱)شمارشی:یک شمارنده دارد که مقدارش تغییر می کند تا به مقدار نهایی برسد سپس از حلقه خارج می شویم(مانند حلقه **for**).

۲)غیر شمارشی:حلقه ادامه می یابد تا هنگامی که شرط برقرار باشد،بایستی در بدنه حلقه دستوراتی باشد که شرط را نقض کند(حلقه **while**).

حلقه **FOR**:

نحوه ی نوشتن:

```
for(مقدار دهی اولیه به اندیس حلقه ; مقدار نهایی(شرط) ; گام حلقه )
{
دستورات یا دستور
}
```

نکات:

۱.در حلقه **for** می توان از ۰ و ۱ و بیشتر اندیس حلقه استفاده کرد.(اندیس حلقه متغیری است که در طول اجرای حلقه مقدارش بررسی می شود).

۲.بین پارامتر های دستور **for** کاراکتر **;** قرار می گیرد.(یک **for** حتما بایستی **for(; ;)** داشته باشد.)

۳.حلقه تا هنگامی ادامه می یابد که شرط ادامه داشته باشد، به محض اینکه شرط نقض شد از حلقه خارج می شویم.

۴.می توان ۰و۱و۲ یا بیشترشرط داشت.

۵.گام حلقه می تواند افزایشی،کاهشی،تقسیم،ضرب و غیره باشد.

۶.بعد از دستور **for** کاراکتر **;** لازم نیست،اگر بعد از **for** کاراکتر **;** بیاید دستورات بعد از حلقه **for** جزء حلقه قرار نمی گیرد.

۷.اگر بعد از دستور **for** تنها یک دستور بیاید نیاز به **}** و **{** نیست اما اگر بخواهیم بیش از یک دستور در حلقه **for** قرار دهیم باید آن دستورات بین **{** و **}** قرار بگیرند.

۸.**for(; ;)** به منزله یک حلقه بی نهایت است(یعنی اگر دستوراتی در این حلقه قرار گیرند بی نهایت بار اجرا می شوند.

.

مثال:دستورات زیر حلقه چند بار اجرا می شوند؟

```
1)for(i=3;i<8;i++)
cout<<"real";
```

جواب:۵ بار اجرا می شود.

```
2)for(m=7;m>=2;m--)
cout<<"*";
```

جواب:۶ بار اجرا می شود.

عملکرد حلقه **for**:

قبل از اینکه دستورات زیر حلقه **for** اجرا شوند شرط بررسی می شود.یعنی اگر ما اندیس حلقه را مقدار دهی کنیم فورا بعد از آن شرط بررسی می شود.

نکته:پس از اینکه یک بار دستورات حلقه اجرا شدند دستوراتی که که گام حلقه را مشخص می کند اجرا شده و سپس بلافاصله شرط بررسی می شود،اگر برقرار بود دستورات داخل حلقه اجرا می شود و اگر برقرار نبود از حلقه خارج شده و به یک دستور بعد از دستورات حلقه می رود.

```
3)for(a=-2;a<=4;a=a*2)
cout<<"*"<<endl;
```

جواب:بی نهایت بار اجرا می شود.

```
4)for(a=-2;a<=4;a=a+2)
cout<<"*"<<endl;
```

جواب:۴بار اجرا می شود.

```
5)for(a=-2;a==4;a++)
cout<<"*"<<endl;
```

جواب:اجرا نمی شود.

```
6)for(i=3,j=15 ; i<j ; i++,j--)
cout<<"*"<<endl;
```

جواب:۶ بار اجرا می شود.

```
7)for(i=1,j=15; i>j ; i=i+3,j--)
cout<<"*";
```

جواب:اجرا نمی شود.

```
8)for(i=1;i<5||i<10;i++)
cout<<"*";
```

جواب:۹ بار اجرا می شود.

حلقه های زیر چند بار اجرا می شود؟

```
 1)for(i=3 ; (i==3)&&(i>3) ; i++)
 2)for(i=1;;i++)
3)for(i=1,j=32;i<j*2;i=i*2,j=j/2)
4)for(i=2;!(i==2);i--)
```

مثال: برنامه ای بنویسید که مساحت و محیط دایره را حساب کند.

```
//in the name of Allah
#include <iostream.h>
#define pi 3.14
void main()
{
int r,m,a;
cin>>r;
m=pi*r*r;
a=2*pi*r;
cout<<"perimeter="<<m<<endl<<"area="<<a;
}
```

نکته : جنس متغیر های m و p از نوع integer است. مقدار اعشاری بوجود آمده بریده می شود تا بتواند در متغیر قرار گیرد. به مثال زیر توجه کنید:

```
int a;
float b=2.5;
a=b*3;
cout<<b<<"   "<< b*3"<<"   "<<a;
```

output: 2.5 7.5 7

عملگرهای پیش‌افزایشی و پیش‌کاهشی:

۱	ابتدا از مقدار متغیر b استفاده می کند، سپس یک واحد به آن اضافه می کند.	b++;
۲	ابتدا به b یک واحد اضافه می کند، سپس از مقدار آن استفاده می کند.	++b;
۳	ابتدا از مقدار متغیر b استفاده می کند، سپس یک واحد از آن کم می کند.	b--;
۴	ابتدا از متغیر b یک واحد کم می کند، سپس از مقدار آن استفاده می کند.	--b;

نکته: عملگرهای پیش افزایشی و پیش کاهشی بایستی حتما روی نام متغیر اعمال شوند.

دستور زیر اشتباه است:

```
++(x+2);
```

شکل درست آن بصورت زیر است:

```
x=x+2;
x++;
```

برای درک بهتر عملگرهای پیش افزایشی و پیش کاهشی به مثال‌های زیر توجه کنید:

۱) حلقه‌ی زیر چند بار اجرا می شود؟

```
c=2;
for(a=1;a<c++;a*=2)
cout<<"*";
```

جواب:دو بار اجرا می شود.

۲) حلقه‌ی زیر چند بار اجرا می شود؟

```
c=2;
for(a=1;a<++c;a*=2)
cout<<"*";
```

جواب:سه بار اجرا می شود.

۳) خروجی تکه برنامه ی روبرو چیست؟

```
a=2;
cout<<a++;
```

جواب:خروجی برنامه ۲ است.

اولویت عملگرها در حالت کلی:

عملگر	اولویت
()	۱
++ -- - !	۲
* / %	۳
+ -	۴
<= < >= >	۵
!= ==	۶
&&	۷
||	۸
?:	۹
= += -= /= *= %=	۱۰

نکته : در هر سطر ، اولویتها با هم یکسان است. در صورت داشتن اولویت مساوی در یک عبارت ، اولویت با عملگر سمت چپ تر است.

مثال) با F و T درست یا نادرست بودن مثال های زیر را مشخص کنید:

3==4*8+4%2 !(4+12/3%2+2<4+3*2)

حلقه while:

یک حلقه‌ی غیرشمارشی است.این حلقه تا هنگامی که شرط داخل آن برقرار باشد اجرا می شود و هنگامی که شرط نقض شد (false شد) از حلقه خارج می شود.

نحوه‌ی نوشتن:

```
while(شرط)
{
دستورات
}
```

نکات:

۱. اگر بعد از دستور while یک دستور بیاید نیازی به { و } نیست.

۲. در داخل حلقه while نیاز است تا دستوری باشد که شرط را تغییر دهد، در غیر اینصورت حلقه‌ی بی نهایت بوجود می‌آید.

نمونه هایی از حلقه های بی نهایت:

```
while(-5)
while(1)
while(3<4)
```

نمونه هایی از حلقه هایی که اصلا اجرا نمی شوند:

```
while(0)
while(4<3)
while('H'<'A')
```

تذکر: در زبان c++ هر عددی غیر از صفر true محسوب می شود.

مثال) حلقه‌ی زیر چند بار اجرا می‌شود؟ (ستاره چند بار چاپ می شود؟)

```
i=2;
while(i<65)
{
cout<<"*\n";
i*=2;
}
```

جواب: ۶ بار اجرا می شود.

حلقه **do while**:

یک حلقه‌ی غیرشمارشی است.این حلقه تا هنگامی که شرط داخل آن برقرار باشد اجرا می شود و هنگامی که شرط نقض شد از حلقه خارج می شود، تنها فرق آن با حلقه‌ی while این است که: در حلقه‌ی while اگر در ابتدا شرط برقرار نباشد،حلقه اصلا اجرا نمی شود اما در do while حد اقل یکبار حلقه اجرا می شود، سپس شرط بررسی می شود.

نحوه ی نوشتن:

```
do
{
دستورات
}while(شرط);
```

مشخص کنید هر کدام از حلقه ها چند بار اجرا می‌شود؟

مثال)

```
i=5;
do
{
i*=(3+2);
cout<<"future ";
}
while(i<76);
```

جواب: دوبار اجرا می شود.

مثال)

```
i=5;
while(i>76)
{
cout<<"feature";
}
```

جواب: اصلا اجرا نمی شود.

مثال)

```
i=5;
do
cout<<"further";
while(i>76);
```

جواب: یک بار اجرا می شود.

به تفاوت بین مثال ها توجه کنید:

مثال)

```
for2=1;
while(for2++<3)
cout<<"seed\n";
```

جواب: ۲ بار اجرا می شود.

مثال)

```
for2=1;
while(++for2<3)
cout<<"seed\n";
```

جواب: ۱ بار اجرا می شود.

دستور **break**:

از این دستور برای خروج ناگهانی از حلقه استفاده می شود. به این ترتیب که هرگاه به دستور break برسیم، کلیه‌ی دستورات داخل حلقه نادیده گرفته می‌شود و کلا از حلقه خارج می‌شود.
اگر چند حلقه‌ی تودرتو داشته باشیم، این دستور باعث خروج از داخلی ترین حلقه می‌شود.

دستور **continue**:

این دستور در هر جای حلقه باشد باعث می‌شود که از اجرای دستورات بعدی صرف نظر شده و شمارنده‌ی حلقه(گام حرکت) تغییر یابد و حلقه ازسر گرفته شود. (به ابتدای حلقه برمی گردد)

به مثال های زیر توجه کنید:
مثال) در حلقه‌ی زیر ستاره چند بار تکرار می‌شود؟

```
for(i=1;i<10;i++)
{
cout<<"*"<<endl;
if(i%5==0)
break;
}
```

جواب: ۵ بار
مثال)خروجی حلقه‌ی زیر چیست؟

```
for(i=1;i<10;i++)
{
if(i%4==0)
continue;
cout<<i<<"   ";
}
```

خروجی: 1 2 3 5 6 7 9

مسائل نمونه

برنامه ای بنویسید که جدول ضرب 10*10 را چاپ کند.

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
int i,j;
for(i=1;i<=10;i++)
  {
   for(j=1;j<=10;j++)
    cout<<setw(4)<<i*j;
    cout<<endl;
  }
}
```

برنامه ای بنویسید که خروجی زیر را چاپ کند.

```
*
**
***
****
*****
```

```
//bename khoda.
#include<iostream.h>
void main()
{
int i,j;
for(i=1;i<=5;i++)
   {
     for(j=1;j<=i;j++)
     cout<<"*";
     cout<<endl;
   }
}
```

برنامه ای بنویسید که خروجی زیر را چاپ کند.

```
*****
****
***
**
*
```

```
#include<iostream.h>
void main()
{
int i,j;
for(i=5;i>=1;i--)
  {
   for(j=i;j>=1;j--)
   cout<<"*";
   cout<<endl;
  }
}
```

برنامه ای بنویسید که خروجی زیر را چاپ کند.

```
1
12
123
1234
12345
```

```
//bename khoda.
#include<iostream.h>
void main()
{
int i,j;
for(i=1;i<=5;i++)
  {
    for(j=1;j<=i;j++)
    cout<<j;
    cout<<endl;  }}
```

برنامه ای بنویسید که شکل زیر را چاپ کند.

```
12345
1234
123
12
1
```

```
//bename khoda.
#include<iostream.h>
void main()
{
int i,j;
for(i=5;i>=1;i--)
 {
  for(j=1;j<=i;j++)
  cout<<j;
  cout<<endl;
 }
}
```

برنامه ای بنویسید که عددی بعنوان شماره‌ی جمله‌ی فیبوناچی گرفته، و از ابتدا تا آن عدد جملات فیبوناچی را چاپ کند.

```
#include <iostream.h>
main()
 {
int f1,f2,fib,i,n;
f1=1;
f2=1;
cin>>n;
cout<<"fib1: 1\n"<<"fib2: 1\n";
for (i=3;i<=n;i++)
{
fib=f1+f2;
```

```
f1=f2;
f2=fib;
cout<<"fib"<<i<<": "<<fib<<"\n";
}
}
```

برنامه ای بنویسید که عددی را گرفته، و سری فیبوناچی ما قبل آن عدد را چاپ کند.(جملات کوچکتر از آن عدد را)

```
#include <iostream.h>
main()
 {
int f1,f2,fib,i=3,n;
f1=1;
f2=1;
fib=1;
cin>>n;
cout<<"fib1: 1\n"<<"fib2: 1\n";
fib=f1+f2;
while (fib<=n)
{
cout<<"fib"<<i<<": "<<fib<<"\n";
f1=f2;
f2=fib;
fib=f1+f2;
i++;
}
}
```

برنامه‌ای بنویسید که تا وقتی عدد صفر وارد نشده، عدد از ورودی بگیرد. سپس جمع آنها را حساب کند.

```
#include <iostream.h>
main()
 { long int s=0,a;
cout<<"enter your numbers : "<<endl;
cin>>a;
while (a!=0){
s+=a;
cin>>a;
}
cout<<"Sum is "<<s;
}
```

برنامه ای بنویسید که یک کاراکتر را بگیرد و کد اسکی‌اش را چاپ کند.

```
#include<iostream.h>
void main()
{
int i;
char h;
cin>>h;
i=h;
cout<<i;
}
```

برنامه‌ای بنویسید که کلیه‌ی کاراکترها به همراه کد اسکی آنها را چاپ کند.

```
#include<iostream.h>
void main()
{
int i;
char h;
for(i=1;i<=122;i++)
  { h=i;
    cout<<i<<"="<<h<<endl;
}
}
```

فصل سوم – دستورات شرطی

دستور شرطی if:

نحوه نوشتن:

```
if (شرط)
{
دستورات قسمت اول;
}
else
{
دستورات قسمت دوم;
}
```

توجه داشته باشید ساختار بالا یک ساختار کلی است و در حالات مختلف می‌تواند شکل این ساختار تغییر کند.
می‌توان دستور شرطی if را در حالات زیر هم بکار برد:
الف)

```
if (شرط)
دستور;
```

ب)

```
if (شرط)
{
دستورات
}
```

پ)

```
if (شرط)
دستور;
else
{
دستورات;
}
```

نکات:
۱.اگر بعد از if تنها یک دستور بیاید نیاز به } و { نیست.
۲.اگر شرط برقرار باشد دستورات قسمت اول اجرا می‌شود و اگر برقرار نباشد دستورات قسمت دوم اجرا می‌شود
۳. شرط بایستی حتما داخل پرانتز باشد.

مثال : برنامه‌ای بنویسید که سه عدد را گرفته و بزرگترین آنها را چاپ کند.(تنها با استفاده از دو if)

```
#include <iostream.h>
main() {
 int a,b,c,max;
cout<<" Enter your numbers ";
cin>>a>>b>>c;
max=a;
if (b>max)
max=b;
if (c>max)
max=c;
cout<<" The max is " <<max<<endl;
 }
```

برنامه ای بنویسید که کلیه‌ی اعداد چهار رقمی را که از دو طرف به یک شکل خوانده می شود را چاپ کند.

```
#include<iostream.h>
void main()
{
int i,a,b,c,d;
for(i=1000;i<=9999;i++)
{
a=i%10;  //yekan
b=(i/10)%10;  //dahgan
c=(i/100)%10; //sadgan
d=i/1000;   //hezargan
if((a==d) && (b==c))
cout<<i<<endl;
}
}
```

چنانچه بخواهیم از چند شرط متوالی استفاده کنیم می توانیم از else ها و if های متوالی استفاده کنیم.

مثال:

```
if(a==1)
cout<<"A";
else if(a==2)
cout<<"B";
else if(a==3)
cout<<"c";
else
cout<<"D";
```

دستور شرطی **switch-case**:

از این دستور برای چک کردن مقادیر به طوری که هیچ یک از مقادیر با هم مساوی نباشند استفاده می‌شود.این دستور می‌تواند جایگزین else-if های پی در پی شود.

نحوه‌ی نوشتن:

```
switch (متغیر یا عبارت)
{
case ۱مقدار: دستورات  ; break;
case ۲مقدار: دستورات  ; break;
.
.
.
case مقدارآخر: دستورات  ; break;
 [default : دستورات;]
}
```

توضیح: ابتدا مقدار یا عبارت جلوی switch محاسبه می‌شود، سپس آن مقدار با تمام مقادیر جلوی case ها مقایسه می‌شود. با هرکدام که برابر بود دستورات جلوی آن case اجرا می‌شود و بقیه case ها در نظر گرفته نمی‌شود.

اگر با هیچ یک از مقادیر جلوی case ها برابر نبود، دستورات default اجرا می‌شود.

تذکر: اگر از break استفاده نکنیم مقدار case ها با هم or می‌شوند.

تذکر: می‌توان در جلوی switch عبارت محاسباتی نیز به کار برد. مانند: switch(a*2%3)

مثال) برنامه‌ای بنویسید که کاراکتری را که معرف نمره است، از ورودی بگیرد و بر حسب جدول زیر خروجی را چاپ کند.

a	۲۰
b	۱۹
c	۱۸
d	زیر ۱۸

```
#include<iostream.h>
void main()
{
char n;
cin>>n;
switch(n)
{
case 'a' : cout<<20; break;
case 'b' : cout<<19; break;
case 'c' : cout<<18; break;
case 'd' : cout<<"under 18"; break;
default : cout<<"Invalid input";
}
}
```

مسائل نمونه

برنامه‌ای که یک عدد را از ورودی بگیرد و مغلوب آن را چاپ کند.

```
#include<iostream.h>
void main()
{
int c,a;
cin>>a;
c=0;
while(a!=0)
{
c=c*10+(a%10);
a=a/10;
}
cout<<c;
}
```

برنامه‌ای بنویسید که عدد n را گرفته و n جمله سری زیر را چاپ کند.

1,-2,4,-7,11,-16,...

```
#include<iostream.h>
void main()
{
int a,i,seri=1,no,n;
cin>>n;
cout<<seri<<",";
i=1; no=1;
for(a=1;a<=n;a++)
{
i=i*(-1);
no=no+a;
seri=no*i;
cout<<seri<<",";
}
}
```

برنامه‌ای بنویسید که عدد n را از ورودی بگیرد سپس کوچکترین عددی که مجموع ارقامش مساوی با n باشد را چاپ کند.

