

موسسه آموزش عالی سناباد گلپهار

برنامه سازی پیشرفته

تهیه کننده: فرزاد زندی

فهرست مطالب

- ① فصل اول : مقدمات زبان C++
- ② فصل دوم : ساختار های تصمیم گیری و تکرار
- ③ فصل سوم : سایر ساختار های تکرار
- ④ فصل چهارم : اعداد تصادفی
- ⑤ فصل پنجم : آرایه ها
- ⑥ فصل ششم : توابع
- ⑦ فصل هفتم : ساختارها و اشاره گرها
- ⑧ فصل هشتم : برنامه نویسی شی گرا

فصل اول

مقدمات C++



فهرست مطالب فصل اول

1. تاریخچه مختصر
2. قانون نامگذاری شناسه ها
3. متغیر ها
4. اعلان متغیر
5. تخصیص مقادیر به متغیر
6. داده های از نوع کرکتر
7. کرکتر های مخصوص
8. رشته ها
9. نمایش مقادیر داده ها
10. دریافت مقادیر
11. عملگر انتساب
12. عملگر های محاسباتی
13. عملگر های افزایش و کاهش
14. عملگر sizeof
15. عملگر های جایگزینی محاسباتی
16. اولویت عملگرها
17. توضیحات (Comments)
18. توابع کتابخانه
19. برنامه در C++



تاریخچه مختصر C++

این زبان در اوائل دهه ۱۹۸۰ توسط Bjarne stroustrup در آزمایشگاه بل طراحی شده. این زبان عملاً توسعه یافته زبان برنامه نویسی C می باشد که امکان نوشتن برنامه‌های ساخت یافته شیء گرا را می دهد.



قانون نامگذاری شناسه‌ها

(1) حروف کوچک و بزرگ در نامگذاری شناسه‌ها متفاوت می‌باشند.

بنابراین xy ، xY ، XY ، Xy چهار شناسه متفاوت از نظر C++ می‌باشد.



قانون نامگذاری شناسه‌ها

(۲) در نامگذاری شناسه‌ها از حروف الفباء، ارقام و زیر خط (**underscore**) استفاده می‌شود و حداکثر طول شناسه ۳۱ می‌باشد و شناسه بایستی با یک رقم شروع نگردد.



قانون نامگذاری شناسه‌ها

۳) برای نامگذاری شناسه‌ها از کلمات کلیدی نبایستی استفاده نمود. در زیر بعضی از کلمات کلیدی داده شده است.



And	Sizeof	then	xor	Template
Float	False	Friend	While	continue
extern	Private	Switch	Default	Const
delete	typedef	if	this	Virtual

لیست کامل کلمات کلیدی



متغیرها

متغیر، مکانی در حافظه اصلی کامپیوتر می باشد که در آنجا یک مقدار را می توان ذخیره و در برنامه از آن استفاده نمود. قانون نامگذاری متغیرها همان قانون نامگذاری شناسه ها می باشد.

در اسلاید بعد به انواع داده ها اشاره می شود.

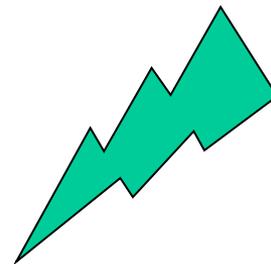


انواع داده ها

نوع داده	مقادیر	حافظه لازم
int	-32768 تا 32767	۲ بایت
unsigned int	0 تا 65535	۲ بایت
long int	-2147483648 تا 2147483647	۴ بایت
unsigned long int	0 تا 4294967295	۴ بایت
char	یک کارکتر	۱ بایت
unsigned char	-128 تا 127	۱ بایت
float	1.2e-38 تا 3.4e38	۴ بایت
double	2.2e-308 تا 1.8e308	۸ بایت



اعلان متغیرها



قبل از آنکه در برنامه به متغیرها مقداری تخصیص داده شود و از آنها استفاده گردد بایستی آنها را در برنامه اعلان نمود.

در اسلاید بعد مثال هایی از اعلان متغیر ذکر شده است.



چند مثال از اعلان متغیر ها :

✓ برای اعلان متغیر x از نوع int :

```
int x;
```

✓ برای اعلان متغیرهای p و q را از نوع float که هر کدام چهار بایت از حافظه را اشغال می کنند :

```
float p , q ;
```

✓ برای اعلان متغیر next از نوع کرکتر که می توان یکی از ۲۵۶ کرکتر را به آن تخصیص داد و یک بایت را اشغال می کند.

```
char next ;
```



تخصیص مقادیر به متغیرها

با استفاده از عملگر = می توان به متغیرها مقدار اولیه تخصیص نمود.