```
#include<iostream.h>
void main()
{
int i,j,sum,n;
cout<<"Enter n:";
cin>>n;
for(i=0;i<=32767;i++)
{
j=i;
sum=0;
while(j>0)
{
sum=sum+j%10;
j/=10; }
if (sum==n)
{cout<<i;
break; }
}
}
```

برنامه‌ای بنویسید که عدد n را از ورودی گرفته سپس کلیه‌ی اعدادی که از n کوچکتر و ارقام آنها فقط ۲ یا ۵ هستند را چاپ نماید.

```
#include<iostream.h>
void main()
{
int i,j,n,m,a;
cin>>n;
for(i=1;i<=n;i++)
{ j=i;
while(j>0)
{ a=j%10;
if((a!=2)&&(a!=5))
break;
j=j/10;
}
if(j==0)
cout<<i<<" ";
}
}
```

برنامه‌ای بنویسید که مثلث زیر را چاپ کند.

```
   11
  1221
 123321
12344321
```

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
int k,j,i;
for(i=1;i<10;i++)
{cout<<setw(10-i);
for(j=1;j<=i;j++)
cout<<j;
for(k=i;k>=1;k--)
cout<<k;
cout<<endl;
}
}
```

برنامه‌ای بنویسید که ۲ عدد را از ورودی بگیرد سپس ب‌م‌م و ک‌م‌م آنها را چاپ کند.

```
#include<iostream.h>
void main()
{
int m,n,p,f,t;
cin>>m>>n;
f=m;
p=n;
do{
t=m%n;
m=n;
n=t;
} while(t!=0);
cout<<"B.M.M="<<m<<endl;
cout<<"K.M.M="<<(p*f)/m;
}
```

فصل چهارم – آرایه ها (ARRAYS)

فرض کنید می‌خواهیم برنامه‌ای بنویسیم که ۱۰۰ عدد را گرفته و ذخیره کند، و عملیاتی روی آنها انجام دهد. برای این منظور می‌توانیم ۱۰۰ متغیر تعریف کنیم. در این مواقع از آرایه ها استفاده می‌کنیم.
به تعدادی عناصر پشت سر هم از حافظه تحت یک نام واحد و یک نوع واحد و اندیس مجزا آرایه گفته می‌شود.
شکل تعریف آرایه ها به صورت زیر است:

```
نوع نام[طول] ;
int  a[10];
float [100];
```

نکته مهم : اندیس(index) آرایه در زبان c++ از صفر شروع می‌شود.

int m[8];

m	4	7	3	9	10	8	7	2
	m(0)	m(1)	m(2)	m(3)	m(4)	m(5)	m(6)	m(7)

دستور فوق آرایه‌ای به طول ۸ با نام m تعریف کرده است که هر خانه آن دارای نوع integer است. و هر خانه آن می‌تواند مقادیری از 32768- تا 32767 را بگیرد. این مقادیر در حافظه پشت سر هم قرار می‌گیرند.
برای گرفتن آرایه از ورودی و همچنین چاپ آن در خروجی نیاز به حلقه داریم.
مثال) برنامه‌ای بنویسید که آرایه‌ای به طول ۶ از ورودی بگیرد و آن را به طور معکوس چاپ کند.

```
#include<iostream.h>
void main()
{ int i,parsa[6];
for(i=0;i<=5;i++)
cin>>parsa[i];
for (i=5;i>=0;i--)
cout<<parsa[i];
}
```

مثال) برنامه‌ای بنویسید که آرایه‌ای به طول ۱۰۰ را از ورودی بگیرد سپس عناصر مضرب ۳ آن را چاپ کند.

```
#include<iostream.h>
void main()
{ int i,mahdi[100];
for(i=0;i<100;i++)
cin>>mahdi[i];
for(i=0;i<100;i+=3)
cout<<mahdi[i];
}
```

مثال) برنامه‌ای بنویسید که یک آرایه به طول ۱۰۰ در نظر گرفته و عناصری از آن که مقدارشان با اندیس‌شان برابر است را چاپ کند.

```
#include<iostream.h>
void main()
{ int i,mehdi[100];
for(i=0;i<=99;i++)
cin>>mehdi[i];
for(i=0;i<=99;i++)
if(mehdi[i]==i)
cout<<mehdi[i]<<" ";
}
```

مثال) برنامه‌ای بنویسید که آرایه‌ای به طول ۱۰ از ورودی گرفته سپس مجموع عناصر آن و میانگین آنها را چاپ کند. این برنامه را بدون آرایه هم بنویسید.

```
#include<iostream.h>
void main()
{ int i,sum=0,a[10];
for(i=0;i<=9;i++)
{ cin>>a[i];
sum=sum+a[i]; }
cout<<"sum:"<<sum<<endl;
cout<<"average:"<<sum/10; }
```

بدون آرایه:

```
#include<iostream.h>
void main()
{ int i,sum=0,n;
for(i=1;i<=10;i++)
{ cin>>n;
sum=sum+n; }
cout<<"sum:"<<sum<<endl;
cout<<"average:"<<sum/10; }
```

بجز اعمال گرفتن و چاپ کردن دو عمل مهم دیگر نیز بر روی آرایه ها انجام می‌شوند.این عملیات عبارتند از:

۱. جستجو(search)

۲. مرتب سازی(sort)

جستجوی خطی(linear search):

در این جستجو عناصر آرایه ها از اول تا آخر چک می‌شوند و اگر با عنصر داده شده برابر بودند جستجو خاتمه می یابد.

الگوریتم جستجوی خطی)

```
#include<iostream.h>
#include<stdlib.h>
#define n 100
void main()
{ int i,x,a[n];
for(i=0;i<=n-1;i++)
cin>>a[i];
cout<<"enter x;";
cin>>x;
for(i=0;i<=n-1;i++)
if(a[i]==x)
{ cout<<"found";
exit(0); }
cout<<"not found"; }
```

بهترین حالت ۱ بار مقایسه می‌شود. بدترین حالت n بار مقایسه می‌شود.(یا پیدا نشود یا آخرین عنصر آرایه باشد)

حالت متوسط n/2 مقایسه می‌شود. n تعداد عناصر آرایه است.

جستجوی دودویی(binary search):

در این نوع جستجو بایستی آرایه از قبل مرتب باشد. در این جستجو هربار x(عدد مورد نظر) با عنصر وسط مقایسه می‌شود. اگر از آن عنصر کوچکتر بود x با عناصر کوچکتر از عنصر وسط مقایسه می‌شود و اگر x بزرگتر از عنصر وسط بود x با عناصر بزرگتر از عنصر وسط مقایسه می‌شود. و اگر برابر بود عنصر پیدا شده است.

در هر مرحله آرایه نصف می‌شود.

الگوریتم جستجوی دودویی)

در این مثال فرض کرده‌ایم آرایه a از قبل به صورت زیر دریافت شده و x=25 است.

-۶	۵	۹	۱۳	۲۰	۲۱	۲۵	۲۸

```
void b_search(int  a[] , int n)
{
int flag=0;
int i=0;
int j=n-1;
int mid=(i+j)/2;
while((i<j) && !(flag))
{
 if(x==a[mid])
   { cout<<"found";
     flag=1; }
else if(x<a[mid])
     j=mid-1;
else
     i=mid+1;
mid=(i+j)/2;
```

```
 }


if(flag==0)
cout<<"not found";
}
```

بهترین حالت ۱ بار مقایسه. بدترین حالت log (n) بار مقایسه.

مرتب سازی:

چندین الگوریتم برای مرتب سازی وجود دارد که عناصر آرایه را به صورت صعودی یا نزولی مرتب می‌کنند. هر الگوریتم ویژگی های خود را دارد و مهمترین ویژگی آنها سرعت‌شان است. تجربه نشان داده است که الگوریتم مرتب سازی سریع (quick sort) سریعترین روش مرتب سازی است. اما اینجا دو الگوریتم مرتب سازی دیگر را بررسی می‌کنیم.

۱. الگوریتم مرتب سازی تعویضی:

```
#include<iostream.h>
#define n 100
void main()
{ int i,j,a[n],temp;
for(i=0;i<=n-1;i++)
cin>>a[i];
for(i=0;i<n-1;i++)
 for(j=i+1;j<n;j++)
  if(a[i]>a[j])
  { temp=a[i];
    a[i]=a[j];
    a[j]=temp; }
 }
```

۲. الگوریتم مرتب سازی حبابی:

```
#include<iostream.h>
#define n 100
main()
{ int j,i,a[n],temp;
for(i=0;i<=n-1;i++)
cin>>a[i];
for(i=0;i<=n-1;i++)
 for(j=0;j<=n-i;j++)
   if(a[j]>a[j+1])
     { temp=a[j];
      a[j]=a[j+1];
      a[j+1]=temp; }
}
```

آرایه‌های دو بعدی:

نحوه‌ی تعریف آرایه دو بعدی به شکل زیر است:

```
نوع  نام آرایه[بعد اول] [بعد دوم] ;
float a[10][20];
int  a[5][5];
```

نمایش دو بعدی آرایه تعریف شده در مثال قبل به صورت زیر است:

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]
a[4][0]	a[4][1]	a[4][2]	a[4][3]	a[4][4]

دستور زیر یک آرایه دو بعدی را از ورودی می‌گیرد.

```
for(i=0;i<=n-1;i++)
 for(j=0;j<=m-1;j++)
    cin>>a[i][j];
```

نکات:

۱. برای گرفتن این نوع آرایه‌ها از ورودی همیشه به دو حلقه نیاز است.

۲. برای چاپ اینگونه آرایه‌ها نیز به دو حلقه نیاز است و باید طوری دستور آن را بنویسیم که خروجی به صورت یک ماتریس چاپ شود. مانند دستور زیر:

```
for(i=0;i<n-1;i++)
{
for(j=0;j<=m-1;j++)
   cout<<a[i][j];
   cout<<endl;
 }
```

مثال) برنامه‌ای بنویسید که آرایه‌ای دو بعدی به ابعاد ۴ در ۴ بگیرد سپس آن را چاپ کند. همچنین عناصر قطر اصلی آن را چاپ کند.

```
#include<iostream.h>
void main()
{ int i,j,a[4][4];
for(i=0;i<4;i++)
 for(j=0;j<4;j++)
cin>>a[i][j];
for(i=0;i<4;i++)
{ for(j=0;j<4;j++)
cout<<a[i][j]<<" ";
cout<<endl; }
for(i=0;i<4;i++)
cout<<a[i][i]<<" "; }
```

مثال) برنامه‌ای بنویسید که جدول ضرب ۱۰در۱۰ را در یک آرایه دو بعدی ذخیره کند و فقط عناصر ستون پنجم آن را چاپ کند.

```
#include<iostream.h>
void main()
{ int a[10][10];
for(int i=0;i<=9;i++)
{ for(int j=0;j<=9;j++)
a[i][j]=(i+1)*(j+1); }
for(i=0;i<=9;i++)
cout<<a[i][5]<<endl;
}
```

مثال) برنامه‌ای بنویسید که یک آرایه ۳ در ۴ را از ورودی بگیرد سپس کوچکترین عنصر(min) و بزرگترین عنصر(max) آن را چاپ کند.

```
#include<iostream.h>
main()
{ int a[3][4],i,j,min,max;
for(i=0;i<=2;i++)
  for(j=0;j<=3;j++)
     cin>>a[i][j];
max=min=a[0][0];
for(i=0;i<=2;i++)
 for(j=0;j<=3;j++)
 { if(a[i][j]>max)
     max=a[i][j];
   if(a[i][j]<min)
     min=a[i][j];
 }
cout<<"max="<<max<<" min="<<min;
}
```

آرایه های با ابعاد بالاتر نیز در C++ قابل تعریف است.

فصل پنجم – توابع (functions)

هر برنامه‌ی c++ از تعدادی تابع تشکیل شده است. یکی از آنها اجباری است به نام main() اما بقیه‌ی آنها توسط خود برنامه نویس نوشته می شوند و اختیاری هستند.

مزایای استفاده از تابع(زیر برنامه):

۱. امکان کار گروهی فراهم می‌شود: اگر از زیر برنامه استفاده نکنیم یک نفر بایستی تمام آنرا بنویسد اما می‌توان برنامه را به چند زیربرنامه تقسیم کرد و هر برنامه را گروهی به عهده بگیرد.

۲. امکان خطایابی برنامه ساده می‌شود: چون قسمت‌های برنامه از هم مجزا شده اند می‌توان فهمید که ایراد کار از کجاست.

۳.حجم کدنویسی کمتر می‌شود: زیرا می‌توان زیر برنامه را نوشت و چندین بار از آن استفاده کرد.

۴. خوانایی برنامه افزایش می‌یابد.

نکته : هر برنامه می تواند از تعدادی زیر برنامه ای(procedure) تشکیل شود. در C++ به زیر برنامه ، تابع گفته می شود.

توابع دو گونه اند:

۱. توابعی که در خود زبان c++ تعریف شده اند و ما می‌توانیم از آنها به کرات استفاده کنیم. برای استفاده از این گونه توابع بایستی کتابخانه ای که این توابع در آن قرار دارند در ابتدای برنامه معرفی شود.

نمونه هایی از این توابع:

نام تابع	کاربرد	نام کتابخانه
clrscr( )	پاک کردن صفحه نمایش	conio.h
Cin	گرفتن اطلاعات	iostream.h
Cout	چاپ اطلاعات	iostream.h
gotoxy(x,y)	بردن مکان نما به سطر**y** و ستون**x**	iostream.h
rand( )	تولید اعداد تصادفی	stdlib.h
strcmp( )	مقایسه رشته ای	string.h
strcpy( )	کپی رشته	string.h
getch()	خواندن یک کاراکتر از ورودی	conio.h
pow(x,y)	**x** را به توان **y** می‌رساند	math.h
fabs(x)	قدر مطلق **x** را حساب می‌کند	math.h
sqrt(x)	مجذور **x** را محاسبه می‌کند	math.h
sin(x),cos(x),tan(x)	محاسبه سینوس و کسینوس و تانژانت **x**	math.h
log(x)	محاسبه **log** عدد **x**	math.h

مثال) برنامه‌ای بنویسید که دو عدد را از ورودی بگیرد و عدد اول را به توان عدد دوم برساند.

```
#include<iostream.h>
void main()
{
int x,y,i,j;
cin>>x>>y;
j=1;
for(i=1;i<=y;i++)
j=j*x;
cout<<j;
}
```

این برنامه را می توان به راحتی با تابع pow(x,y) جایگزین کرد.

برنامه‌ای بنویسید که ماکزیمم دو عدد را بدون مقایسه مشخص کند.

```
#include<iostream.h>
#include<math.h>
void main()
{
int a,b;
cin>>a>>b;
cout<<"maximum is:"<<((a+b)+fabs(a-b))/2;
}
```

برنامه ای بنویسید که پس از پاک کردن صفحه نمایش ، در سطر دوم و ستون ۱۴ حاصل عبارت زیر را چاپ کند.

$$\sin\left(\sqrt{\frac{\log(x+yx^2)}{x^3+\sqrt{x+y^2}}}\right)$$

```
//in the name of Allah
#include <iostream.h>
#include <math.h>
#include<conio.h>
main()
{
float x,y;
cin>>x>>y;
clrscr();
gotoxy(14,2);
cout<<sin(sqrt((log(x+y*x*x))/(pow(x,3)+sqrt(x+y*y))));
}
```

برنامه‌ای بنویسید که با گرفتن سه ضلع مثلث مساحت و محیط آن را چاپ کند.

```
#include<iostream.h>
#include<math.h>
#include<stdlib.h>
void main()
{
float a,b,c,d;
cin>>a>>b>>c;
if(!((a+b>c)&&(a+c>b)&&(b+c>a)))
{
cout<<"cannot create triangle" ;
exit(0);
}
cout<<"mohit="<<(a+b+c);
d=(a+b+c)/2;
cout<<"area="<<sqrt(d*(d-a)*(d-b)*(d-c));
}
```

۲. توابعی که توسط خود کاربر تعریف می‌شوند.

اینگونه توابع را خود برنامه نویس می‌نویسد و می‌تواند چندین بار آنها را فراخوانی کند.

توابع یا زیر برنامه‌ها در زبان c++ می‌توانند قبل از تابع main و یا بعد از تابع main تعریف شوند.

نکته: اگر تابعی بعد از main تعریف شود، حتما بایستی اعلان یا الگوی آن قبل از main آمده باشد.

نکته: توابع می‌توانند خروجی داشته باشند یا خروجی نداشته باشند. اگر تابعی خروجی نداشته باشد، قبل از نام آن عبارت void می‌آید و اگر تابع خروجی داشته باشد قبل از نام آن نوع خروجیش را می‌نویسیم و بایستی در این تابع حتما دستور return به کار برده شود.

نکته: دستور return پارامتر جلویش را به تابعی که آن را فراخوانی کرده بر می‌گرداند. تابع می‌تواند پارامتر ورودی داشته باشد یا نداشته باشد. یعنی می‌تواند ۰ و ۱ و یا بیشتر پارامتر ورودی داشته باشد.

نکته: اگر تابعی پارامتر ورودی داشته باشد بایستی نوع آن پارامتر قبل از آن ذکر شود.

به مثال های زیر توجه کنید:

۱) توابع بدون ورودی و بدون خروجی:

```
1) void noble()
{
دستورات
}

2) void nobel()
{
دستورات
}
```

۲) توابع بدون ورودی و با خروجی:

```
3) int mohsen()
{
دستورات
return
}

4) float hasan()
{
دستورات
return
}
```

۳) توابع با ورودی و بدون خروجی:

```
5) void fake(int a)
{
دستورات
}

6) void pseudo(int a,float b,char c)
{
دستورات
}
```

۴) توابع با ورودی و با خروجی:

```
7) int kentucky(char x,float y)
{
دستورات
return
}

8) float state(int s)
{
دستورات
return
}
```

مثال : تابعی که دو ضلع یک مستطیل را گرفته و مساحت آنرا چاپ کند. این تابع دو ورودی دارد اما خروجی ندارد:

```
void area (float a, float b)
{
  float c;
  c=a*b;
  cout<<c<<endl;
}
```

نکته بسیار مهم: داشتن خروجی در توابع به این معنی نیست که تابع چیزی را در خروجی چاپ کند بلکه به این معناست که تابع چیزی را به تابع دیگر (تابع قبلی) که آنرا فراخوانی کرده با دستور return برگرداند.

فراخوانی توابع:

تابعی که نوشته شده است برای استفاده ابتدا باید فراخوانی شود. تابع area در مثال فوق هنوز فراخوانی نشده است . بر خروجی یا عدم خروجی توابع ، دو گونه فراخوانی داریم:

۱. فراخوانی توابعی که خروجی ندارند. (در پاسکال به اینگونه توابع procedure می‌گویند)

برای فراخوانی اینگونه توابع کافیست با آن بصورت یک دستور برخورد کنیم، یعنی نام این تابع نوشته می‌شود و پارامترهای آن قید می‌شود و در آخر کاراکتر ; قرار می‌گیرد.

مثال)

```
1) seek();
2) sick(3,4);
3) mean(x,y);
4) means(k,5);
```

نکته: نام تابع به همراه پارامترهایش در یک دستور نوشته می‌شود. آن دستور چیز اضافه‌ای ندارد.

۲. فراخوانی توابعی که دارای خروجی هستند (در پاسکال به اینگونه توابع function می‌گویند)

با اینگونه توابع بصورت یک مقدار رفتار می‌شود. در حقیقت برنامه نویس پس از فراخوانی این گونه توابع منتظر بازگشت یک مقدار است تا از آن استفاده نماید.

مثال)

```
1) b=bypass();
2) k=delivery(3,x)*f;
3) cout<<sum(x);
4) if(deny(5)>=decline(x);
```

مثال) برنامه‌ای بنویسید که عددی از ورودی بگیرد، سپس آنرا به تابعی ارسال کند و آن تابع تعداد ارقام آن عدد را محاسبه کند و برگرداند و تابع فراخواننده (main) آنرا چاپ کند.

```
#include<iostream.h>
int count(int n)
{
int a=0;
while(n>0)
{
a=a+1;
n=n/10;
}
return a;
}
main()
{
int n;
cin>>n;
cout<<"the count is: "<<count(n);
}
```

نکات:

۱. همیشه برنامه از تابع main آغاز می‌شود.

۲. اگر تابع خروجی نداشته باشد و آن را فراخوانی کنیم پس از بازگشت از زیر برنامه به یک خط بعد از دستور فراخوانی برمی‌گردیم، اما اگر تابع خروجی داشته باشد پس از فراخوانی مقدار برگشت داده شده استفاده می‌شود.

۳. فراخوانی می‌تواند توسط تابع main صورت گیرد یا توسط توابع دیگر. (یعنی توابع می‌توانند توابع دیگر را فراخوانی کنند، به غیر از تابع main)

مثال) برنامه‌ای بنویسید که عددی را از ورودی خوانده و آن عدد را به تابعی به نام complete ارسال کند. تابع مشخص کند که عدد کامل است یا خیر. اگر عدد کامل بود تابع مقدار ۱ و در غیر اینصورت تابع مقدار ۰ را برگرداند و برنامه بر حسب خروجی تابع پیغام مناسب را چاپ کند.

```
#include<iostream.h>
int complete(int n)
{
int i,sum=0;
for(i=1;i<=n/2;i++)
 if(n%i==0)
  sum+=i; // or sum=sum+i;
if(sum==n)
return(1); // or return 1;
else
return(0);
}

void main()
{
int n;
cout<<"Enter Number";
cin>>n;
if(complete(n)==1) // or if(complete(n))
cout<<"it`s complete";
else
cout<<"it`s not complete";
}
```

نکته : دستور return باعث می‌شود که از زیر برنامه خارج شده و به زیر برنامه‌ی فراخوان بازگردیم در اینصورت کلیه دستوراتی که بعد از return آمده‌اند بی مصرف می‌ماند.

برنامه‌ای بنویسید که کلیه اعداد چهار رقمی را که از دو طرف به یک شکل خوانده می‌شود را چاپ کند. (این کار باید با استفاده از تابع انجام شود)
نکته: تابع باید بعد از main قرار گیرد، پس نیاز به اعلان دارد.

```
#include<iostream.h>
int kk(int n);
void main()
{
int i;
for(i=1000;i<=9999;i++)
if(kk(i)==1)
cout<<i<<endl;
}
//*****************
int kk(int n)
{
if((n%10==n/1000)&&((n/100)%10==(n/10)%10))
return 1;
else
return 0;
}
```

معمولا برای خوانایی برنامه ، مرز بین توابع را مانند مثال فوق با کاراکترهایی جدا می کنند.

اعلان توابع:

در برنامه فوق چون تابع kk بعد از main قرار گرفته بایستی تابع را اعلان کنیم. به این ترتیب که نوع خروجی، نام تابع و نوع پارامترها را نوشته و در آخر ; قرار می‌دهیم. خط اعلان توابع باید قبل از main قرار گیرد . توجه داشته باشید که بردن نام پارامترها اختیاری است اما ذکر نوع آن اجباری است.
به اعلانهای زیر توجه کنید:

```
void s(int,int,int);
int mm();
float ali(int,int);
```

برنامه‌ای بنویسید که ضرایب یک معادله درجه‌ی دوم را از ورودی بگیرد، سپس آن را به تابعی ارسال کند، تابع معادله را حل کرده و جواب را چاپ کند.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void root(int,int,int);
void main()
{
clrscr();
int a,b,c;
cin>>a>>b>>c;
root(a,b,c);
}
//*******************************
void root(int a,int b,int c)
{
float delta;
clrscr();
delta=b*b-4*a*c;
if(delta>0)
{
cout<<"x1="<<(-b-sqrt(delta))/(2*a)<<endl;
cout<<"x2="<<(-b+sqrt(delta))/(2*a)<<;
}
else if(delta==0)
cout<<"x="<<-b/(2*a);
else
cout<<"no Root";
}
```

پارامترها:

پارامترها دو گونه‌اند:

۱) صوری (formal): پارامترهای صوری پارامترهایی هستند که در جلوی اسم توابع همراه نوع آنها می‌آیند. این پارامتر ها فقط نماد هستند و مقدار ندارند .

۲) واقعی (actual): پارامترهای واقعی آن مقادیری هستند در موقع فراخوانی در پارامترهای صوری copy می‌شوند.

مثال) برنامه‌ای بنویسید که عددی از ورودی بگیرد، سپس آنرا به تابعی ارسال کند و آن تابع مجموع ارقام آن عدد را محاسبه و چاپ کند.

```
#include<iostream.h>
void sum(int n)
{
int a;
a=0;
while(n>0)
{
a=a+n%10;
n=n/10;
}
cout<<"the sum=: "<<a;
}
void main()
{
int p;
cin>>p;
sum(p);}
```

در مثال فوق ، پارامتر n صوری و پارامتر p واقعی است.

نکته : نام پارامترهای صوری و واقعی میتوانند یکسان نباشند.

نکته : برای سادگی ، به پارامتر های صوری پارامتر (parameter) و به پارامتر های واقعی آرگومان (argument) گفته می شود.

نکته: در هنگام فراخوانی تابع نیازی به ذکر نوع خروجی نیست و نوع خروجی فقط در هنگام اعلان تابع و معرفی تابع استفاده می‌شود.

به مثال زیر توجه کنید:

نادرست cout<<int sum(n); درست cout<<sum(n);

توابع بازگشتی یا خود فراخوان(recursive):

قبلا گفته شد که توابع می‌توانند توسط تابع main فراخوانی شوند. اما خود توابع نیز می‌توانند توابع دیگر را فراخوانی کنند. حتی یک تابع می‌تواند خودش را فراخوانی کند. به اینگونه فراخوانی، فراخوانی بازگشتی گفته می‌شود.

یک تابع بازگشتی بایستی شامل دستورات زیر باشد:

۱. دستوری که مجددا باعث فراخوانی همان تابع شود.

۲. دستوری که باعث می‌شود فراخوانی‌های مکرر خاتمه یابند. (شرط توقف)

بسیاری از مسائل ماهیتی بازگشتی دارند. یعنی برای حل آنها نیاز است که از خودشان استفاده کنیم. مسائلی از قبیل بدست آوردن فاکتوریل، بدست آوردن تعداد ارقام یک عدد، بدست آوردن مجموع ارقام یک عدد، مسئله برج‌های Hanoi، به توان رساندن یک عدد، جستجوی دودویی، سری فیبوناچی و...

مثال) تابع فاکتوریل را هم به صورت بازگشتی و هم به صورت غیر بازگشتی بنویسید.

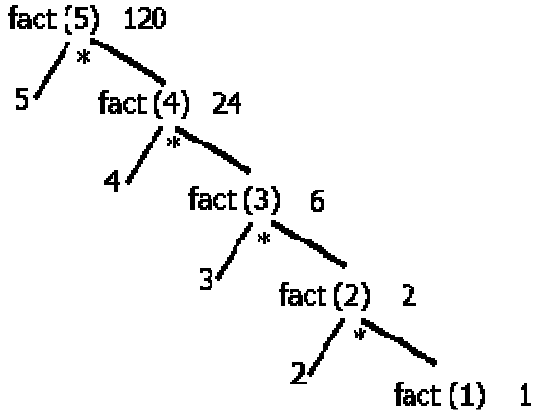
بازگشتی	غیر بازگشتی

long int fact(int n)
{ if(n==1)
return(1);
else
return(n*fact(n-1));
}
	long int fact(int n)
	{ int f=1,i;
	for(i=n;i>=1;i--)
	f=f*i;
	return(f);
	}

در تابع غیر بازگشتی هر بار از مقدار n یک واحد کم شده و در مقادیر قبلیش ضرب می‌شود و در آخر مقدار f با دستور return بازگردانده می‌شود.

در الگوریتم بازگشتی دو شرط داریم: یک شرط توقف که همان (n==1) است. اگر این شرط برقرار نباشد بایستی یکبار دیگر تابع فاکتوریل با پارامتر n-1 فراخوانی شود. پس محاسبه فاکتوریل n-1 نتیجه را به برنامه برمی‌گرداند و این نتیجه‌ی بدست آمده ضرب در n می‌شود و حاصل آن با دستور return به main برمی‌گردد.

برای یادگیری مبحث توابع بازگشتی فراخوانی‌های مکرر را به صورت ساختار درختی نشان می‌دهیم.



برای محاسبه‌ی فاکتوریل ۵ نیاز است فاکتوریل ۴ و سپس فاکتوریل ۳ و فاکتوریل ۲ و فاکتوریل ۱ را محاسبه کرد.

تا وقتی که فاکتوریل ۱ جواب ندهد هیچ چیزی به برنامه‌ی اصلی برگردانده نمی‌شود، زیرا هر بار منتظر بدست آوردن فاکتوریل مقدار جدید هستیم.

تابع فاکتوریل با پارامترهای n=5 و n=4 و n=3 و n=2 و n=1 فراخوانی می‌شود.

اگر شرط توقف نداشته باشیم فراخوانی‌های مکرر انجام می‌شود تا فضای **پشته** پر شود.

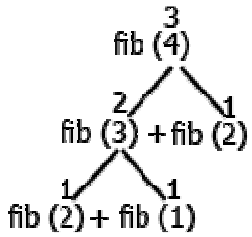
پشته فضایی است که در داخل ram قرار دارد و در هنگام اجرای برنامه از آن استفاده می‌شود.

مثال) بدست آوردن جمله n ام سری فیبوناچی را هم به صورت بازگشتی و هم به صورت غیر بازگشتی بنویسید.

بازگشتی	غیر بازگشتی

int fib(int n)
{
if((n==1)||(n==2)
return(1);
else
return(fib(n-1)+fib(n-2));
}
	int fib(int n)
	{ int f1,f2,sum;
	f1=1;
	f2=1;
	for(i=3;i<=n;i++)
	{ sum=f1+f2;
	f1=f2;
	f2=sum;
	}
	return(sum);
	}

مثال) fib(4) را با استفاده از الگوریتم بازگشتی حساب کنید. (از روش درختی استفاده کنید)

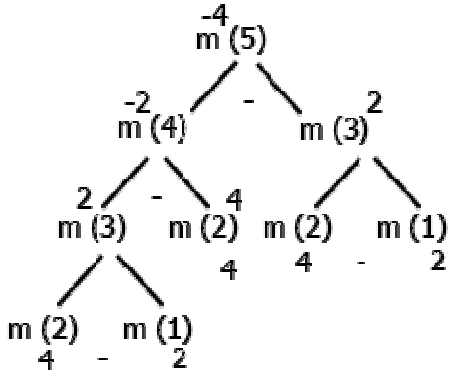


جواب: fib4 برابر با ۳ است.

مثال) m(5) را با توجه به تابع بازگشتی زیر محاسبه کنید.

```
int m(int n)
{
 if((n==1)||(n==0))
 return(2);
 else if(n==2)
 return(4);
 else
 return(m(n-1)-m(n-2));
}
```

جواب: ۴-

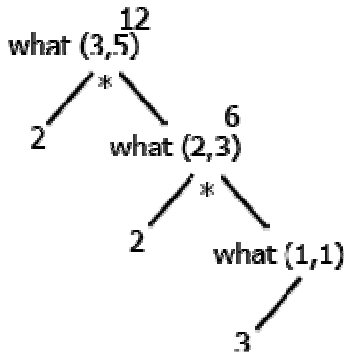


نکته : همانطور که گفته شد دستور return باعث می‌شود که از زیر برنامه خارج شده و به زیر برنامه‌ی فراخوان بازگردیم . در اینصورت کلیه دستوراتی که بعد از return آمده‌اند بی مصرف می‌ماند.

در مثال فوق هرکدام از return ها که اجرا شوند اجرای تابع خاتمه یافته و به تابع فراخواننده بر میگردیم.

مثال) با توجه به تابع بازگشتی زیر what(3,5) را محاسبه کنید.

```
int what(int m,int n)
{ if(m==1)
return(3);
if(n==1)
return(2);
if((m==0)||(n==0))
return(1)
else
return(2*what(m-1,n-2));
}
```



جواب: ۱۲

بازگشتی مجموع ارقام

```
int sum(int n)
{ if(n==0)
return(0);
else
return(n%10+sum(n/10));
}
```

بازگشتی تعداد ارقام

```
int count (int n)
{ if(n==0)
return(0);
else
return(1+count(n/10));
}
```

بازگشتی ب.م.م(gcd)

```
int gcd(int m,int n);
{ if(m%n==0)
return(n);
else
return(gcd(n,m%n));}
```

متغیرهای سراسری(**global**) و محلی(**local**):

اگر متغیری در بیرون از توابع(حتی بیرون main) تعریف شود، آن متغیر سراسری است. و اگر متغیری داخل تابعی تعریف شود، آن متغیر محلی همان تابع است.

به مثال زیر توجه کنید:

```
int a;   //global
main()
{ int b;  //local
}
```

نکته: متغیرهای سراسری در همه‌ی توابع قابل دسترسی هستند اما متغیرهای محلی فقط داخل همان تابعی که در آن تعریف شده اند قابل دسترسی هستند.

برای درک بهتر به کدهای زیر توجه کنید:

```
#include<iostream.h>
int a,b;        //global
int majid(int n)
{ float k;         // local
  int c;           // local
...
}
void main()
{ int p;          //local
  char f;         // local
...
}
```

در کدهای فوق متغیرهای a و b سراسری هستند. یعنی در توابع main و majid قابل دسترسی و استفاده هستند.

n و k و c متغیرهای محلی تابع majid هستند. یعنی فقط در تابع majid قابل دسترسی هستند و در تابع main نمی‌توانند مورد استفاده قرار گیرند.

p و f متغیرهای محلی main هستند.

اگر تابعی متغیرهای سراسری را دستکاری کند مقدار آن در متغیر باقی می‌ماند و این مقدار در فراخوانی های توابع دیگر برای آن توابع قابل دسترسی است.

مثال) خروجی برنامه‌ی زیر چیست؟

```
#include <iostream.h>
int a,b;
void majid()
{ a=2;
b=3; }
void main()
{ a=4;
b=4;
cout<<a<<b;
majid();
cout<<a<<b; }
```

جواب:۴۴۲۳ را چاپ می‌کند.

متغیرهای سراسری همنام با متغیرهای محلی:

چنانچه متغیری درون تابعی هم نام با متغیر سراسری تعریف شود، در آن تابع به متغیر سراسری همنام دسترسی نخواهیم داشت . هر بار که از متغیر همنام استفاده کنیم در حقیقت فقط به متغیر محلی دسترسی داریم نه متغیر سراسری. برای استفاده از متغیر سراسری در آن تابع باید قبل از نام متغیر دو کالن (::) قرار دهیم.

مثال) خروجی برنامه‌ی زیر را مشخص کنید.

```
#include<iostream.h>
int m=8;
void plagiarism()
{ m=10;}
void main()
{ int m=11;
cout<<m<<endl;
plagiarism();
cout<<::m<<endl;
cout<<m;
}
```

جواب: 11
10
11

مثال) خروجی برنامه زیر را مشخص کنید.

```
#include<iostream.h>
int p=1,q=2;
void Ali()
{ int p=8;
cout<<p<<::p<<endl;
}
void main()
{ cout<<p<<endl;
cout<<q<<endl;
Ali();
cout<<p<<q;}
```

جواب: 1
2
81
12

مثال) خروجی برنامه زیر را مشخص کنید.

```
#include<iostream.h>
int a,b=2;
int alternate()
{ a=a*b;
return(a); }
void main()
{ int a,b=3;
a=5;
::a=6;
cout<<a<<b<<endl;
a=b* alternate ();
cout<<a<<::a; }
```

جواب: 53
3612

فرستادن آرایه‌ها به توابع:

می‌توان آرایه ها را نیز مانند متغیرها و اعداد به توابع به عنوان پارامتر ارسال کرد. شروط زیر باید رعایت شوند:

۱. در اعلان توابع نیازی به ذکر نام و اندیس آرایه نیست و فقط نوع آن باید ذکر شود.

۲. در دستور تعریف توابع نیازی به ذکر اندیس نیست ولی نوع آرایه و نام آن و کروشه ها باید قید شوند.

۳. در دستور فراخوانی تابع نیازی به ذکر نوع ، اندیس و کروشه ها نیست و فقط ذکر نام آرایه کافی است.

مثال) برنامه‌ای بنویسید که آرایه‌ای را به تابعی ارسال کند و آن تابع مجموع عناصر آرایه را برگرداند. تابع را بعد از main بنویسید.

```
#include<iostream.h>
#define n 10
int sum(int [],int );      //declaration
 void main()
{ int a[n],i;
for(i=0;i<=n-1;i++)
cin>>a[i];
cout<<sum(a,n);      // call
}
int sum(int b[],int m)   //definition
{ int s=0,i;
 for(i=0;i<=m-1 ;i++)
  s=s+b[i];
 return(s);
}
```

نکته : a[i] یک عدد تلقی می شود نه یک آرایه. چون در حقیقت یک عنصر آرایه است.

سوال : اگر آرگومان تابعی به نام p یک آرایه به نام a با طول ۲۰ باشد و این تابع خروجی داشته باشد کدام دستور فراخوانی زیر می‌تواند درست باشد.

1. cout<<p(a[20])
2. p(a[20])
3. cout<<p(a);
4. p(a);

جواب: شماره‌ی ۳ صحیح است.

سربارگذاری توابع (**overloading**):

در زبان c++ می‌توان توابعی هم نام با هم تعریف کرد. اما بایستی حداقل یک مشخصه از آنها با هم اختلاف داشته باشد. که این اختلاف می تواند در تعداد پارامترهایشان باشد یا در نوع پارامترها . اما تنها در نوع خروجی ها قابل قبول نیست.
کامپایلر بر حسب نوع پارامترهای واقعی و تعداد آنها تشخیص می‌دهد که کدام تابع را فراخوانی کند و خطایی نیز گرفته نمی‌شود مگر آنکه نام توابع، تعداد پارامترها، و نوع پارامترها همه یکسان باشند.
مثال) خروجی برنامه زیر چیست؟

```
#include<iostream.h>
int add(int a,int b)
{ return(a+b);}
int add(float a,float b)
{ return(a+b)*2;}
void main()
{ int m=4,n=10;
float p=3.33,q=4.67;
cout<<add(p,q)<<endl<<add(m,n);}
```

خروجی: ۱۶
۱۴

کلاسهای حافظه

کلاس‌های حافظه مشخص کننده طول عمر متغیرها و مقادیری که می‌توانند بگیرند هستند. چهار نوع کلاس حافظه داریم: auto و static و extern و register که در اینجا فقط دو مورد اول را مورد بررسی قرار می‌دهیم.

۱. auto: تا بحال متغیرهایی که تعریف می‌کردیم از کلاس auto بودند. یعنی نوشتن کلمه‌ی auto اختیاری است.
بنا براین دو دستور زیر با هم برابرند.