در اسلاید بعد مثال هایی از اعلان متغیر ذکر شده است.



مثال :

```
int x=26;
```

✓ در دستورالعمل
X را از نوع int با مقدار اولیه 26 اعلان نموده .

```
long a=67000 , b=260;
```

✓ در دستورالعمل
متغیرهای a و b را از نوع long int تعریف نموده با مقادیر بترتیب
260 و 67000 .



داده‌های از نوع کرکتر

برای نمایش داده‌های از نوع char در حافظه کامپیوتر از جدول ASCII استفاده می‌شود. جدول اسکی به هر یک از ۲۵۶ کرکتر یک عدد منحصر بفرد بین ۰ تا ۲۵۵ تخصیص می‌دهد.



کرکترهای مخصوص



کامپیلر C++ بعضی از کرکترهای مخصوص که در برنامه می‌توان از آنها برای فرمت بندی استفاده کرد را تشخیص می‌دهد. تعدادی از این کرکترهای مخصوص به همراه کاربرد آنها در اسلاید بعد آورده شده است .



گرکترهای مخصوص

\n	Newline
\t	Tab
\b	Backspace
\a	Beep sound
\"	Double quote
\'	Single quote
\0	Null character
\?	Question mark
\\	Back slash

بعنوان مثال از گرکتر \a می توان برای ایجاد صدای beep استفاده نمود.

```
char x = '\a' ;
```



رشته‌ها

رشته یا string عبارتست از دنباله‌ای از کرکترها که بین " " قرار داده می‌شود. در حافظه کامپیوتر انتهای رشته‌ها بوسیله ۱۰ ختم می‌گردد.

در اسلاید بعد به دو مثال دقت نمایید.



مثال ۱ :

"BOOK STORE" یک رشته ده کرکتری می باشد که با توجه به کرکتر \0 که به انتهای آن در حافظه اضافه می شود جمعاً یازده بایت را اشغال می کند.



مثال ۲ :

دقت نمایید که "w" یک رشته می باشد که دو بایت از حافظه را اشغال می کند در حالیکه 'w' یک کرکتر می باشد که یک بایت از حافظه را اشغال می نماید.



نمایش مقادیر داده‌ها

برای نمایش داده‌ها بر روی صفحه مانیتور از `cout` که بدنبال آن عملگر درج یعنی `>>` قید شده باشد استفاده می‌گردد. بایستی توجه داشت که دو کرکتر `>` پشت سر هم توسط `C++` بصورت یک کرکتر تلقی می‌گردد.



مثال :

✓ برای نمایش پیام good morning بر روی صفحه نمایش :

```
cout << "good morning";
```

✓ برای نمایش مقدار متغیر X بر روی صفحه نمایش :

```
cout << x ;
```



دریافت مقادیر متغیرها

به منظور دریافت مقادیر برای متغیرها در ضمن اجرای برنامه از صفحه کلید، از cin که بدنبال آن عملگر استخراج یعنی << قید شده باشد می توان استفاده نمود.



مثال :

```
int x;  
cout << "Enter a number:" ;  
cin >> x;
```



عملگر انتساب

عملگر انتساب = می باشد که باعث می گردد
مقدار عبارت در طرف راست این عملگر ارزیابی
شده و در متغیر طرف چپ آن قرار گیرد.



مثال :

```
x=a+b;  
x=35 ;  
x=y=z=26 ;
```

از عملگرهای انتساب چندگانه نیز می‌توان استفاده نمود. که مقدار سه متغیر Z و Y و X برابر با 26 میشود.



عملگرهای محاسباتی

در C++ پنج عملگر محاسباتی وجود دارد که عبارتند از :

جمع	+
تفریق	-
ضرب	*
تقسیم	/
باقیمانده	%

این عملگرها دو تائی می‌باشند زیرا روی دو عملوند عمل می‌نمایند. از طرف دیگر عملگرهای + و - را می‌توان بعنوان عملگرهای یکتائی نیز در نظر گرفت.



مثال ۱ :

در حالتی که هر دو عملوند عملگرهای $+$ ، $-$ ، $*$ ، $/$ ، $\%$ از نوع صحیح باشد نتیجه عمل از نوع صحیح می‌باشد.

عبارت	نتیجه
$5 + 2$	7
$5 * 2$	10
$5 - 2$	3
$5 \% 2$	1
$5 / 2$	2



مثال ۲ :

در صورتیکه حداقل یکی از عملوندهای عملگرهای / ، * ، - ، + از نوع اعشاری باشد نتیجه عمل از نوع اعشاری می باشد.