```
int a;
or
auto int a;
```

۲. static: چنانچه متغیری از کلاس static تعریف شود دارای دو خصوصیت زیر می‌باشد.

۱. فقط یکبار مقدار اولیه می‌گیرد.

۲. هنگام خروج از تابعی که این متغیر در آن تعریف شده است، این متغیر مقدار نهایی‌اش را از دست نمی‌دهد.

(تذکر : با پایان یافتن یک زیر برنامه ، عمر متغیرهای محلی آن هم خاتمه می یابد و مقدار درون آن متغیر از دست می رود مگر آنکه از کلاس static تعریف شده باشد.)

مثال)

```
#include<iostream.h>
int s()
{ int q=1;   //initialization
cout<<q<<"\n";
q++;
return(q); }
main()
{ cout<<s()<<endl;
cout<<s()<<endl; }
```

خروجی : ۱

۲

۱

۲

مثال)

```
#include<iostream.h>
int s()
{ static int q=1;
cout<<q<<"\n”;
q++;
return(q); }
main() {
cout<<s()<<endl;
cout<<s()<<endl;
}
```

خروجی: ۱

۲

۲

۳

مثال)

```
int vote(int a)
{ static int b;
cout<<b;
b=a+1;
cout<<b;
return (b);}
main()
{ int a=1;
a=vote(2);
a=vote(a);}
```

خروجی: 4 3 3 نامشخص

مثال پایانی :

الگوریتم جستجوی دودویی را با استفاده از توابع بازگشتی بنویسید.

```
#include<iostream.h>
int bsearch(int,int,int);
const int n=10;
int a[n];
void main()
{ int x;
for(int i=0;i<=n-1;i++)
cin>>a[i];
cout<<"enter x:";
cin>>x;
if(bsearch(0,n-1,x)==1)
cout<<"found";
else
cout<<"not found"; }

int bsearch(int low,int high,int x)
{ int mid;
mid=(low+high)/2;
if(low>high)
return(0);
else if(x==a[mid])
return(1);
else if(x<a[mid])
return(bsearch(low,mid-1,x));
else
return(bsearch(mid+1,high,x));
}
```

فصل ششم – رشته ها(strings)

رشته ها در زبان c++ در حقیقت آرایه ای از جنس کاراکتر هستند که می‌توانند تمامی انواع کاراکترها را در خود جای دهند.

دستور زیر تعریف رشته‌ای است به طول ۸:

```
char a[8];
```

در زبان c++ پایان بخش رشته "\0" است. یعنی پایان رشته به این کاراکتر ختم می‌شود. پس اگر رشته‌ای به طول n تعریف کنیم فقط می توان رشته‌ای به طول n-1 در آن ذخیره کرد چون یکی از خانه‌ها به "\0" اختصاص می‌یابد.

```
char a[8];
cin>>a;
```

هرچند متغیر a با طول ۸ تعریف شده است اما ۷ کاراکتر بیشتر نمی‌گیرد.

برای گرفتن رشته از دستور cin استفاده می‌شود. مثلا در بالا cin>>a; باعث می‌شود یک رشته از ورودی گرفته شود و در a قرار داده شود.

برای چاپ رشته از دستور cout استفاده می‌کنیم.

نکاتی در مورد کار با رشته‌ها:

۱- اگر سایز رشته را مشخص نکنیم و به آن مقدار اولیه بدهیم، سایز آن برابر" طول رشته + ۱ " می‌شود. زیرا یک خانه برای ذخیره \0 مصرف می‌شود.

```
char a[]="Ali";
```

A	l	i	\0

۲- اگر بخواهیم کاراکتری از رشته را عوض کنیم، کاراکتر در دو کوتیشن تکی(' ') قرار می‌گیرد.

مثلا در مثال فوق اگر دو دستور زیر را بنویسیم :

```
a[1]='*';
cout<<a;
```

خروجی برابر A*i است. چون رشته آرایه‌ای از کاراکترهاست و آرایه از صفر شروع می شود.

برای کار با رشته از توابعی استفاده می‌شود که الگوی آنها در کتابخانه <string.h> قرار دارد. بعضی از این توابع عبارتند از:

strcat(s_1,s_2)	s2 را به آخر s1 می‌چسباند(الحاق)
strcpy(s_1,s_2)	s2 را در s1 کپی می‌کند(مقدار اولیه s1 از بین می‌رود)
a=strlen(s_1)	طول رشته s1 را در a می‌ریزد
a=strcmp(s_1,s_2)	دو رشته s2 و s1 را با هم مقایسه می‌کند

در strcmp:

اگر $a<0$ آنگاه $s_1<s_2$ (از لحاظ الفبایی)

اگر $a=0$ آنگاه $s_1=s_2$ (از لحاظ الفبایی)

اگر $a>0$ آنگاه $s_1>s_2$ (از لحاظ الفبایی)

مثال)

```
char s1[]="majid ";
char s2[]="nazari";
strcpy(s1,s2);
cout<<s1;
```

خروجی: nazari

مثال)

```
char s1[]="majid";
char s2[]="nazari";
strcat=(s1,s2);
cout<<s1;
```

خروجی: majid nazari

نکته: در تابع strcat و تابع strcpy بایستی طول s1 از s2 بیشتر باشد. در غیر این صورت قسمت اضافی آن حذف می‌شود. در دو مثال قبلی فرض کرده ایم که این امر رعایت شده است.

```
cout<<strlen<<("saeid");        // 5
cout<<strlen("123");            // 3
```

طول رشته را چاپ می‌کند.

مثال) خروجی برنامه زیر را مشخص کنید:

```
char a[]="mirza";
char b[]="zari";
cout<<strcmp(a,b);          //strcmp("mirza","zari");
```

خروجی: یک عدد منفی

توجه داشته باشید مقدار اختلاف برابر اختلاف کد اسکی کاراکترهاست.

به مثال‌های زیر توجه کنید:

```
strcmp("ebrahimzad","ebrahimpoor");
strcmp("ali","Ali"); // a is greater than A
```

آرایه‌ای از رشته‌ها:

اگر بخواهیم بیش از یک رشته تعریف کنیم می‌توانیم از آرایه رشته‌ها استفاده کنیم. به شکل زیر:

```
char a[5][10];
```

این دستور آرایه‌ای به طول ۵ تعریف می‌کند که هر خانه‌ی آن رشته ای به طول ۱۰ است. البته در حقیقت به طول ۹ است.

مثال) برنامه‌ای بنویسید که ۱۰ رشته از ورودی گرفته و یک رشته را جداگانه از ورودی بگیرد و آن رشته را در این ۱۰ رشته به صورت جستجوی خطی جستجو کند.

```
#include<iostream.h>
#include<string.h>
void main()
{ int i;
char x[20],a[10][20];
for(i=0;i<10;i++)
cin>>a[i];
cout<<"enter x:";
cin>>x;
for(i=0;i<10;i++)
 if(strcmp(a[i],x)==0)
  { cout<<"found";
     break;   }
}
```

مثال) برنامه‌ای بنویسید که نام ۱۰ نفر را از ورودی بگیرد و آن‌ها را به ترتیب حروف الفبا مرتب کرده و چاپ کند. از مرتب سازی حبابی استفاده کنید.

```
#include<iostream.h>
#include<string.h>
void main()
{ int i,j;
char a[10][20],temp[20];
for(i=0;i<=9;i++)
cin>>a[i];
for(i=0;i<=9;i++)
 for(j=0;j<=9-i;j++)
  if(strcmp(a[i],a[j+1])>0)
   { strcpy(temp,a[j]);
     strcpy(a[j],a[j+1]);
     strcpy(a[j+1],temp);
   }
 for(i=0;i<=9;i++)
  cout<<a[i]<<endl;
}
```

فصل هفتم – اشاره‌گرها (pointers)

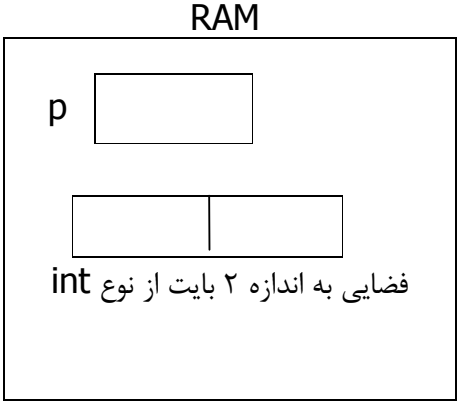
اشاره‌گر متغیری است که به مکانی از حافظه اشاره می‌کند. یعنی در خود آدرس مکانی از حافظه را نگهداری می‌کند.
مقدار درون یک اشاره‌گر برای ما مهم نیست، بلکه جایی که به آن اشاره می‌کند مهم است. اشاره‌گر می‌تواند به متغیرهایی از نوع صحیح،اعشاری،کاراکتر،آرایه و... اشاره کند.
شکل تعریف اشاره گر به صورت زیر است:

نوع * نام اشاره‌گر;

مثال) در دستور زیر p اشاره‌گری است که به متغیری از جنس integer اشاره می‌کند یعنی در p آدرس مکانی از حافظه که متغیر صحیحی وجود دارد قرار می‌گیرد.

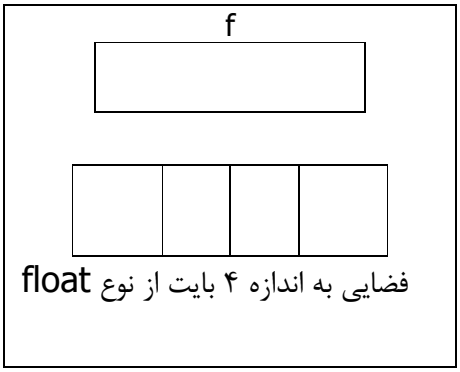
```
int *p;
```

شکل حافظه ای این دستور به صورت زیر است:



مثال)

```
float *f;
```



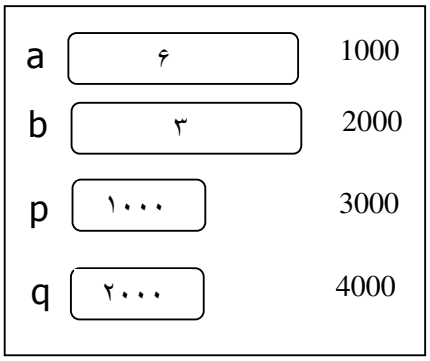
اعمالی که روی اشاره‌گرها انجام می‌شوند:

۱. p=&a آدرس a را در p قرار می‌دهد.
۲. cout<<*p محتویات جایی که p به آن اشاره دارد را چاپ می‌کند. یعنی محتویات جایی که آدرس آن در p قرار دارد را چاپ می‌کند.
۳. cout<<p یعنی محتویات p را چاپ می‌کند.

برای درک بهتر دستورات فوق به مثال‌های زیر توجه کنید: (در این مثالها فرض کرده ایم که آدرسها در حافظه از ۱۰۰۰ شروع شده و به ترتیب هزار تا هزار تا جلو می رود.)

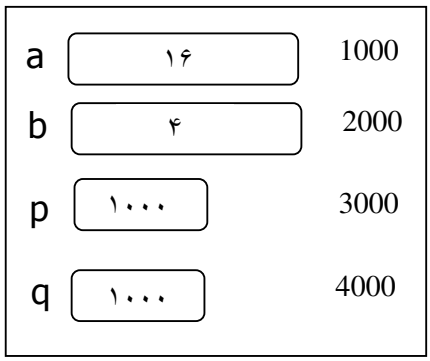
مثال)

```
int a,b,*p,*q;
a=5;
b=3;
p=&a;
q=&b;
a=b*2;
cout<<p<<*p<<endl;        // 1000   6
cout<<q<<*q;              // 2000   3
```



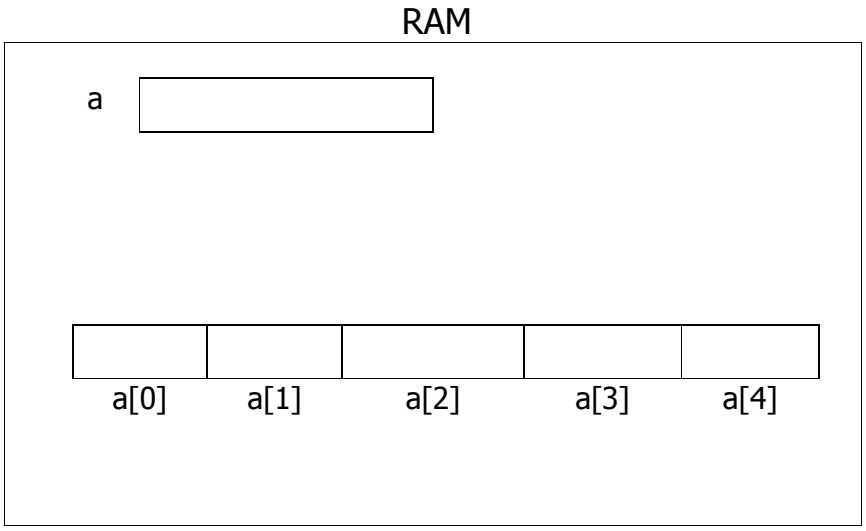
مثال)

```
int a,b,*p,*q;
a=3;
b=4;
p=&b;
q=&b;
a=(*p)*(*q);
p=&a;
q=p;
cout<<p<<*p<<endl;       // 1000  16
cout<<q<<*q;             // 1000  16
```



اشاره‌گرها و آرایه ها:

هنگامی که یک آرایه تعریف می‌کنیم نام آن آرایه در حقیقت نام اشاره گری است که در آن آدرس شروع آرایه در حافظه قرار می گیرد. یعنی نام هر آرایه واقعا یک اشاره‌گر است. آدرس شروع آرایه همان آدرس خانه صفرم آرایه است.



می‌توان گفت دستور فوق معادل دو دستور زیر است:

```
int *a;
a=& a[0];
```

دستور زیر را در نظر بگیرید:

```
int a[5],*p;
```

با توجه به دستور فوق ، این دستورات معادل یکدیگرند.

```
1. p=a;            IS EQUIVALENT WITH    p=&a[0];
2. *(p+3)=2;       IS EQUIVALENT WITH    a[3]=2;
3. *p=*p+3;        IS EQUIVALENT WITH    a[0]=a[0]+3;
```

مثال) خروجی برنامه زیر را مشخص کنید:

```
int a[5],*p,*q;
a[0]=a[1]=a[2]=4;
p=a;
q=&a[2];
*(p+2)=a[1]*a[2];
*q=(*q)+*p;
cout<<*p<<" "<<*q;
```

خروجی 4 20

اشاره گر به اشاره گر

اگر قبل از نام اشاره گر دو ستاره بگذاریم آنگاه آن اشاره گر متغیری است که در آن آدرس اشاره گری قرار می گیرد که در آن اشاره گر آدرس مکانی از حافظه که مورد نظرمان است قرار دارد.

مثال : خروجی تکه برنامه زیر را مشخص کنید:

```
int  a,*q,**p;
a=10;
q=&a;
p=&q;
cout<<a<<" "<<q<<" "<<*q<<" "<<p<<" "<<*p<<" "<<**p;
```

خروجی : 10 address#1 10 address#2 address#1 10

نحوه‌ی فراخوانی و ارسال پارامتر به توابع:

قبلا گفتیم که توابع می‌توانند پارامتر ورودی داشته باشند و یا نداشته باشند. اگر تابعی پارامتر ورودی داشته باشد به سه طریق می‌توان پارامتر را به آن تابع ارسال کرد.

۱. ارسال مقداری

۲. ارسال به صورت مرجع

۳. ارسال به صورت مرجع با استفاده از اشاره‌گر

گفتیم که توابع می‌توانند خروجی داشته باشند که آن را با دستور return بر می گرداند.

اگر بخواهیم تابع ما بیش ازیک مقدار برگرداند تنها دستور return کافی نیست چون این دستور فقط می تواند یک مقدار را برگرداند. اگر بیش از یک return هم استفاده کنیم ، به محض برخورد به اولین return از تابع خارج شده و نوبت به مابقی return ها نمی رسد.

می‌توان پارامترها را طوری به توابع ارسال کرد که تابع روی آنها تغییراتی دهد و موقع برگشت از تابع مقدار تغییر یافته به عنوان خروجی تابع تلقی گردد.

روش دوم و سوم این امر را محقق می‌سازند.

ارسال مقداری call by value:

در این روش در تعریف تابع قبل از نام پارامتر چیزی نمی‌آید. هنگام فراخوانی ، مقدار پارامتر واقعی (actual) در پارامتر صوری(formal) کپی می‌شود و هر تغیری در پارامتر صوری بدهیم آن تغییر در پارامتر واقعی منعکس نمی‌شود. به عبارت دیگر مقدار پارامتر واقعی قبل و بعد از فراخوانی یکی است.

مثال)

```
#include<iostream.h>
void rasul(int a,int b)
{
a=a*2;
b++;
}
void main()
{ int p,q;
p=3;
q=4;
cout<<p<<q<<endl;
rasul(p,q);
cout<<p<<q;
}
```

خروجی : 34
34

ارسال به صورت مرجع(call by reference):

در این روش قبل از نام پارامتر در خط تعریف تابع کاراکتر & قرار می‌دهیم هر تغییری در پارامتر صوری به پارامتر واقعی منعکس خواهد شد. هنگام فراخوانی آدرس پارامتر واقعی و پارامتر صوری یکی می‌شود و می‌توان گفت به یک مکان از حافظه تحت دو نام مختلف دسترسی داریم. به این عمل aliasing گویند.

قبل از نوشتن کد یادآور می‌شویم که تابع بعد از main قرار دارد. (یعنی نیاز به اعلان تابع است)

مثال)

```
void adjust(int a,int &b);
void main()
{ int p,q;
p=3;
q=4;
cout<<p<<q<<endl;
adjust(p,q);
cout<<p<<q;
}
void adjust(int a,int &b)
{ a=a*2;
  b=b*2;
  cout<<a<<b<<endl;
}
```

خروجی: 3 4
6 8
3 8

مثال) خروجی برنامه زیر را مشخص کنید:

```
void alias(int a,int&b,int&c)
{ a=b=c=4;
cout<<a<<b<<c<<endl; }
void main()
{ int a,b,m;
a=2;
b=3;
m=4;
cout<<a<<b<<m;
alias(m,a,b);
cout<<a<<b<<m;
}
```

خروجی: 234444
444

ارسال به صورت مرجع با استفاده از اشاره‌گر:

در این روش قبل از نام پارامتر صوری کاراکتر * و قبل از نام پارامتر واقعی در دستور فراخوانی کاراکتر & می‌آید.
دسترسی به پارامتر به صورت اشاره‌گر است. هز تغییری روی پارامتر صوری برروی پارامتر واقعی منعکس می‌شود. می‌توان گفت در این روش آدرس متغیر ارسال می‌شود نه خود آن.
نکته: در این روش و روش دوم اگر بخواهیم از اعلان تابع استفاده کنیم نیازی به ذکر نام متغیر نیست و فقط کاراکترهای &(روش دوم) و *(روش سوم) کافی است.

مثال)

```
void pointer(int  *);
void main()
{ int a=5;
cout<<a<<endl;
pointer(&a);
cout<<a;
}
void pointer(int *p)
{ *p=*p*2;
cout<<*p<<endl; }
```

خروجی: 5
10
10

مثال)

```
void ahmad(int a,int &b,int *p)
{ a=a*2;
  b=b*2;
  *p=*p*2;
  cout<<a<<b<<p<<*p<<endl;
}
void main()
{int a,b,c;
a=b=c=5;
cout<<a<<b<<c<<endl;
ahmad(a,b,&c);
cout<<a<<b<<c;}
```

خروجی:

```
5     5     5
10   10   ADDRESS   10
5     10   10
```

گرفتن حافظه و حذف حافظه بصورت پویا(**dynamic**):

اگر متغیری را تعریف کنیم تا آخر برنامه طول عمر دارد و نیز تعدادش قابل افزایش نیست. اگر بخواهیم مثلا اسامی ۲۰ نفر را ذخیره کنیم می‌توان یک آرایه ۲۰ عنصری تعریف کرد ولی اگر بعدا بخواهیم نفر ۲۱ ام اضافه شود دیگر نمی‌توان آن را اضافه کرد و باید از دوباره برنامه تغییر یابد. راه حل این مشکل گرفتن حافظه در زمان اجرا است. این کار به کمک اشاره‌گرها انجام می‌گیرد به این ترتیب که ما نام متغیر را تعریف نمی‌کنیم بلکه درزمان اجرا از حافظه به اندازه‌ی مورد نیاز(مثلا به اندازه int یا float) حافظه می‌گیریم و آدرس آن مکان را در اشاره‌گر قرار می‌دهیم.

حتی می‌توان فضای گرفته شده را که آدرس آن در اشاره‌گر قرار دارد باز پس داد. در این صورت فضا هدر نمی‌رود.

گرفتن فضا و بازپس دادن آن به صورت پویا در ساختمان داده هایی مانند پشته و لیستهای پیوندی و درخت ها کاربرد دارد.

دستور new:

این دستور باعث می‌شود که فضا از حافظه گرفته شود و آدرس آن در اشاره‌گر قرار گیرد. شکل کلی این دستور به صورت زیر است:

```
نام اشاره‌گر = new(نوع);
```

مثال)

```
int *p;
p=new(int);
float *q;
q=new(float);
```

دستور delete:

این دستور فضای گرفته شده را که آدرس آن در اشاره‌گر قرار دارد به سیستم باز می‌گرداند. یعنی می‌توان به کمک دو دستور فوق چندین متغیر تعریف کرده، با آنها کار و سپس فضای اختصاص یافته به آنها را باز پس داد.

شکل کلی دستور:

delete نام اشاره‌گر;

مثال) برنامه‌ای بنویسید که ۲ مقدار صحیح را از ورودی بگیرد و آنها را با هم جمع کرده و سپس فضای اختصاص داده شده را برگرداند.

```
void main()
{ int *p,*q;
p=new int;
q=new int;
cin>>*p>>*q;
cout<<*p+*q;
delete p,q;
}
```

مثال) برنامه‌ای بنویسید که با استفاده از اشاره‌گر ۱۰۰ عدد را از ورودی بگیرد سپس مجموع آنها را چاپ کند و تمام آن ۱۰۰ عدد را از حافظه حذف کند.

```
void main()
{ int *p,i,sum=0;
for(i=1;i<=100;i++)
{ p=new(int);
  cin>>*p;
  sum=sum+*p;
  delete (p);
}
 cout<<"sum: "<<sum;}
```

فصل هشتم – مباحث متفرقه

تولید اعداد تصادفی:

بسیاری از برنامه‌ها مانند برنامه‌های شانسی، طالع بینی و پرتاب تاس نیاز به تولید اعداد تصادفی دارند. در غیر این صورت خروجی آنها تکراری می‌شود.

برای این منظور در زبان c++ تابع rand() در نظر گرفته شده است. این تابع در کتابخانه stdlib.h قرار دارد. برای استفاده از این تابع به شکل زیر عمل می‌کنیم:

```
cout<<a+rand()%b
```

در این دستور a حد پایین و b حد بالای اعداد تولید شده خواهند بود و اعداد به صورت صحیح تولید خواهند شد.

مثلا برای پرتاب تاس داریم:

```
cout<<1+rand()%6;
```

برای سکه:

```
cout<<1+rand()%2;
```

مثال) برنامه‌ای بنویسید که کار یک تاس را انجام دهد.

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
void main()
{ srand(time(0));
getch();
cout<<1+rand()%6;
}
```

نکته: در هربار اجرای برنامه درست است که عدد تصادفی تولید می‌شود اما عدد تکراری خواهد بود. برای رفع این مشکل از تابعی به نام srand((time(0)) استفاده می‌کنیم. این دستور را در ابتدای برنامه می‌نویسیم. پارامتر time(0) زمان فعلی سیستم را نشان می‌دهد.

مثال) برنامه فوق را طوری تغییر دهید که مدام این کار را انجام دهد.

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
void main()
{ srand(time(0));
while(1)
{
getch();
cout<<1+rand()%6<<endl; }
}
```

نکته: تجربه نشان داده است که پس از فراخوانی های متعدد تابع rand معمولا اعداد بدست آمده تعدادشان مساوی است.
یعنی مثلا در ۱۰۰ بار پرتاب تاس خروجی به شکل زیر خواهد بود.

۱	۱۷ بار
۲	۱۷ بار
۳	۱۸ بار
۴	۱۷ بار
۵	۱۵ بار
۶	۱۶ بار

برنامه‌ای بنویسید که مشخص کند در ۱۰۰ بار پرتاب سکه چه تعداد شیر و چه تعداد خط آمده است؟

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
void main()
{ int h=0,i,b;
srand(time(0));
for(i=1;i<=100;i++)
{ b=1+rand()%2;
if(b==1)
h++;
}
cout<<"head="<<h<<endl;
cout<<"tail="<<100-h; }
```

مقادیر پیش فرض آرگومانها

C++ این امکان را به ما می دهد که توابعی را تعریف کنیم که دارای آرگومان پیش فرض هستند یعنی اگر این توابع را فراخوانی کنیم و پارامترهایی برایش ارسال نکنیم بصورت اتوماتیک از مقادیر اولیه آرگومانهایش استفاده می کند و اگر مقداری را به عنوان پارامتر ارسال کنیم آنگاه این مقدار استفاده می شود نه مقدار پیش فرض. این ویژگی این امکان را به ما می دهد تا توابعی با اشکال مختلف فراخوانی استفاده کنیم.

مثال :

```
int f (int a=0, int b=1, int c=4)
{return (a*1 + b*2 +c);
}
main ( )
{cout <<f (3, 4, 1) << endl ;
  cout <<f (2, 4) << endl ;
  cout << f (1) << endl ;
  out << f ( );
}
```

هر چهار دستور داخل main درست اند و در نتیجه خروجی برنامه به صورت زیر است:

12
14
7
6

توجه به نکات زیر حائز اهمیت است :

۱- پارمترها از سمت چپ خوانده می شوند و نظیر به نظیر فراخوانی می شوند.

۲- اگر پارامترهای قبلی مقدار اولیه داشته باشند حتماً بایستی پارمترهای بعدی هم مقدار اولیه داشته باشند. اما اگر پارامترهای اولی (جلوتر) مقدار پیش فرض نداشته باشند بعدی ها می توانند مقدار اولیه نداشته باشند تا آنکه یک پارامتر پیدا شود با مقدار اولیه و از آن به بعد بقیه پارمترها بایستی مقدار اولیه پیش فرض داشته باشند.

تعاریف زیر درست اند :

```
int f (int a, int b)
int f (int a, int b, int c=5)
int f (int a, int b=1)
int f (int a , int b=1, int c=6)
```

و دستورهای زیر همگی نادرست اند :

```
int f (int a=0 , int b)
int f (int a=1 , int b , int c)
int f (int a, int b=1, int c)
int f (int a=1, int b=2, int c)
```

نـوع داده اي شمارشي Enumeration

فرض کنید شما می خواهید در برنامه ای فصل های سال را نشان دهید شما می توانید از اعداد صحیحی مانند ۰و ۱و ۲و ۳ استفاده کنید مثلاً فصل بهار عدد ۰ و تابستان عدد ۱ و ...

این برنامه کارخواهد کرد اما نه به صورت خوانا و خوب

چون استفاده از عدد 0 برای نشان دادن فصل بهار وجهه مناسبی ندارد و راه حل خوبی نیز نیست . یکی از نقاط ضعف این روش آن است که محدوده اعداد int از ۳۲۷۶۸- تا ۳۲۷۶۷+ است و اگر ما عددی در این Range (محدوده) به عنوان فصل انتخاب کنیم چون در محدوده int است قابل قبول است اما در واقعیت این امر بی معنی است.

برای حل این مشکل راهی وجود دارد بنام Enum که مقادیر را به مجموعه ای از نامهای سمبلیک محدود می کند.

برای تعریف Enumeration لغت کلیدی enum و سپس مجموعه مقادیر معتبر می آید.

Syntax : enum Name {value1 [= مقدار] , value2 [= مقدار] , ...} ;
Example : enum season { spring, summer, fall, winter} ;

حال در برنامه دستوری به صورت زیر می نویسیم

```
main ( )
{season  a,b;
}
```

نکات :

۱- متغیرهایی که از جنس enum تعریف شوند فقط می توانند یکی از مقادیر شمارشی را بگیرند و خارج از این مجموعه نمی توان به آنها مقدار دهی کرد. مثلاً در زیر دستور ۱ درست و دستور ۲ غلط است (با توجه به مثال فوق) :

1)a = winter ;

2)a = 5;

۲- به هر یک از اعضای این مجموعه عددی اختصاص داده می شود اگر این عدد صریحاً قید شده باشد که هیچ در غیر اینصورت مقادیر از صفر شروع شده و یکی یکی اضافه می شود

یعنی در مثال فوق مقدار متناظر با winter، 3 و مقدار Spring ، 0 است.

زیر نکته ۲ : اگر به یکی از اعضای مجموعه مقدار بدهیم مقدار عضو بعدی (Next) یکی بیشتر از مقدار عضو ماقبلش است.

مثال:

```
Enum    mahdi {a, b = 4 , c, d , m= 50 , k} Ali ;
```

در مثال فوق متغیری بنام Ali از جنس mahdi تعریف شده که می تواند یکی از مقادیر داخل مجموعه را بگیرد.

مقادیر اعضای مجموعه به ترتیب زیر می باشند:

a=0 , b=4 , c=5 , d=6

m=50 , k=51

دستورات زیر همگی درست اند

Ali = b ; Ali = 51; Ali = c ; Ali = 0 ;

و دستورات زیر همگی نادرست اند

Ali = p ; Ali = 52 ; Ali = 7;

مثال ۱ :

```
enum    week { Saturday, Sunday, Monday, Tuesday, wednesday, Thursday, Friday };
main ( )
{week     a , b ;
for ( a= Sunday ; a <= Thursday; a++)
cout << a ;
b = a;
b++ ;
a = Sunday + 11;
Tuesday ++;
}
```

دربرنامه فوق حلقه for از مقادیر 1 تا 5 می شمارد و آنها را چاپ می کند . سپس یک جایگزینی انجام داده و مقدار b را یک واحد افزایش می دهد . دو دستور آخر غلط است. در نهایت در b مقدار ۶ (Friday) قرار می گیرد و با حذف خطوط غلط خروجی به صورت 12345 است.

اگر خطوط غلط را حذف کرده و خط اول را بصورت زیر تغییر دهیم خروجی به صورت 8081828384 میشود.

```
enum    week { Saturday,Sunday=80,Monday,Tuesday,Wednesday,Thursday,Friday };
```

نکته ۳ : در مثال ۱ با دستور enum در حقیقت7 ثابت (const) تعریف کرده ایم که مقادیر 0 تا 6 را در خود دارند یعنی معادل دستورات زیر:

```
# define Saturday   0
# define Sunday     1
        ……..
```

کاربرد Enumeration در برنامه هایی است که آنها داده های : سال، ماه، هفته، روز، ترم و ... استفاده می شود.

فصل نهم – ساختمان‌ها(structs)

در برخی از زبانهای برنامه سازی نظیر پاسکال به ساختمان، رکورد می‌گویند. اگر بخواهیم اطلاعات ۱۰۰ دانشجو شامل نام و نام خانوادگی و معدل را ذخیره کنیم راه اول این است که سه آرایه یکی برای ذخیره نام، دومی برای ذخیره نام خانوادگی(این دو باید از نوع رشته باشند) و سومی برای ذخیره معدل و جنس float تعریف کنیم اما راه بهتر استفاده از structure یا ساختمان است.

یک آرایه تمام خانه‌هایش از یک جنس هستند یعنی همه integer یا float یا char هستند. اما یک ساختمان فیلدهایش می‌تواند از نوع‌های مختلف باشد مثلا یک ساختمان با فیلدهای نام،نام خانوادگی،معدل و...

در حقیقت ما یک نوع داده‌ای جدید(مثل نوع های اصلی C++ نظیر integer،floatو...) تعریف می‌کنیم با نام دلخواه و از آن نوع داده‌ای جدید می‌توان متغیر و حتی آرایه تعریف کرد. به این نوع داده‌ای جدید ADT یا Abstract Data Type می‌گویند.

برای تعریف ساختمان‌ها به روش زیر عمل می‌کنیم:

ساختار کلی این دستور به شکل زیر است:

```
struct   نام ساختمان
{نوع فیلد اول   نام فیلد اول ;
 نوع فیلد دوم  نام فیلد دوم ;
              .
              .
              .
 نوع فیلد آخر   نام فیلد آخر ;
  };
```

مثال)

```
struct student
{ char name[15];
  char family[20];
  int ID;
  float Avg;
  };
```

به وسیله دستور فوق نوع داده‌ای جدیدی بنام student با فیلدهای name و family و ID و Avg تعریف می‌شود. حال می‌توان در برنامه از این نوع داده‌ای جدید استفاده کرد به صورت زیر:

```
student m;
```

به وسیله دستور فوق متغیری به نام m با جنس student تعریف می‌شود که شامل چهار فیلد ذکر شده است.

برای دسترسی به فیلدها از کاراکتر . (dot) استفاده می‌شود به صورت زیر:

```
cin>>m.name;
m.Avg=15.75;
```

بدست آوردن سایز structure یا ساختمان ها:

```
struct student
{ char name[15];          15*1
  char family[20];        20*1
  int ID;                 1*2
  float Avg;              1*4          size=41 byte
};
```

می‌توان آرایه‌ای از جنس ساختمان تعریف کرد به این ترتیب که هر خانه‌ی آرایه حاوی ۴ فیلد است.

```
student a[4];
```

که نمای حافظه ای دستور فوق به صورت زیر است.

name	name	name	name
family	family	family	family
ID	ID	ID	ID
Avg	Avg	Avg	Avg
a[0]	a[1]	a[2]	a[3]

طریقه دستیابی به عناصر آن به صورت زیر است:

```
cout<<a[3].ID;
cin>>a[i].name;
```

نکته: می‌توان متغیر از جنس structure را در پایان structure تعریف کرد به صورت زیر:

```
struct majid
{ int ID;
  int age;
 } k,a[10];
```

با دستورات فوق ساختمانی به نام majid با دو فیلد ID و age ساخته می‌شود و یک متغیر به نام k و همچنین آرایه‌ای به طول ۱۰ به نام a از جنس majid تعریف می‌شود.

تابع sizeof()

این تابع سایز آرگومان خود را بدست می‌آورد مثلا:

```
cout<<sizeof(int)          // 2
cout<<sizeof(student)      // 41
```

نکته: عمل جایگزینی در ساختمان‌ها هم می‌تواند به صورت فیلد فیلد انجام گیرد و هم می‌تواند به یکباره تمامی فیلدها جابه‌جا شوند:

مثال)

```
#include<iostream.h>
struct Ali
{ int ID;
char name[10];
} p;
void main()
{ Ali q;
cin>>p.ID;
cin>>p.name;
q.ID=p.ID;
strcpy(q.name,p.name);
cout<<q.ID<<q.name;
}
```

```
#include<iostream.h>
struct Ali
{ int ID;
char name[10];
} p;
void main()
{ Ali q;
cin>>p.ID;
cin>>p.name;
q=p;
cout<<q.ID<<q.name;
}
```

توضیح: این برنامه ساختمانی به نام Ali با فیلدهای ID و name تعریف کرده و یک متغیر به نام p در بالا و یکی به نام q در پایین(در برنامهmain) تعریف می‌کند سپس p.name,p.ID را از ورودی می‌گیرد و به دو حالت آن را در q می‌ریزد. حالت اول هر فیلد تک تک copy می‌شود. حالت دوم تمام فیلدها به یکباره copy می‌شوند.

مثال) برنامه‌ای بنویسید که مختصات ۲۸ دانشجو شامل نام، نام خانوادگی، جنسیت، معدل را از ورودی گرفته و آنها را بر حسب معدل به صورت صعودی مرتب کرده و چاپ کند.

```
#include<iostream.h>
#define n 28
struct student
{ char name[15];
  char family[20];
  int sex;
  float Avg;
} a[n],temp;
void main()
{ int i,j;
for(i=0;i<=n-1;i++)
{ cin>>a[i].name;
  cin>>a[i].family;
  cin>>a[i].sex;
  cin>>a[i].avg;}
for(i=0;i<n-1;i++)
 for(j=i+1;j<n;j++)
  if(a[i].Avg>a[j].Avg)
   { temp=a[i];
     a[i]=a[j];
     a[j]=temp;
   }
cout<<"name\tfamily\tsex\tAvg\n";
for(i=0;i<=n-1;i++)
{ cout<<a[i].name<<"\t"<<a[i].family<<"\t"<<a[i].sex<<"\t"<<a[i].Avg<<endl;
}
```

یونیون ها (union)

یونیونها نیز مانند ساختمانها تعریف و استفاده می شوند با دو تفاوت زیر:

۱-فیلدهای یونیون به فضای مشترک دسترسی دارند و مقداردهی به هر فیلد یونیون ، باعث دستکاری مقادیر دیگر فیلدهای یونیون می شود.

۲-سایز یک یونیون برابر با سایز بزرگترین فیلد آن است.

مثال) خروجی برنامه زیر چیست؟

```
#include <iostream.h>
union unique{
int m;
double d;
};
void main()
{ unique p;
p.m=5;
p.d=6.22;
cout<<p.m;}
```

خروجی: مقداری نامشخص

مثال: اگر در برنامه بالا unique از نوع struct باشد خروجی آن چیست؟

خروجی ۵ می‌باشد.

مثال : در مثال زیر سایز bachelor را در صورتی که account یکبار یونیون باشد و بار دیگر struct باشد بدست آورید؟

```
union account{
int a ;
float d;
char q;
char r[2];
}bachelor;
```

جواب:

Union: 4
Struct: 9

فصل دهم – فایل‌ها

در برنامه‌هایی که قبلا می‌نوشتیم اطلاعات بر روی RAM ذخیره می‌شد و در پایان اجرای برنامه آن داده ها از بین می‌رفت.
حال می‌خواهیم روشی را معرفی کنیم که داده‌ها را به جای RAM در هارد دیسک ذخیره کند و آنها را بخواند.
نکات:

۱. فایلها در c++ به صورت پیش‌فرض در پوشه‌ای که کد برنامه درآنجا قرار دارد ذخیره و خوانده می‌شوند.
۲. توابعی که در ادامه می‌آیند در کتابخانه(stdio.h) قرار دارند.

کار با فایل‌ها:

برای کار با فایل باید به ترتیب مراحل زیر را انجام داد:

۱. اشاره‌گر از نوع فایل تعریف کرد.

این اشاره‌گر به مکان و نوع و شیوه‌ی دسترسی به فایل اشاره می‌کند. از این به بعد با این اشاره‌گر کار خواهیم کرد. این اشاره‌گر به منزله‌ی شماره‌ی پشت پیراهن بازیکنان فوتبال است که داور با آن شماره کار دارد نه با نام بازیکن.

۲. فایل را بایستی باز کرد برای این منظور روش هایی وجود دارد که نشان می‌دهد چه کاری قرار است با فایل انجام گیرد مثل خواندن یا نوشتن.

۳. پس از باز شدن فایل می‌توان در آن نوشت و یا از آن خواند.

۴. پس از خاتمه‌ی کار، باید فایل را ببندیم.

انواع فایل:

۱. فایل متنی ۲. فایل binary

در فایل متنی می‌توان متن را ذخیره کرد و در فایل باینری می توان متغیرها و رکوردها را ذخیره کرد.
نکته: پسوند فایل را می‌توان fil. و dat. و txt. گذاشت.

تعریف اشاره‌گر به فایل با دستور روبه‌رو امکان پذیر است:

```
FILE *p;
```

دستور باز کردن فایل

```
اشاره گر به فایل=fopen("نام و پسوند فایل" , "روش") ;
```

این دستور فایل مورد نظر را که نامش آمده است با روشی که قید شده است باز می‌کند و شیوه‌ی آن را در اشاره‌گر قرار می‌دهد.
مثال)

```
p=fopen("sajad.txt","w");
```

دستور فوق فایلی به نام sajad.txt را که در پوشه‌ی پیش فرض قرار دارد برای نوشتن درآن باز می‌کند و این mode را در p قرار می‌دهد. یعنی هر بار که به p دسترسی پیدا کنیم مشخصات فایل و شیوه‌ی دستیابی به آن داخل p قرار دارد.

شیوه‌های دستیابی(**modes**):

شیوه	کار مورد نظر
r (read)	فایل برای خواندن باز می‌شود.
w (write)	فایل برای نوشتن باز می‌شود. اگر از قبل نباشد، ایجاد می‌شود. اگر باشد دوباره‌نویسی می‌شود.(محتویات قبلی از بین می‌رود)
a (append)	فایل برای نوشتن در آخرش باز می‌شود. اگر از قبل نباشد، ایجاد می‌شود.
r+	فایل برای خواندن و نوشتن باز می‌شود
w+	فایل برای خواندن و نوشتن باز می‌شود. اگر از قبل نباشد ایجاد می‌شود. اگر باشد دوباره‌نویسی می‌شود.(محتویات قبلی از بین می‌رود)
a+	فایل برای نوشتن و خواندن و تغییرات در آخرش باز می‌شود. اگر از قبل نباشد، ایجاد می‌شود.
t	فایل متنی
b	فایل باینری

نکته: t و b به تنهایی به کار نمی‌روند بلکه به r و w و a می‌چسبند.

همانطور که در جدول قابل ملاحظه است ، با دستور fopen میتوان فایلهایی را که قبلا ایجاد نشده اند ، ایجاد کرد.(با مد w و w+ و a و a+)

مثال)

```
q=fopen("majid.fil","rt");
```

فایل majid.fil به صورت متنی برای خواندن باز می‌شود.

```
p=fopen("Ahmad.dat","w+b")
```

فایل Ahmad.dat به صورت باینری برای نوشتن و خواندن باز می‌شود.

تابع fseek: این تابع برای بردن مکان نما به مکان مورد نیاز در فایل به‌کار می‌رود.

شکل کلی این دستور به شکل زیر است:

fseek (فایل اشاره‌گر , مقدار جابجایی , مکان) ;

مثال)

```
fseek(p,10,SEEK_END);
```

این دستور می‌گوید در فایلی که p به آن اشاره می‌کند از آخر فایل ۱۰ بایت حرکت کن.

در پارامتر مکان ، ثوابت یا اعداد زیر را می‌توان قرار داد:

	مقدار	ثابت
اول فایل	0	SEEK_SET
مکان فعلی	1	SEEK_CUR
آخر فایل	2	SEEK_END

مثال) این دستور از مکان فعلی ۲ بایت جلو می‌رود:

```
fseek(p,2,SEEK_CUR);
```

تابع ftell(اشاره‌گر به فایل):

این تابع مکان فعلی را در فایل نشان می‌دهد.

مثال) این دستور می‌گوید در فایل p مکان نما الان کجاست:

```
ftell(p);
```

مثال :

```
fseek(p,ftell(p),SEEK_SET);
```

از ابتدای فایل مکان نما را به اندازه‌ی موقعیت کنونی مکان نما جلو می‌برد. یعنی در حقیقت مکان نما را سرجای خود تثبیت می‌کند.

نکته: برای فایل‌های متنی مقدار جابجایی کاراکتر به کاراکتر است و برای فایلهای باینری مقدار جابجایی بایت به بایت است.

تابع fprintf :

این تابع برای نوشتن رشته ها و مقادیر در فایل بکار می رود.

fprintf(آرگومانها,رشته یا کاراکترهای فرمت بندی,اشاره گر به فایل);

کاربرد کاراکترهای فرمت بندی مانند تابع printf است. (رجوع کنید به تابع printf در صفحه ۷۸)

مثال:

```
p=fopen ("a.txt","w+");
fprintf (p, "In The Name Of Allah");
```

رشته ای را در فایل متنی می نویسد.

مثال:

```
int i=2;
float f=3.4;
P=fopen("A.fil","w+b");
fprintf(P,"%d %f",i,f);
```

دو مقدار صحیح و اعشاری را در فایل باینری می نویسد.

تابع fscanf :

این تابع مقادیری را از فایل می خواند و در متغیر مربوطه قرار می دهد. (ر.ک . تابع scanf در صفحه ۷۸)

fscanf(متغیر ها ,رشته یا کاراکترهای فرمت بندی,اشاره گر به فایل);

این تابع در فایلهای متنی با رسیدن به کاراکتر SPACE متوقف می شود.

مثال:

```
int a;
P=fopen("A.fil","rb");
fscanf(p,"%d",&a);
```

تابع fputs:

مثال)

fputs(متغیر رشته‌ای یا رشته , اشاره‌گر به فایل) ;

این تابع رشته را تا رسیدن به کاراکتر newlline (\n) در فایل می‌نویسد.

مثال)

```
fputs("Ahmad\n",p);
```

اگر \n را نگذاریم داده‌ها را پشت سر هم می‌نویسد.

تابع fgets:

```
fgets(متغیر رشته‌ای , تعداد کاراکتر , اشاره گر به فایل )  ;
```

این تابع به اندازه‌ی "تعداد کاراکتر منهای یک از فایل" می‌خواند و در متغیر رشته‌ای قرار می‌دهد.

نکته: اگر این تابع به کاراکتر \n به خط جدید برسد متوقف می‌شود و باقی تعداد کاراکترها را نادیده می‌گیرد.

مثال)

```
fgets(k,33,p);
```

این دستور از فایل p ، 33-1=32 کاراکتر خوانده و در k قرار می‌دهد.

تابع fclose(اشاره‌گر به فایل):

این تابع فایلی را که قبلا باز شده بود با استفاده از اشاره‌گرش می‌بندد.

مثال)

```
fclose(p);
```

اگر بیش از یک فایل باز بود و خواستیم همه آنها را ببندیم می توان از تابع fcloseall به صورت زیر استفاده کرد:

```
fcloseall();
```

اگر فایل را پس از استفاده نبندیم ممکن است فایل صدمه ببیند یا اطلاعاتی ناخواسته وارد آن شود.

مثال : برنامه ای که کاربرد دستورات بالا را نشان می دهد.

```
//In The Name Of CREATOR
#include<iostream.h>
#include <stdio.h>
#include <string.h>
void main()
{
FILE *p;
char w[10];
p=fopen("ww.txt","at");
cout<<ftell(p)<<endl;
fprintf(p,"not only - but also");
cout<<ftell(p)<<endl;
fclose(p);
p=fopen("ww.txt","r");
fscanf(p,"%s",w);
cout<<w<<endl;
fseek(p,15,SEEK_SET);
fgets(w,5,p);
cout<<w;
fclose(p);}
```

خروجی این برنامه در صورتی که فایل قبلا وجود نداشته باشد یا خالی باشد ، بصورت زیر است:

0
19
not
also

تابع fwrite:

مثال)

fwrite(اشاره‌گر به آرایه یا رکورد یا متغیر , size , n , p) ;

این تابع در فایلی با اشاره گر p ، n آیتم از داده ها را که هرکدام به اندازه‌ی size طول دارند را در فایل می‌نویسد. آدرس آن داده‌ها در اشاره‌گر قرار دارد.

مثال)

```
int m,*p;
p=&m;
fwrite(p,2,4,q);
```

این دستورات ابتدا یک اشاره‌گر به متغیری صحیح تعریف می‌کنند. سپس در فایلی که اشاره‌گر آن q است چهار آیتم که هرکدام ۲ بایت طول دارند را از p استخراج کرده و می‌نویسد.

تابع fread:

مثال)

fread(متغیر یا به رکورد یا به آرایه اشاره‌گر , size , n , p) ;

این تابع مانند تابع قبلی n آیتم از داده‌ها را که هرکدام به اندازه‌ی size طول دارند از فایل p می‌خواند و در جایی که اشاره‌گر به آن اشاره می‌کند قرار می‌دهد.

تابع feof: (end of file)

این تابع انتهای فایل را چک کرده و اگر به آخر فایل رسیده بود مقدار ۱ و در غیر اینصورت مقدار صفر را بر می گرداند.

مثال:

```
#include<iostream.h>
#include <stdio.h>
void main()
{
FILE *p;
char w[20];
p=fopen("ww.txt","r");
while (!feof(p)){
fgets(w,10,p)    ;
cout<<w<<"\t";
}
fclose(p);}
```

برنامه فوق از ابتدا تا انتهای فایل ww به صورت ده تا ده تا کاراکتر می خواند و در خروجی چاپ می کند..

توابع دیگری به نام fputc و fgetc نیز وجود دارند که برای مطالعه می‌باشد.

مثال پایانی در رابطه با فایل‌ها:
برنامه‌ای بنویسید که فایل 1.txt و فایل 2.txt را بگیرد و از فایل ۱ رشته‌ای خوانده و از فایل ۲ نیز رشته‌ای بخواند و این دو رشته را به هم متصل کند و حاصل را در فایل 3.txt ذخیره کند. (این کار را برای ۱۰ خط از فایل‌ها انجام دهد)

```
#include<iostream.h>
#include<stdio.h>
#include<string.h>
void main()
{ FILE *p,*q,*r;
int i;
char a[100],b[50];
p=fopen("1.txt","rt");
q=fopen("2.txt","rt");
r=fopen("3.txt","wt");
for(i=1;i<=10;i++)
{ fgets(a,100,p);
fgets(b,50,q);
strcat(a,b);
strcat(a,"\n");
fputs(a,r);}
fclose(p);
fclose(q);
fclose(r);
}
```

فصل یازدهم

برنامه نویسی شی‌گرا: object oriented programming

تا به حال برنامه‌هایی که می‌نوشتیم برنامه‌های ساخت یافته (structred) بودند. برنامه‌های ساخت یافته پس از برنامه‌هایی آمدند که در آنها دستور goto و jump استفاده می‌شد. پس از برنامه‌نویسی ساخت یافته که در آن دستورات else و if و swtich و for و while استفاده می‌شد. برنامه نویسی شی‌گرا روی کار آمد. در این قسمت از برنامه نویسی ما با مولفه‌هایی به نام شی کار داریم که این اشیا را می‌توان چندین بار استفاده کرد و نیز از برنامه‌ای به برنامه‌ی دیگر جابه‌جا کرد. یک شی یا object تشکیل شده است از تعدادی از داده‌ها (صفات) و توابعی که روی این داده‌ها کار انجام می‌دهند(متد).

به این ویژگی که صفات و متدها در کنار هم قرار دارند، بسته بندی یا encapsulation گویند. حال شیءی که کپسوله‌سازی شده را می‌توان در برنامه‌های دیگر به کرات استفاده کرد. بدین ترتیب حجم برنامه‌ها کاهش یافته و خوانایی برنامه‌ها افزایش می‌یابد.

نکته: لفظ شی را با آنچه در ذهن دارید اشتباه نگیرید!

می‌توان مثالی در این زمینه زد:

مثلا کارخانه‌ای برای تولید ماشین از قطعاتی که از بیرون تهیه می‌شود استفاده می‌کند و فقط آنها را مونتاژ می‌کند و ارتباط بین آنها را برقرار می‌سازد. این کارخانه می‌تواند هریک از قطعات ماشین را از جاهای مختلف تهیه کند، و همچنین کارخانه‌های متعددی می‌توانند از همان قطعه استفاده کنند.

مثال ۲: اشیائی که در toolbox ویژوال بیسیک قرار دارند هرکدام دارای صفات و متدهای خود هستند.

مثلا شی (command button) صفاتی مثل caption و name دارد و متدهایی مانند click و set focus دارد.

برای استفاده از شی گرایی در c++ ابتدا class ها را تعریف می‌کنیم سپس هر متغیری که از جنس این کلاس تعریف شود، شی نام دارد یعنی می‌توان گفت class ها مجموعه‌ای از اشیا را در بر می‌گیرند که دارای صفات و متدهای مشترک هستند.

مثلا کلاس انسان یا human صفاتی مانند قد، وزن، جنسیت، و... دارد و متدهایی مانند راه رفتن، خوابیدن، غذا خوردن. و هر فردی را که در نظر بگیریم عضوی از این کلاس است.

تعریف class در زبان c++:

```
class{
private:
صفات و متدها
public:
صفات و متدها
protected:
صفات و متدها
};
```

private (خصوصی) : صفات و توابعی که به صورت private (اختصاصی) تعریف می‌شوند در همه توابع قابل دسترسی نیستند و فقط در توابع مختص آن class و توابع دوست آن کلاس قابل دسترسی هستند.
public (عمومی): صفات و متدهایی که به صورت public تعریف شوند در همه توابع بعدی (بعد از اعلان) قابل دسترسی هستند.
protected (حفاظت شده): اینگونه صفات و متدها که به صورت protected تعریف می‌شوند در توابع مختص آن class و توابع دوست آن کلاس قابل دسترسی هستند. و همچنین در کلاسهای مشتق شده از آن کلاس قابل دسترسی هستند و برای ارث بری یا وراثت کاربرد دارند.
نکته: کلمات private و public و protected با هر ترتیبی و با هر تعدادی قابل استفاده‌اند.
نکته: اگر قبل از صفات و متدها چیزی نیاید به صورت پیش فرض private در نظر گرفته می‌شود.
مثال)

```
class student
{ float Avg;
protected: char family[20];
private: int age;
void kk()
{ cout<<"salam";}
protected:
int m,n;
public:
void ss();
};
```

دستورات فوق class ای به نام student تعریف می‌کند. متغیری به نام Avg و age و تابعی به نام kk از نوع private تعریف می‌کند و رشته ای بنام family و متغیرهایی به نام n و m از نوع protected و تابع (متدی) به نام ss از نوع public تعریف می‌کند.
متغیر Avg چون قبلش لغتی نیامده است به صورت پیش فرض از نوع private است.
حال در برنامه اصلی دستورات زیر را داریم:

```
void main()
{ student p;
p.age=5;
p.ss();
}
```

در تابع main، p شی‌ای است از کلاس student. این شی تمام صفات و متدهای class ِ student را دارا می‌باشد و لذا به صفت age و متد ss دسترسی پیدا کرده است.
توابع(متدهای) یک class را هم می‌توان در داخل خود کلاس دستوراتش را نوشت و هم می‌توان در بیرون class این کار را کرد. در مثال فوق تابع kk دستوراتش داخل class نوشته شده است. اما تابع ss دستوراتش داخل class نوشته نشده است لذا باید در بیرون class آنها را تعریف کرد. برای تعریف دستورات متدهای یک class دربیرون کلاس، نام class آمده و دو کالن (::) بعد ازآن می‌آید و سپس نام تابع دستوراتش می‌آید به صورت زیر:

```
void student:: ss()
{ cout<<"Ali";
}
```

توابع سازنده (**constructor**):

تابعی است که هنگامی که شیای از جنس کلاس تعریف می‌شود، اجرا می‌گردد. این تابع نیز جزء کلاس است و دقیقا همنام با کلاس است.

نکته: هنگام تعریف شی و هنگام فراخوانی تابعی که در آن شیای تعریف می‌شود، تابع سازنده می‌تواند اجرا شود.

توابع مخرب (**destructor**):

تابعی است که هنگامی که شیای از جنس کلاس عمرش به پایان می‌رسد اجرا می‌گردد. این تابع جزء کلاس است و همنام با نام کلاس است اما قبل از اسم آن کاراکتر ~ (tilde) می‌آید.