عبارت	نتیجه
$5.0 + 2$	7.0
$5 * 2.0$	10.0
$5.0 / 2$	2.5
$5.0 - 2$	3.0
$5.0 / 2.0$	2.5



عملگرهای افزایش و کاهش

در C++ ، افزایش یک واحد به مقدار یک متغیر از نوع صحیح را افزایش و بطور مشابه کاهش یک واحد از مقدار یک متغیر از نوع صحیح را کاهش می‌نامند..



عملگرهای افزایش و کاهش

عملگر کاهش را -- و عملگر افزایش را ++ نمایش می‌دهند. چون عملگرهای ++ و -- فقط روی یک عملوند اثر دارند این دو عملگر نیز جزء عملگرهای یکتائی می‌باشند.



مثال :

سه دستور العمل :

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

معادل می باشند و بطریق مشابه سه دستور العمل زیر نیز معادل می باشند.

```
--y;  
y=y-1;  
y--;
```



از عملگرهای ++ و -- می توان بدو صورت پیشوندی و پسوندی استفاده نمود.
در دستورالعمل های پیچیده عملگر پیشوندی قبل از انتساب ارزیابی میشود و عملگر
پسوندی بعد از انتساب ارزیابی می شود.



مثال :

```
int x=5;  
y=++x * 2;
```

y=12

پس از اجرای دستورالعملهای فوق :

```
int x=5;  
y=x++ * 2;
```

y=12

پس از اجرای دستورالعملهای فوق :



عملگر sizeof

sizeof از عملگرهای یکتائی می باشد و مشخص کننده تعداد بایت هائی است که یک نوع داده اشغال می کند.

مثال :

```
int x;  
cout << sizeof x ;
```

مقدار ۲ نمایش داده می شود .

```
cout << sizeof(float) ;
```

مقدار ۴ نمایش داده می شود.



عملگرهای جایگزینی محاسباتی

برای ساده تر نوشتن عبارتها در C++ ، می توان از عملگرهای جایگزینی محاسباتی استفاده نمود.

`+=` `-=` `*=` `/=` `%=`



اولویت عملگرها

ارزیابی مقدار یک عبارت ریاضی براساس جدول اولویت عملگرها انجام می‌گردد. در ذیل جدول اولویت عملگرها براساس بترتیب از بیشترین اولویت به کمترین اولویت داده شده است.

()	پرانتزها	چپ به راست
- + -- ++ sizeof	عملگرهای یکتایی	راست به چپ
* / %	عملگرهای ضرب و تقسیم و باقیمانده	چپ به راست
+ -	عملگرهای جمع و تفریق	چپ به راست
<< >>	عملگرهای درج و استخراج	چپ به راست
= += -= *= /= %=	عملگرهای جایگزینی و انتساب	راست به چپ



مثال ۱ :

$$(5+2) * (6+2*2)/2$$

با توجه به جدول اولویت عملگرها داریم که

$$7 * (6+2*2)/2$$

$$7 * (6+4)/2$$

$$7 * 10 / 2$$

$$70 / 2$$

$$35$$



مثال ۲ :

```
int a=6 , b=2, c=8, d=12;  
d=a++ * b/c ++;  
cout << d << c << b << a;
```

خروجی :

1 9 2 7



توضیحات (Comments)

توضیحات در برنامه باعث خوانائی بیشتر و درک بهتر برنامه میشود. بنابراین توصیه بر آن است که حتی الامکان در برنامه‌ها از توضیحات استفاده نمائیم. در **C++**، توضیحات بدو صورت انجام می‌گیرد که در اسلایدهای بعد به آن اشاره شده است.



توضیحات (Comments)

الف: این نوع توضیح بوسیله // انجام می شود. که کامپیوتر هر چیزی را که بعد از // قرار داده شود تا انتهای آن خط اغماض می نماید.

مثال :

```
c=a+b;//c is equal to sum of a and b
```

ب: توضیح نوع دوم با /* شروع شده و به */ ختم می شود و هر چیزی که بین /* و */ قرار گیرد اغماض می نماید.

مثال :

```
/* this is a program  
to calculatate sum of  
n integer numbers */
```



توابع کتابخانه

زبان **C++** مجهز به تعدادی توابع کتابخانه می باشد. بعنوان مثال تعدادی توابع کتابخانه برای عملیات ورودی و خروجی وجود دارند. معمولاً توابع کتابخانه مشابه ، بصورت برنامه های هدف (برنامه ترجمه شده بزبان ماشین) در قالب فایل های کتابخانه دسته بندی و مورد استفاده قرار می گیرند. این فایلها را فایل های header می نامند و دارای پسوند **.h** می باشند.



نحوه استفاده از توابع کتابخانه ای

برای استفاده از توابع کتابخانه خاصی بایستی نام فایل header آنرا در ابتدای برنامه در دستور `#include` قرار دهیم.