نکته: هنگام بازگشت از تابع، عمر متغیرهای محلی آن تابع نیز به پایان می‌رسد. همچنین اگر در آن تابع شیای تعریف شده باشد عمرش به پایان می‌رسد. در این صورت تابع مخرب آن کلاس می‌تواند اجرا شود.

توابع دوست (**friend**):

این گونه توابع داخل کلاس اعلان می‌گردند ولی کد آنها معمولا بیرون از کلاس نوشته می‌شود.

این گونه توابع جزء کلاس نیستند اما دارای قابلیت دسترسی به عضوهای private کلاس هستند. این توابع را می‌توان در هرجای کلاس اعلان کرد. (فقط اعلان)

مثال)

```
class student{
        friend void print();
        public:
        int m,n;
        student();
       ~student();
      };
```

مثال) برنامه ای مربوط به چگونگی کارکرد توابع سازنده و مخرب.

```
#include <iostream.h>
class student{public:
                char name[10];
                float avg;
                student();
                ~student();
                };
void main()
{ cout<<"Hello";
 student k;
}
student::student()
{ cout<<"salam";}
student::~student()
{ cout<<"goodbye";}
```

خروجی: Hello salam goodbye

مثال) برنامه ای مربوط به چگونگی کارکرد توابع دوست. (برنامه یک نام و یک عدد را می گیرد و آنها را چاپ می کند)

```
#include<iostream.h>
class student{private:
                   char name[10];
                   float avg;
                   get();
                   print();
                 friend void start();
                 };
void main()
{ start();
}
//*****************
void start()
{ student p;
  p.get();
}
//*****************
student::get()
{ cin>>name>>avg;
  print();
}
//*****************
student::print()
{
cout<<name<<" "<<avg;
}
```

مثال : از توابع سازنده و مخرب

```
#include <iostream.h>
int      s1 = 0 , s2 = -1;
class   pnu {
          Public:
               int     n ;
               pnu ( ) {n = s1; s1 ++ ; s2 ++; }
              ~pnu ( ) {s2 = s1 * 2; }
          } ;
void   f () {pnu   A, B; }

void   main ( ) {
         pnu  x, y ;
         cout <<s1 + s2 << "\n" ;     f ( ) ;     cout <<s1 + s2;
}
```

خروجی : ۳

۱۲

نکات تکمیلی در مورد کلاسها و اشیاء

۱- معمولا متدهای یک کلاس را public و صفات را private (خصوصی) تعریف می کنند.

۲- به عملگر : : عملگر جدا سازی دامنه می گویند.

۳- اگر تمامی متدهای یک کلاس را داخل خود کلاس تعریف کنیم و کدشان را بنویسیم، آنگاه آن کلاس را خودکفا گویند. اما این کار چندان خوب نیست زیرا با خاصیت نهان سازی اطلاعات تناقض دارد.

۴- در برخی از کتاب ها تابع مخرب را نابودگر (نابودکننده) نیز می گویند.

۵- تابع سازنده معمولاً از حوزه public تعریف می شود.

۶- یک کلاس می تواند چندین سازنده داشته باشد اما فقط یک مخرب می تواند داشته باشد.

۷- توابع سازنده همنام بایستی overload شده باشند یعنی با یکدیگر در تعداد پارامترها و نوع پارامترها اختلاف داشته باشند که در این صورت با نوشتن پارامتر جلوی شی در هنگام تعریف شی می توان تعیین کرد که کدام سازنده فراخوانی شود.(ر.ک مثال ۲)

۸- اگر برای کلاس سازنده مشخص نکنیم خود C++ بصورت پیش فرض یک سازنده برای در نظر می گیرد. این سازنده بدون پارامتر است.

۹- اگر برای کلاس بیش از یک سازنده تعریف کنیم باید مشخص کرد که کدام سازنده اجرا شود وگرنه سازنده پیش فرضش (سازنده بدون پارامترش) اجرا خواهد شد.

۱۰- اگر برای کلاس مخرب تعیین نکنیم نیز C++ خودش یک مخرب در نظر می گیرد.

۱۱- اگر برای کلاسی سازنده یا مخربی خودمان تعریف کنیم عمل ما را تعریف صریح (Expilicit) نامیده و در غیر اینصورت عمل در نظر گرفتن سازنده و مخرب پیش فرض توسط C++ را ضمنی (Impilicit) گویند.

۱۲- به داده های عضو (صفات) در محل اعلانشان در کلاس نمی توان مقدار اولیه داد (Initialize) مگر اینکه static باشد و لذا برای این کار از تابع سازنده استفاده می شود.

۱۳- بهتر است توابعی که در هیچ برنامه دیگر پیاده سازی شان تغییر نمی کند را در داخل کلاس تعریف کنیم.

۱۴- تابع مخرب پارامتر نمی گیرد خروجی هم نمی دهد یعنی سربار گذاری (overload) نمی شود.

۱۵- اگر کلاسی سازنده غیر پیش فرض داشته باشد دیگر نمی تواند سازنده پیش فرض داشته باشد.

۱۶-عمل جایگزینی عضو به عضو assignment by members به صورت زیر تعریف می شود که می توان یک شی را با عملگر انتصاب درون شیء دیگر ریخت در این صورت صفات نظیر به نظیر کپی خواهند شد.(ر.ک مثال ۱)

۱۷- برای تابع سازنده و مخرب یک کلاس نمی توان نوع برگشتی حتی void را تعریف کرد و این دو نوع تابع بصورت ساده می آیند یعنی مثلاً دستورات زیر منجر به Error می شوند:

```
class mahdi {
                    void mahdi ( ) {  . . . . . }
                };
```

مثال ۱. عمل جایگزینی عضو به عضو (نکته ۱۷):

```
class eye {    int   p ;
               public :
                    float   k ;
         } ;
main ( )
{eye   m ,n;
    m.k = 3 ;
    n = m ;
}
```

در مثال بالا فیلدها (صفات) شئ m در شئ n نظیر به نظیرکپی می شوند. یعنی m.k در n.k و m.p در n.p کپی خواهند شد.

مثال۲. توابع سازنده سربارگذاری شده (همنام) (نکته ۷):

```
class vehicle {
                    vehicle ( )                 { caut << "fortune ";}
                    vehicle (int n)             {cout << "fortunately" ; }
                    vehicle ( int m , int n) {cout <<"unfortunately" ; }
                } ;
main ( )
{
 vehicle protest (2);
 vehicle bulk ;
 vehicle bold (2 , 2);
}
```

خروجی بصورت روبرو می باشد: fortunately fortune unfortunately

در برنامه بالا یک کلاس با ۳ سازنده همنام اما پارامترهای متفاوت تعریف شده است. در تابع main سه شیء تعریف شد که با توجه به عبارتی که جلوی شیء در هنگام تعریف شیء نوشته می شود کامپایلر تصمیم می گیرد که کدام یک با سازنده ها match می شود (جور می شود) و سازنده مناسب فراخوانی می گردد.

فصل دوازدهم – ارث بری یا وراثت
(inheritance)

صورتی از استفاده مجدد از نرم افزار است. قابلیت استفاده مجدد از نرم افزار باعث صرفه جویی زیاد در وقت برنامه نویسی هنگام طراحی می گردد.
برنامه نویس هنگام ایجاد کلاس بجای نوشتن داده ها و متدهای جدید می تواند تعیین کند که کلاس جدید آنها را از کلاس [های] موجود به ارث ببرد.
کلاس موجود را کلاس پایه (پدر ، والد) و کلاس وارث را کلاس مشتق (فرزند) گویند.

وراثت انواعی دارد مانند یگانه و جندگانه :
۱_ وراثت یگانه :
وراثت یگانه یعنی کلاس فقط یک پدر داشته باشد.
۲_ وراثت چندگانه :
یعنی یک کلاس از چند کلاس دیگر مشتق شده باشد و بعضی از صفات را از یک کلاس و بعضی دیگر را از کلاس های دیگر به ارث ببرد.

« هر وسیله نقلیه ماشین نیست ولی هر ماشین یک وسیله نقلیه است»

ارث بری یا وراثت به سه روش public و protected و private قابل انجام است اما در اینجا ما فقط به روش public می پردازیم.

syntax رابطه وراثت

```
class DERIVED : public BASE {
                                  متدهای صفات مربوط به وارث
      نام کلاس مشتق      نام کلاس پایه
                                  }
```

نکات:

۱- کلاس مشتق شده علاوه بر خصوصیات کلاس پدرش خصوصیات منحصر به فرد را هم می تواند داشته باشد.
(ر.ک مثال ۱)
۲- توابع friend به ارث نمی رسند.
۳- سازنده ها و مخرب ها به ارث نمی رسند.

۴- هر عضوی (متد یا صفت) که به ارث می رسد حوزهٔ خودش را در کلاس مشتق حفظ می کند. یعنی اگر در کلاس پایه public بوده در کلاس مشتق نیز public است. همچنین است protected و private .(ر.ک مثال ۵)

۵- یک کلاس مشتق شده می تواند به اعضای غیر private کلاس پایه دسترسی یابد. یعنی توابع عضو یک کلاس مشتق نمی توانند مستقیماً به صفات یا متدهای خصوصی کلاس پایه دسترسی یابند مگر از راه توابع friend. (ر.ک مثال ۲)

۶- یک کلاس پایه می تواند چندین کلاس مشتق داشته باشد. (یک پدر چند فرزند می تواند داشته باشد)

۷- در یک کلاس مشتق شده به اعضای private یک کلاس پایه دسترسی نداریم با اینکه همان عضو را هم داریم.

۸- می توان تابعی را که به ارث برده شده دوباره تعریف کرد (بازنویسی کرد) و در این حالت است که می توان اعضای public یا protected کلاس پایه را دستکاری کرد. در صورت ذکر نام متد دوباره نویسی شده ، متد کلاس مشتق مورد نظر است نه متد همنام در کلاس والد. برای دسترسی به متد همنام در کلاس والد بایستی از عملگر جداسازی (::) دامنه استفاده کرد. (ر.ک مثال ۳ و ۴)

۹- در یک کلاس مشتق شده می توان به اعضای protected کلاس پایه دسترسی داشت.

مثال ۱. تعریف کلاس مشتق (نکته ۱)

```
class AM { int a;
                public :
                   int b;
                     void m ( ) {cout <<"*" ;}
                protected :
                     int c;
                     void n ( ) {cout <<"$"; }
             };
 class ARE : public AM { public:
                                  int k;
                                  void p ( );
                               private:
                                  int d ;
                             };
```

در مثال فوق کلاس AM کلاس پایه و کلاس ARE کلاس مشتق (کلاس وارث) است.

نوع ارث بری از جنس public است.

کلاس ARE تمانی صفات و متدهای کلاس AM رادارا می باشد و علاوه بر اینها صفاتی مثل k و d ومتدی به نام p را اضافه تر دارد.

در کلاس ARE به متغیر a دسترسی نداریم چون در کلاس AM به صورت private است اما به مابقی صفات و متدها دسترسی داریم چون از جنس public یا protected اند.

مثال ۲. (نکته ۵) در برنامه زیر مشخص کنید کدام خطوط Error دارند و اگر Error ها را برطرف کنیم خروجی چیست ؟

```
class A { private :
                  int a;
          public :
                  int b;
                  void kk ( ) {cout <<"$";}
          protected :
                  void M ( ) {cout <<"*"; }
                  int d ;
        };

Class B : public A {     int s ;
                          public :
                             void z ( ) {cout << a;}       //1
                          protected :
                             void E ( ) {M ( ) ; }       // 2
                        };
main ( )
{    A    N ;
     B    L;
     N.d = 1       // 3
     N.kk ( );     // 4
     L.E ( );      // 5
     N.M ( );      // 6
}
```

خط 1 غلط است زیرا به a که یکی از اعضاء private کلاس A است دسترسی یافته ایم.

خط 3 غلط است زیرا به یکی از اعضای protected کلاس A به نام d دسترسی یافته ایم.

خط 5 غلط است زیرا به یکی از متدهای protected کلاس B دسترسی داشته ایم.

خط 6 هم غلط است به همان دلیل خط سوم.

خروجی در صورت حذف خطوط غلط چاپ یک علامت دلار ($) است.

توضیح برای خط شماره 2 :

با اینکه متد M یک متد protected است و در بیرون از کلاس نمی توان به آن دسترسی داشت اما در کلاس وارث (مشتق) می توان به اعضای protected دسترسی پیدا کرد.

مثال ۳. بازنویسی متدهای به ارث برده شده (نکته ۸):

```
class   A { public :
                  void M ( ) {cout <<"Hi ";}
                  void N ( ) {cout <<"Hello"; }
            protected :
                  void  O ( ) {cout <<"salam"; }
          };
```

```
class  B : public  A { public :
                                    void M ( ) {N ( );}
                                    void N ( ) { O ( ); }
                                    void L ( )  {M ( ); }
                          };
main ( )
{  A    p ;                                  Hello salam salam salam خروجي
   B   q ;
   p.N ( ) ;
   q.M ( );
   q.N ( ) ;
   q.L ( ) ; }
```

توضیح : کلاس B وارث کلاس A و در حقیقت وارث O,N,M است. اما در بدنهٔ کلاس B متد های N و M بازنویسی شده اند پس در کلاس B اگر نامی از N وM بیاید N وM محلی مد نظر است. اما اگر نامی از O ببریم چون خود کلاس B ، متد O را ندارد پس متد O پدرش را فراخوانی می کند.

مثال ۴. بازنویسی متدهای به ارث برده شده (نکته ۸) :

```
class A { public :
                void s ( ) {cout <<" 1" ; }
            private :
                int a;
                void sss ( ) {cout <<" 3"; }
            protected:
                void ss ( ) {cout <<" 2"; }
          };
class B :  public A {     public :
                                void sss ( ) {cout <<" 5"; }
                                void s ( )   {ss ( ); }
                          };
                                                                خروجي:  5 2
main ()
{
 B    q ;
 q . s ( );
 q.ss ( );   *
 q.sss ( );
}
```

کلاس A دارای 3 متد SSS() و SS() و S() است و کلاس B که وارث کلاس A است نیز هر سه متد را به ارث برده است اما متدهای SSS() و S() را دوباره نویسی کرده یعنی دیگر دسترسی به دو متد SSS() و S() کلاس A نخواهد داشت.

خط * اشتباه است به دلیل دسترسی به متد SS() که در حقیقت protected است.

مثال ۵. حفظ حوزه در کلاس مشتق (نکته ۴)

```
class A {  public :
                    void s ( ) {cout <<" 1"; }
                    int a ;
             protected :
                    void ss ( ) {cout << "5"; }
           };
class B :  public A { public :
                    void sss ( ) {cout <<" 2"; }
                        };
main(  )
{
 A     p ;
 B     q ;
 p.s ( );
 q.sss ( ) ;
 q.s ( ) ;
 q.a = 3;    //1
 q.ss ( ) ;   //2
}
```

خروجی
1 2 1

درست است که کلاس B به علت وراثت از کلاس A ، متد ss را به ارث برده است اما قبلاً گفتیم اگر متد یا صفتی در کلاس پایه protected باشد در کلاس مشتق (فرزند) هم protected خواهد بود در نتیجه دسترسی شیء q که از جنس کلاس B است به متد ss ای که protected است در تابع main غیر مجاز است. پس خط ۲ غلط است.

نکته : اگر در کلاس A متغیر a بصورت private یا protected تعریف می شد خط ۱ نیز غلط می شد به دلیل دسترسی به یک عضو غیر public در تابع main .

مثال ۶. (جمع بندی مطالب) خطوط غلط برنامه زیر کدامند ؟ و خروجی در صورت حذف آن خطوط؟

```
class A  {          int a ;
             public :

                    void s ( ) {cout << "1" ; }
                                                  protected:

                    void ss ( ) {cout <<" 2" ; }
            private :
                    void sss ( ) {cout <<" 3" ; }
            };
class B :  public  A {    public :
                                  void sss ( ) {cout << a; }      // 1
                                  void sss ( ) {cout <<"2" ; }    // 2
                                  void ss ( ) {sss ( ) ; }        // 3
                                  void s ( ) {ss ( ); }           // 4
                        };
main ( ) {
```

```
A   p ;
B   q ;
p.ss ( ) ;      // 5
q.ss ( );       // 6
p.sss ( );      // 7
q.sss ( );      // 8
q.s ( );        // 9
}
```

خروجی
222

خطوط غلط
۱و ۵ و۷

خط 1 به علت دسترسی به عضو private a غلط است.
خط 5 به علت فراخوانی متد protected ss غلط است.
خط 7 به علت فراخوانی متد private sss غلط است.
اگر خط شماره 4 را comment کنیم به علت نبود متد s در کلاس B ، این متد از کلاس A فراخوانی می شود.

سازنده ها و مخرب و رابطه آنها با وراثت

قبلاً گفتیم که سازنده ها ومخرب ها به ارث برده نمی شوند.
قانون فراخوانی سازنده ها و مخرب ها در ارث بری به شکل زیر است :

۱- ابتدا سازنده کلاس پایه اجرا شده و سپس سازنده کلاس مشتق

۲- ابتدا مخرب کلاس مشتق فراخوانی شده و سپس مخرب کلاس پایه (برعکس سازنده)

اگر شیء را از کلاس مشتق تعریف کنیم قبل از اجرای دستورات سازنده اش ، سازنده کلاس پایه را فراخوانی می کند که این کار یا به صورت صریح (با دادن پارامتر) و یا به صورت ضمنی انجام می شود و اگر کلاس پایه خودش وارث کلاس پایه دیگری بود بصورت سلسله مراتبی سازنده ها سازنده قبلی را فرا می خوانند تا به کلاس پایه اولی برسند.
C++ سازنده کلاس مشتق را ملزم می کند که سازنده کلاس پایه را فراخوانی کند. اگر خودمان سازنده کلاس پایه را فراخوانی کنیم که هیچ وگرنه خودبخود فراخوانی می گردد.
در توابع مخرب ابتدا مخرب مشتق و سپس مخرب پایه فراخوانی می شود اما شیء مشتق از بین نمی رود تا اینکه آخرین مخرب اجرا گردد سپس از حافظه پاک می شود.

شکل کلی:

```
class A {  public :
                A ( ) { }
               ~A ( ) { }
        } ;
class B : public A { public :
                        B ( ) { }
                       ~B ( ) { }
                    };
```

```
Main ( )
{
}
```

هنگام تعریف شیء از جنس کلاس مشتق ، ابتدا سازنده پدر فراخوانی شده سپس سازنده فرزند.
هنگام مرگ یک شی از جنس کلاس مشتق ، ابتدا مخرب فرزند فراخوانی می شود سپس مخرب پدر.

مثال : خروجی برنامه زیر را مشخص کنید .

```
class  A { public :
                A ( ) {cout <<" S1"; }
               ~A ( ) {cout << " M1" ; }
          };
class  B : public A { public :
                        B ( ) { cout << " S2" ; }
                       ~B ( ) {cout << " M2" ; }
                       };
main ( )
{ B   q ;
  A   P ;
  B   S ;                    S1  S2   S1  S1  S2  M2  M1  M1   M2  M1      خروجی :
}
```

نکته: اگر در تابعی شئ ای متولد شود و پس از آن شیء دوم متولد شود موقع خروج از آن تابع ابتدا شیء دوم می میرد و سپس شی اول می میرد یعنی ساختار پشته ای.