`#include < header فایل اسم >`



<u>تابع</u>	<u>نوع</u>	<u>شرح</u>	<u>فایل هیدر</u>
abs(i)	int	قدر مطلق i	stdlib.h
cos(d)	double	کسینوس d	math.h
exp(d)	double	e^x	math.h
log(d)	double	$\log_e d$	math.h
log10(d)	double	$\text{Log}_{10} d$	math.h
sin(d)	double	سینوس d	math.h
sqrt(d)	double	جذر d	math.h
strlen(s)	int	تعداد کرکترهای رشته S	string.h
tan(d)	double	تانژانت d	math.h
toascii(c)	int	کداسکی کرکتر c	stdlib.h
tolower(c)	int	تبدیل به حروف کوچک	stdlib.h
toupper(c)	int	تبدیل به حرف بزرگ	stdlib.h



C++ در برنامه

اکنون با توجه به مطالب گفته شده قادر خواهیم بود که تعدادی برنامه ساده و کوچک به زبان C++ بنویسیم. برای نوشتن برنامه بایستی دستورالعملها را در تابع () `main` قرار دهیم و برای اینکار می توان به یکی از دو طریقی که در اسلایدهای بعد آمده است ، عمل نمود.



روش اول :

```
#include < >
int main( )
{
    دستورالعمل 1 ;
    دستورالعمل 2 ;
    .
    .
    .
    دستورالعمل n ;
    return 0 ;
}
```

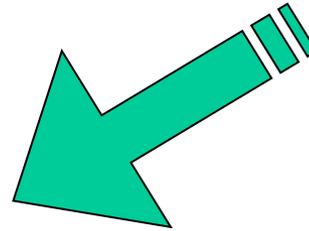


روش دوم :

```
#include < >  
void main( )  
{  
  دستورالعمل 1 ;  
  دستورالعمل 2 ;  
  .  
  .  
  .  
  دستورالعمل n ;  
}
```



برنامه ای که پیغام **C++ is an object oriented language** را روی صفحه مانیتور نمایش می دهد.

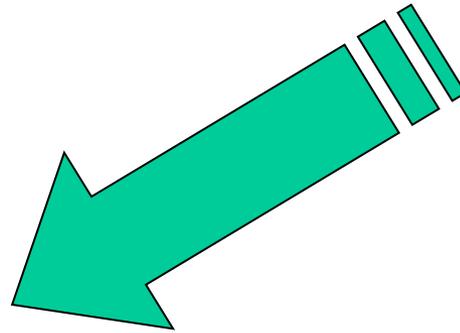


```
#include <iostream.h>
int main()
{
cout <<"C++ is an object oriented language \n" ;
return 0 ;
}
```



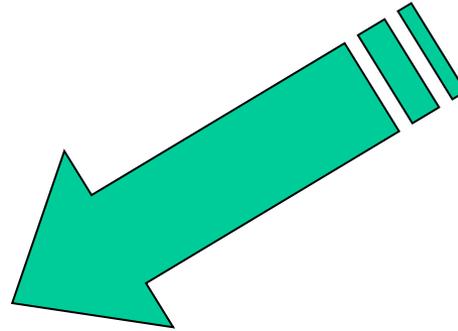
برنامه زیر یک حرف انگلیسی کوچک را گرفته به حرف بزرگ تبدیل می نماید.

```
#include <iostream.h>
#include <stdlib. h>
int main( )
{
    char c1 , c2;
    cout << "Enter a lowercase letter:"
    cin >> c1;
    c2 = toupper(c1);
    cout << c2 << endl;
    return 0; }
```



دو عدد از نوع اعشاری را گرفته مجموع و حاصلضرب آنها را محاسبه و نمایش می دهد.

```
#include <iostream.h>
int main()
{
float    x,y,s,p ;
cin >> x >> y ;
s= x+y ;
p=x*y;
cout << s <<endl << p;
return 0 ;
}
```



فصل دوم

ساختارهای تصمیم‌گیری و تکرار



فهرست مطالب فصل دوم

1. عملگر های رابطه ای
2. عملگر شرطی
3. دستورالعمل شرطی
4. عملگر کاما
5. عملگر های منطقی
6. دستورالعمل For



عملگرهای رابطه ای

از این عملگرها برای تعیین اینکه آیا دو عدد با هم معادلند یا یکی از دیگری بزرگتر یا کوچکتر می باشد استفاده می گردد. عملگرهای رابطه ای عبارتند از:

$==$	مساوی
$!=$	مخالف
$>$	بزرگتر
$>=$	بزرگتر یا مساوی
$<$	کوچکتر
$<=$	کوچکتر یا مساوی



عملگر شرطی

شکل کلی عملگر شرطی بصورت زیر می باشد:

```
expression _ test ? expression _ true : expression _ false
```

عملگر شرطی تنها عملگری در C++ می باشد که دارای سه عملوند می باشد.



مثال ۱ :

```
int x=10,y=20,b;  
b=(x>y) ? x : y ;
```

این دو دستور العمل باعث میشوند که ماکزیمم مقادیر x و y در b قرار بگیرد.

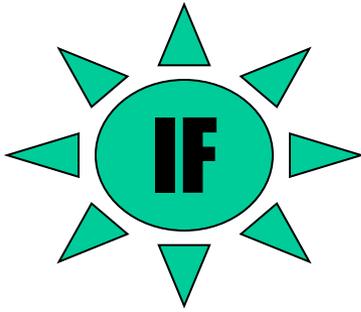
مثال ۲ :

```
x>=10 ? cout << "passed" : cout << "failed" ;
```

اگر مقدار x بزرگتر یا مساوی ده باشد رشته `passed` در غیر اینصورت رشته `failed` نمایش داده میشود.

رشته





دستور العمل شرطی

توسط این دستور شرطی را تست نموده و بسته به آنکه شرط درست یا غلط باشد عکس العمل خاصی را نشان دهیم.

```
if( عبارت )  
{  
    دستورالعمل 1  
    .  
    دستورالعمل n  
}  
else  
{  
    دستورالعمل 1  
    .  
    دستورالعمل n  
}
```



مثال ١ :

```
if(x != y)
{
cout << x ;
++ x ;
}
else
{
cout << y ;
-- y ;
}
```



مثال ۲:

برنامه زیر یک عدد اعشاری را از ورودی گرفته جذر آنرا محاسبه می نماید.

```
#include <iostream.h>
#include <math . h>
int main( )
{
float x,s;
cin >> x ;
if( x < 0 )
cout << " x is negative" << endl ;
else
{
s = sqrt(x) ;
cout << s << endl ;
}
return 0;
}
```



عملگر کاما

تعدادی عبارت را می توان با کاما بهم متصل نمود و تشکیل یک عبارت پیچیده تری را داد. این عبارتها به ترتیب از چپ به راست ارزیابی شده و مقدار عبارت معادل عبارت n می باشد.



(عبارت n , ..., عبارت 3, عبارت 2, عبارت 1)



مثال :

اگر داشته باشیم `int a=2 , b=4 , c=5 ;` عبارت زیر را در نظر بگیرید:

`(++ a , a+b, ++ c, c+b)`

مقدار عبارت برابر است با `b+c` که معادل 10 می باشد.



عملگرهای منطقی

با استفاده از عملگرهای منطقی می توان شرطهای ترکیبی در برنامه ایجاد نمود.
عملگرهای منطقی عبارتست از :

AND

OR

NOT

که در C++ به ترتیب بصورت زیر نشان داده میشود.

&&

||

!



جدول درستی سه عملگر شرطی

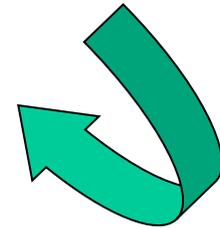
And



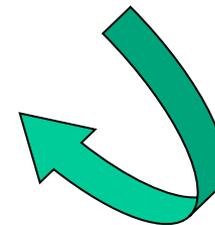
a	b	a && b
true	true	True
true	false	False
false	true	False
false	false	False

a	b	a b
true	true	True
true	false	True
false	true	True
false	false	False

Or



Not

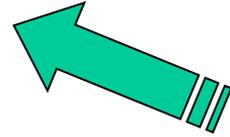


a	!a
true	False
false	True



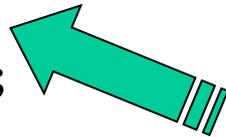
چند مثال :

```
if ((x == 5) || (y != 0))  
    cout << x << endl ;
```



اگر x برابر با 5 یا y مخالف صفر باشد مقدار x نمایش داده شود .

```
if(x)  
    x = 0 ;
```



اگر مقدار x مخالف صفر باشد، آنگاه x برابر با صفر شود .