مثال : خروجی برنامه زیر را مشخص کنید .

```
int  k =1 ;
class  A { public:
             A ( ) { k ++ ;}
          };
class  B :  public   A { public :
                          B ( ) { k =k + 2;}
                         ~B ( ) {k--; }
                      };
void f() {
           A  x ;
           B  J , L ;      خروجی :  6
          }
main ( ) { f ( ) ;
           cout<< k;
}
```

در تابع f یک شی ء از کلاس A بنام x و دو شیء از کلاس B بنام j , L تعریف شده اند. ترتیب تولد آنها به همین صورت است.
اما موقع خروج از تابع f مرگ اشیاء تعریف شده به این ترتیب است که ابتدا L سپس j و در نهایت x می میرد.
در نتیجه با توجه به مطالبی که قبلاً گفته شد با دقت در فراخوانی سازنده ها و مخرب های کلاس پایه A و مشتق B خروجی برابر 6 است.

فصل سیزدهم – مباحث متفرقه شئ گرایی

متدهای const

اگر بخواهیم متدی را تعریف کنیم که نتواند شیء را تغییر دهد آن را const می کنیم در این صورت در آن متد صفات کلاس را نمی توان دستکاری کرد و فقط می توان به آنها دسترسی داشت و مقدار آنها را نمی توان تغییر داد (Read only).

نکته : سازنده و مخرب چون کارشان تغییر مقدار است پس نمی توان آنها را const کرد.

مثال : پس از comment کردن خط [های] اشتباه ، خروجی را مشخص کنید.

```
class   T { public :
        T ( int x = 0)
                {a = x ; }                                    خروجی   1 0
        void print ( ) const {
                    a++ ;         //*
                      cout << a ;
                                }
        private : int a ;
};

main ( )
{ T   K ;          //1
  T    P (1) ;     //2
  K. print ( ) ;   //3
  P. print ( ) ;    //4
}
```

خط * به این علت باید comment شود که دارد عضو a را تغییر می دهد و این کار در متدهای const ممنوع است.

خط 1 شی ای بنام K تعریف کرده و چون جلویش پارامتر نگذاشته ایم سازنده با مقدار پیش فرض x = 0 اجرا می شود و مقدار متغیر a را در شی K برابر صفر می کند.

خط 2 شی ای بنام P تعریف کرده و پارامتر یک را نیز همان جا به سازنده ارسال می کند در نتیجه مقدار متغیر a در شی P برابر یک است.

اشاره گر **this** (اشاره گر به کلاس)

اگر متدی در داخل خودش متغیری همنام با یکی از صفات کلاس داشته باشد در داخل متد به آن صفت دسترسی نخواهیم نداشت مگر اینکه از اشاره گر به شیء استفاده کنیم.

هر شیء ازاشاره گری بنام this به آدرس خودش دسترسی دارد.

اشاره گرthis قسمتی از خود شیء نیست اما به عنوان آرگومان ضمنی به تمام متدهای غیر استاتیک کلاس ارسال می شود.

لغت this یکی از کلمات رزرو شده C ++ است . برای دسترسی به صفتی با استفاده از اشاره گر this به یکی از دو روش زیر عمل می کنیم:

this->*نام صفت*
(*this).*نام صفت*

مثال : خروجی برنامه زیر را مشخص کنید :

```
//In the name of Allah
# include <iostream.h>
class  Test {                                                    خروجی :
            public :                                             22121
               Test (int = 0) ;            //1
               void print ( ) const ;    //2
            private :
                int   x ;
           };
Test :: Test (int  a) {x =a ;}
void Test :: print ( ) const {
                                int  x= 1 ;
                                x ++;
                                cout <<x <<this -> x << (*this) . x; }
void main ( )
{Test  T (21) ;
  T. print ( ) ;
}
```

1. تابع سازنده Test در خط1 فقط اعلان شده است و در خارج از کلاس تعریف شده و x را مقدار دهی می کند.
2. متد *print* در خط 2 به صورت ثابت اعلان شده است و در بیرون کلاس تعریف شده است.

نکته مهم اینکه متد print داخلش یک x از جنس int تعریف کرده است پس در این متد هر جا اسم *x* خالی بیاید منظور x متد print است نه *x* کلاس *Test*.

اعضای استاتیک کلاس

هر شی ای که از جنس کلاس تعریف کنیم تمام اعضاء را دارا می باشد و همه آنها مختص به خودش است اما اگر بخواهیم یکی از اعضای این کلاس (صفت یا متد) ، برای تمام اشیاء تعریف شده یکسان باشد آن را به صورت استاتیک تعریف می کنیم یعنی این صفت برای تمام اشیاء مشترک است.

این کار مانند استفاده از یک متغیر سراسری است.

مقدار اولیه یک عضو استاتیک صفر است.

مثال : فرض کنید در یک بازی کامپیوتری چند جنگنده داریم . اگر تعداد این جنگنده ها زیر 5 تا باشد قدرتشان 2 و اگر بالای 5 تا باشند قدرتشان مضاعف می گردد. هر جنگنده می تواند بمیرد یا زنده شود. هر جنگنده هنگام تولد بایستی بقیه جنگنده های زنده را بشمارد تا قدرتشان محاسبه گردد. به همین صورت موقع مرگ هر جنگنده نیز باید محاسبات اجرا شود . در اینجا می توان یک عضو کلاس جنگنده را استاتیک تعریف کرد تا هر وقت که جنگنده ای متولد می شود یا می میرد این عضو کم یا زیاد شود نه اینکه هر جنگنده به صورت مجزا محاسبه کند.

به یک صفت استاتیک می توان حتی وقتی هیچ شئ ای از جنس آن کلاس تعریف نشده باشد هم دسترسی یافت. تابعی که استاتیک تعریف شده اشاره گر this ندارد چون صفات و متدهای استاتیک مستقل از هر شیء در کلاس وجود دارند.

مثال : خروجی برنامه زیر را مشخص کنید.

```
class   A {  public:
                    static   int  x ;
              };
int   A :: x = 6 ;   //1
main ( )
{ cout <<A :: x ;
A    P ;
P . x = 3 ;
cout << P. x ;
A    q;
cout <<q.x;   //*
}
```

خروجی :
6 3 3

در خط 1 عضو x از کلاس A که استاتیک است مقدار دهی اولیه شده است بدون آنکه شی ای تعریف شود. در خط * به عضو x از شیء q دسترسی یافته ایم و مقدار آن را چاپ کرده ایم. این مقدار همان مقداری است که آخرین بار وارد x شده است.

فضای نام Namespaces

اگر برنامه ای کوچک باشد نام متغیرها و کلاس ها در ذهنمان می ماند اما با بزرگتر شدن سایز برنامه به خاطر سپردن این نامها سخت شده و همچنین نام های تکراری زیاد خواهند شد و این باعث برخورد (conflict) زیاد می شود.
برای رفع این مشکل کلاس ها و کدها را داخل فضایی به نام namespace ذخیره کرده و موقع استفاده از کلاس ها نام namespace را می آوریم.
نحوه نوشتن و استفاده از namespace ها بصورت زیر است:

```
namespace A {
                class  X { . . .
                          } ;
               } ;

namespace B {
               class X { . . . .
                         } ;
                } ;
```

حال برای دسترسی به اعضای namespace به صورت زیر عمل می کنیم

A : : X یا
B : : X

برای اینکه نخواهیم مرتباً نام namespace را بیاوریم می توان قبل از استفاده بالای برنامه نوشت:

using A ;
using namespace A ; یا در لینوکس

و در اینصورت هر کلاسی که استفاده شود از فضای نام A محسوب می شود

فصل چهاردهم – درآمدی بر زبان C#

در سال ۲۰۰۱ همراه تکنولوژی .Net ارائه شد اما زبانی مانند VB , VC++ از قبل وجود داشتند و بعداً یکی از اعضای تکنولوژی .Net شدند.

تکنولوژی(چهار چوب) دات نت (Net framework.)

یک چهارچوب ارائه شده توسط مایکروسافت که برنامه نویسی را آسان کرده است به این صورت که مجموعه بزرگی از کلاس ها، انواع داده ای، بانکهای اطلاعاتی، ساختار شبکه ای، XML ، وب سرویس ها و ... در کنار هم قرار دارند و برنامه نویس به دلخواه خود یکی از زبانهای برنامه نویسی درون بسته .Net را انتخاب کرده و کدش را به آن زبان می نویسد. این برنامه ها حتی می توانند با یکدیگر ارتباط داشته باشند مثلاً یک تابع نوشته شده به زبان VB.Net را در C#.net فراخوانی کنیم.
تعدادی از زبانهای برنامه نویسی ارائه شده در .Net نسخه 4 که همراه Visual studio 2010 ارائه شده است عبارتند از
VF #, VB , VC + + , VC #
همچنین در چهارچوب .Net می توان صفحات Asp ایجاد کرده که کد پشت این صفحات می تواند به زبان VB یا C # باشد.

دو نوع برنامه با C # می توان نوشت:

۱) Console Application

در این نوع برنامه خروجی روی خط فرمان می آید و بیشتر برای برنامه های سیستمی بکار می رود.

مثال : دستور زیر را در زیر قسمت Main می نویسیم

```
Console.writeline (“salam”);
```

برای اجرا از منوی debug گزینه start debugging (یا کلید F5) را می زنیم یا گزینه start without debuging را می زنیم .

خروجی : salam

نکته : دستور console.Write مکان نما را به خط بعدی نمی برد اما دستور console.WriteLine این کار را انجام می دهد و کاراکترهای کنترلی را می توان در این متد به کار برد.

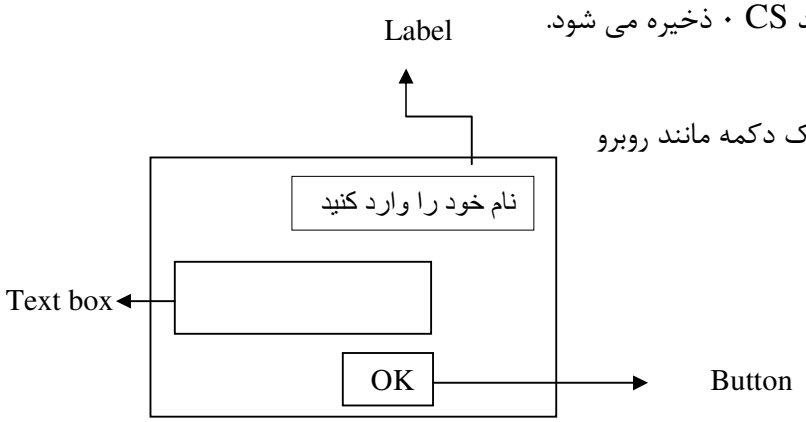
۲) Windows form Application

برنامه هایی که در آن فرم دکمه و دیگر object (اشیاء) بصری وجود دارد در این دسته قرار می گیرند پروژه ی برنامه ای که در .Net می سازیم تحت یک نام کلی با پسوند SLN . ذخیره شده که در این پروژه می توان کدها فرم ها و ... را جا داد.

Solution و محتویاتش در داخل پنجره Solution Explorer قابل مشاهده است.
کدهای نوشته شده به زبان #C در فایلی با پسوند CS ۰ ذخیره می شود.

ساخت برنامه های بصری :

مثال: روی فرم یک text box و یک label و یک دکمه مانند روبرو قرار دهید.



و کد زیر را برای دکمه بنویسید:

```
MessageBox.Show("Hello " + textBox1.Text);
```

سپس کلید F5 یا Ctrl+F5 را بزنید و اجرای برنامه را ببینید.

مانند C++ در C # نیز می توان متغیر تعریف کرد که محدوده این متغیرها مقداری با C++ تفاوت دارد و نیز کلمات کلیدی در C# با حرف کوچک نوشته می شوند و در کامپایلر 2010 به رنگ آبی در می آیند.
عملگرها نیز مانند C++ هستند حتی عملگرهای پیش افزایشی و پیش کاهشی نیز قابل استفاده هستند و عملگرهای منطقی مانند not و or و and مشابه با C + + کار می کنند.
در C # نیز می توان تابع تعریف کرد به همان صورتی که در C + + داشتیم یعنی به چهار صورت (با ورودی با خروجی) (با ورودی بدون خروجی) (بدون ورودی با خروجی) (بدون ورودی بدون خروجی)

تبدیل انواع

در ویژوال بیسیک برای تبدیل یک رشته به یک مقدار int از تابعی بنام val یا تابعی بنام cint استفاده می شد و برعکس برای تبدیل از int به رشته از تابعی به نام cstr استفاده می شد.
در C # به جای لفظ تابع از لفظ متد استفاده می کنیم.
برای تبدیل یک رشته به int از متد int32.parse و int.parse استفاده می شود و برای تبدیل از int به رشته از متد بدون ورودی Tostring استفاده می شود.
هر کلاسی در Net۰ متدی بنام Tostring دارد می توان گفت متد Tostring از class اصلی system به ارث برده شده است.
نکته : در C # نوع داده ای string هم داریم.

پردازش استثنا ها Exception Handling

هر زبان برنامه نویسی بایستی مکانیزم هایی جهت مقابله با Error ها داشته باشد. مثلاً اگر دستوری داشتیم که قرار است فایلی را باز کند اما در حین کار این فایل وجود نداشته باشد به یک Error برخورد کرده ایم و بایستی مدیریت شود یا هنگام تبدیل رشته ای به عدد اگر مثلاً رشته این باشد "12C" این تبدیل موفقیت آمیز نخواهد بود و نیز اگر کاربرتان داده ای را نادرست وارد کند نبایستی کل برنامه بهم بریزد و باعث خروج از کل برنامه شود.

در ویژوال بیسیک از دستور on Error go to استفاده می شد اما در C + + و C # و java مکانیزی به نام بلوک های try و catch داریم. به این ترتیب که دستوراتی که ممکن است ایجاد Exception کند را در بلوک try نوشته و بلافاصله بعد از بلوک try دستورات مدیریت Exception را در بلوک catch می نویسیم. بصورت زیر :

```
try {
دستورات
}
catch {
دستورات مدیریت خطا
}
```

اما اگر دستوری را در بلوک try ننویسیم و ایجاد خطا کند برنامه با مشکل برخورد می کند. حتی می توان چندین بلوک catch برای یک try نوشت در این صورت با توجه Error رخ داده شده catch مناسب انتخاب می شود.

مثال : فرمی با یک Text box ساخته و یک دکمه در کنارش قرار می دهیم در کد دکمه می نویسیم :

```
try {
 int a = int.parse (textBox1.Text);
}
catch {
   MessageBox.Show("please Enter True number");
}
```

تابع int.parse یک رشته را به عدد تبدیل می کند و اگر تبدیل موفقیت آمیز نبود یک Exception تولید می کند .در مثال فوق اگر تبدیل رشته به عدد درست صورت نگیرد دستورات بلوک catch اجرا خواهند شد.

نکته : تقسیم بر صفر تولید استثنا می کند.

پیوست ۱ برخی تفاوت‌های c با ++c

1. دستور printf در زبان c معادل دستور cout است. در این دستور برای چاپ مقادیر متغیرها بایستی نوع آن متغیرها را نیز مشخص کنیم که %f برای float، %d، برای صحیح، %s، برای رشته، %g، برای اعشاری بکار می‌روند. \t و \n نیز همان کاربرد زبان ++c را دارند.

مثال)

در ++c:

```
int a=3;
float b=4.5;
cout<<"the result is:\t”<<a<<b;
```

در c:

```
int a=3;
float b=4.5;
printf("the result is:\t%d%f",a,b);
```

۲. دستور scanf در زبان c معادل دستور cin در ++c است. در این دستور نیز مانند دستور printf بایستی نوع متغیرها را مشخص کنیم. قبل از نام متغیر کاراکتر & می‌آید . اما برای گرفتن یک رشته نیازی به کاراکتر & نیست.

مثال)

در زبان ++c:

```
int a;
float b;
cin>>a>>b;
```

در زبان c:

```
int a;
float b;
scanf("%d%f,&a,&b);
```

هر دو تابع printf و scanf در کتابخانه‌ی stdio.h قرار دارند.

۳. در زبان c چیزی بنام کلاس و شی گرایی نداریم.

۴. در زبان c به جای دستور new از تابع malloc و به جای دستور delete از تابع free استفاده می کنیم.
هر دو تابع مذکور در کتابخانه stdlib.h قرار دارند.

۵. عبارت first-> info که در زبان c برای اشاره به لیست پیوندی بکار می‌رود معادلش در ++c برابر است با (*first).info زیرا عملگر . اولویت بالاتری نسبت به عملگر * دارد واگر پرانتز نگذاریم معادل *(first.info) می‌شود که اشتباه است.

۶. در زبان c کتابخانه توابع فایلی stdio.h است ولی در ++c کتابخانه توابع فایلی fstream.h است.

پیوست ۲ ERROR ها

خطاها به دو نوع کلی تقسیم می شوند:
۱)زمان اجرا: مثل تقسیم بر ۰، و یا داده ی نادرست.
۲)خطاهای زمان کامپایل: اگرspell کلمه را اشتباه بنویسیم یا syntax یک دستور را رعایت نکنیم و ....
نکته: اگر به خطا برخوردید آن خط و خط قبل را مورد بررسی قرار دهید.

:warning
بعضی از برنامه ها ممکن است **error** یا **warning** داشته باشند.اگر برنامه **error** داشته باشد اجرا نمی شود،اما اگر **warning** داشته باشد اجرا می شود و به شما تذکر می دهد که برنامه ممکن است درست جواب ندهد یا بعدا به مشکل برخورد کند.
تذکر: برای توقف اجرای برنامه از کلید ترکیبی **ctrl+break** استفاده می شود.

انواع error:

undefined symbol	متغیر تعریف نشده یا کتابخانه نوشته نشده، یا اسم تابعی اشتباه نوشته شده.
unable to open file	نام کتابخانه اشتباه نوشته شده است.
compouand statement missing	تعداد } و { برابر نیست.
statement missing	یعنی در خط جاری یا خط بالا ; فراموش شده
expected"کاراکتر"	یعنی کاراکتری که داخل" " آمده را در جایی فراموش کرده ایم.
syntax error	از لحاظ نحوی غلط وجود دارد. مثلا else قبل از if آمده است.
lvalue requaired	سمت چپ عبارت Lvalue نیست یا بجای ==،= گذاشته اید.

Lvalue: یعنی مقدار سمت چپ. یعنی چیزی که می‌تواند در سمت چپ یک عبارت قرار گیرد و مقداری بگیرد. مانند متغیر، عنصر آرایه و...
در دستورات زیر آنهایی که زیرشان خط کشیده شده lvalue هستند.

```
a=b*c;
b=3*f;
b=a[i]/2;
```