برنامه زیر طول سه پاره‌خط را از ورودی گرفته مشخص می‌نماید که آیا تشکیل یک مثلث میدهد یا خیر؟

```
#include <iostream.h>
int main()
{
float a, b, c;
cout << "Enter three real numbers" << endl ;
cin >> a >> b >> c; //
if(( a < b + c) &&(b < a+c) &&(c < a+b))
cout << "It is a triangle" ;
else
cout << "Not a triangle" ;
return 0 ;
}
```



دستور العمل For

از دستور العمل **for** برای تکرار دستور العملها استفاده میشود. شکل کلی دستور **for** بصورت زیر می باشد:

(عبارت 3 ; عبارت 2 ; عبارت 1) **for**

```
{  
  دستور العمل 1 ;  
  دستور العمل 2 ;  
  .  
  .  
  .  
  دستور العمل n ;  
}
```



برنامه زیر عدد صحیح و مثبت n را از ورودی گرفته فاکتوریل آنرا محاسبه و نمایش می دهد.

```
#include <iostream.h>
int main( )
{
int n, i ;
long fact = 1 ;
cout << "Enter a positive integer number";
cin >> n;
for( i=1; i<=n; ++i) fact *= i;
cout << fact << endl;
return 0 ;
}
```



برنامه زیر مجموع اعداد صحیح و متوالی بین ۱ تا n را محاسبه نموده و نمایش می دهد.

```
#include <iostream.h>
int main( )
{
int n, i=1 ;
long s = 0 ;
cin >> n ;
for(; i<=n; i++) s += i;
cout << s ;
return 0 ; }
```



برنامه زیر ارقام 0 تا 9 را نمایش می دهد.

```
#include <iostream.h>
int main( )
{
int j=0 ;
for( ; j <= 9 ; ) cout << j++ << endl;
return 0 ;
}
```



برنامه زیر کلیه اعداد سه رقمی که با ارقام 1 ، 2 ، 3 ایجاد می شوند را نمایش می دهد.

```
#include <iostream.h>
int main( )
{
int i,j,k,n;
for(i=1; i<=3; ++i)
for(j=1; j<=3; ++j)
for(k=1; k<=3; ++k)
{
n=i*100 + j*10+k;
cout << n << '\n' ;
}
return 0 ;
}
```



فصل سوم

سایر ساختارهای تکرار



فهرست مطالب فصل سوم

1. دستورالعمل while
2. دستورالعمل do while
3. دستورالعمل break
4. دستورالعمل continue
5. دستورالعمل switch
6. تابع cin.get()
7. عملگر static_cast<>()
8. جدول اولویت عملگرها



دستور العمل while

از این دستور العمل مانند دستور العمل for برای تکرار یک دستور العمل ساده یا ترکیبی استفاده می گردد. شکل کلی این دستور العمل بصورت زیر می باشد.

```
while( شرط )
```

```
{
```

```
  دستور العمل ۱ ;
```

```
  دستور العمل ۲ ;
```

```
  .
```

```
  .
```

```
  دستور العمل n ;
```

```
}
```



تفاوت دستورهای for و while

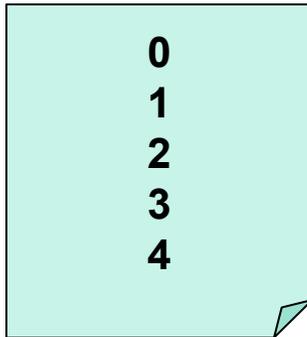
دستورالعمل **for** زمانی استفاده میشود که تعداد دفعات تکرار از قبل مشخص و معین باشد. در صورتیکه تعداد دفعات تکرار مشخص نباشد بایستی از دستورالعمل **while** استفاده نمود.



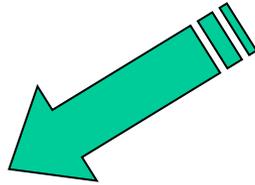
مثال :

```
int x=0  
while(x<5)  
cout << x ++<< endl;
```

با اجرای قطعه برنامه فوق مقادیر زیر نمایش داده میشود :



0
1
2
3
4



برنامه فوق n مقدار از نوع اعشاری را گرفته میانگین آنها را محاسبه و در متغیر avg قرار می دهد.

```
#include <iostream.h>
int main( )
{
int count = 0 , n;
float x, sum = 0 , avg ;
cin >> n ; /* تعداد مقادیر ورودی n*/
while(count < n){
cin >> x ;
sum += x ;
++ count ; }
avg = sum / n ;
cout << avg << endl;
return 0 ; }
```



دستور العمل do while

این دستور العمل نیز برای تکرار یک دستور العمل ساده یا ترکیبی استفاده می شود. شکل کلی این دستور العمل بصورت زیر می باشد.

```
do
{
  دستور العمل ۱ ;
  دستور العمل ۲ ;
  .
  .
  دستور العمل n ;
} while( شرط );
```



تفاوت دستورهای while و do while

در دستورالعمل **while** ابتدا مقدار شرط ارزیابی شده اما در دستورالعمل **do while** ابتدا دستورالعمل اجرا شده سپس مقدار شرط ارزیابی می‌گردد. بنابراین دستورالعمل **do while** حداقل یک بار انجام میشود.



مثال :

```
#include <iostream.h>
int main( )
{
int count = 0;
do
cout << count ++<<endl ;
while(count <= 9);
return 0 ; }
```

ارقام 0 تا 9 را روی ده خط نمایش می دهد



دستور العمل break

این دستور العمل باعث توقف دستور العملهای تکرار (for , while ,do while) شده و کنترل به خارج از این دستور العملها منتقل می نماید.

Break



مثال ١ :

```
#include <iostream.h>
int main( )
{
float x, s=0.0 ;
cin >> x ;
while(x <= 1000.0) {
if(x < 0.0){
cout << "Error-Negative Value" ;
break;
}
s += x ;
cin >> x ;}
cout << s << endl ;
return 0 ; }
```



```
#include <iostream.h>
int main( )
{
int count = 0 ;
while( 1 )
{
count ++ ;
if(count > 10 )
break ;
}
cout << "counter : " << count << "\n";
return 0 ;
}
```



```
#include <iostream.h>
void main( )
{
int count;
float x, sum = 0;
cin >> x ;
for(count = 1; x < 1000 . 0; ++ count )
{
cin >> x ;
if(x < 0.0) {
cout << "Error – Negative value " <<endl;
break ;
}
sum += x ; }
cout << sum << '\n' ; }
```



```
#include <iostream.h>
int main( )
{
float x , sum = 0.0 ;
do
{
cin >> x ;
if(x < 0.0)
{
cout << "Error – Negative Value" << endl ;
break ;
}
sum += x ;
} while(x <= 1000.0);
cout << sum << endl ;
return 0 ; }
```



دستورالعمل continue

از دستورالعمل **continue** می توان در دستورالعملهای تکرار **for** ، **while** ، **do while** استفاده نمود. این دستورالعمل باعث می شود که کنترل بابتدای دستورالعملهای تکرار منتقل گردد.

Continue



مثال ١:

```
#include <iostream.h>
int main( )
{
float x, sum = 0.0 ;
Do
{
cin >> x ;
if(x < 0 . 0)
{
cout << "Error" << endl ;
continue ;
}
sum += x ;
} while(x <= 1000.0 );
cout << sum ;
return 0 ; }
```



```
#include <iostream.h>
int main( )
{
int n , navg = 0 ;
float x, avg, sum = 0 ;
cin >> n ; /* عبارت n * /
از تعداد اعداد ورودی
for(int count = 1 ; count <=n; ++ count )
{
cin >> x ;
if(x < 0 ) continue ;
sum += x ;
++ navg ;
}
avg = sum / navg;
cout << avg << endl ;
return 0 ;
}
```



دستور العمل switch

همانطور که می دانید از دستور العمل شرطی (if else) می توان بصورت تودرتو استفاده نمود ولی از طرفی اگر عمق استفاده تو در تو از این دستور العمل زیاد گردد، درک آنها مشکل میشود. برای حل این مشکل ++C ، دستور العمل switch که عملاً یک دستور العمل چند انتخابی می باشد را ارائه نموده است.

switch
case



شكل کلی دستور العمل Switch

switch(عبارت)

```
{  
case valueone : statement;  
                break;  
case valuetwo: statement;  
                break;  
:  
case valuen : statement;  
                break;  
default: statement ;  
}
```



مثال ١ :

```
#include <iostream.h>
void main( )
{
  unsigned int n ;
  cin >> n;
  switch(n)
  {
    case 0:
      cout << "ZERO" << endl ;
      break;
    case 1:
      cout << "one" << endl ;
      break ;
    case 2:
      cout << "two" << endl ;
      break;
    default :
      cout << "default" << endl;
      /* end of switch statement */
  }
}
```



مثال ٢ :

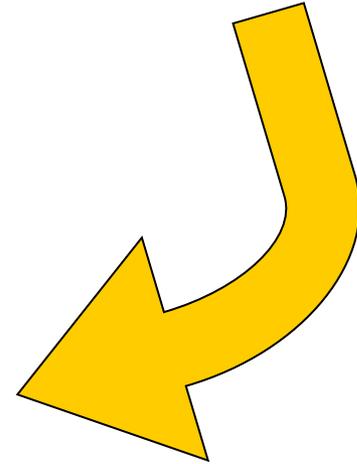
```
#include <iostream.h>
void main( )
{
    unsigned int n;
    cin >> n ;
    switch(n) {
    case 0 :
    case 1:
    case 2:
        cout << "Less Than Three" << endl;
        break;
    case 3:
        cout << "Equal To Three" << endl ;
        break;
    default:
        cout << "Greater Than Three" << endl;
    }
}
```



تابع cin.get() :

این تابع یک کرکتر را از صفحه
کلید می‌گیرد. برای استفاده از این
تابع در ابتدای برنامه بایستی داشته
باشیم:

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



قطعه برنامه ذیل یک کرکتر را از صفحه کلید گرفته و نمایش می دهد.

```
char x;  
x = cin.get( );  
cout << x ;
```



برنامه ذیل یک سطر متن انگلیسی که به CTRL Z ختم میشود را گرفته دقیقاً نمایش می دهد.

```
#include <iostream.h>
int main( )
{
char x;
while((x = cin.get( ) !=EOF)
cout << x ;
return 0 ;
}
```

EOF به معنی End of File می باشد که در `iostream.h` تعریف شده و مقدار آن برابر با ۱- می باشد. مقدار آن در سیستم عامل DOS عبارتست از `ctrl z`.



در قطعه برنامه ذیل از تابع `cin.get()` و دستور `switch` استفاده شده است.

```
char    x;  
x = cin.get( );  
switch(x) {  
case ' r ' :  
case ' R ' :  
        cout << "RED" << "\n" ;  
        break ;  
case ' b ' :  
case ' B ' :  
        cout << "BLUE" << endl ;  
        break ;  
case ' y ' :  
case ' Y ' :  
        cout << "YELLOW" << endl;  
}
```

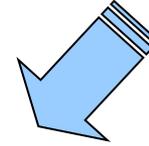


برنامه ذیل یک سطر متن انگلیسی را گرفته کرکتهای خالی (blank) آنرا حذف نموده و نمایش میدهد.

```
#include <iostream.h>
int main( )
{
char next;
while((next = cin.get( ) ) !=EOF)
if(next != ' ')
cout << next ;
return 0 ;
}
```



عملگر static_cast



از این عملگر برای تبدیل موقت یک نوع data به نوع دیگر استفاده می‌شود. این عملگر یک عملگر یکتائی می‌باشد.



مثال ۱:

```
int x = 25 ;  
float y ;  
y = static_cast  
< float >(x) ;
```

مقدار x موقتاً بصورت اعشاری در می آید و در نتیجه مقدار y برابر با 25.0 می شود. بایستی توجه داشت که نوع متغیر x عوض نمی شود بلکه موقتاً مقدار آن بصورت اعشاری در آمده است.



```
float x = 14.75 ;  
cout <<  
static_cast < int  
>(x) << endl;  
cout << x ;
```

ابتدا مقدار ۱۴ نمایش داده میشود و سپس مقدار ۱۴.۷۵ نمایش داده میشود.



جدول اولویت عملگرها

()	چپ به راست
Static_cast < >() ++ -- + - sizeof	راست به چپ
* / %	چپ به راست
+ -	چپ به راست
<< >>	چپ به راست
< <= > >=	چپ به راست
== !=	چپ به راست
? :	راست به چپ
= += -= *= /= %=	راست به چپ
,	چپ به راست



فصل چهارم

اعداد تصادفی



فهرست مطالب فصل چهارم

1. تولید اعداد تصادفی
2. تعریف نوع داده (typedef)
3. داده های از نوع شمارشی
4. فرمت های مختلفه مقادیر خروجی



اعداد تصادفی

مقادیر تصادفی یا شانسی در اکثر برنامه‌های کاربردی در زمینه شبیه‌سازی و بازیهای کامپیوتری نقش مهمی را ایفا می‌نمایند. برای ایجاد یک عدد تصادفی صحیح بین ۰ و ۳۲۷۶۷ بایستی از تابع rand () استفاده نمائیم.

rand()



برنامه زیر 10 عدد تصادفی بین 0 و 32767 را ایجاد می نماید.

```
#include <stdlib.h>
#include <iostream.h>
int main( )
{
for(int j=1; j<=10; ++j)
cout << rand( ) << '\n' ;
return 0 ;
}
```



اگر برنامه فوق را چندبار اجرا نمائیم جواب یکسانی را از کامپیوتر می گیریم.
برای تصادفی کردن اعداد می بایستی از تابع `srand()` استفاده نمائیم.
این تابع به یک آرگومان صحیح از نوع `unsigned` نیاز دارد.
به این آرگومان `seed` گفته می شود.

در اسلاید بعد برنامه قبلی را با تابع `srand()` نوشته ایم.



برنامه زیر 10 عدد تصادفی بین 0 و 32767 را ایجاد می نماید. (srand())

```
#include      <stdlib.h>
#include      <iostream.h>
int main( )
{
  unsigned   seed;
  cout << "Enter seed value : " ;
  cin >> seed ;
  srand(seed);
  for(int j=1; j<=10; ++j)
  cout << rand( ) << '\n ' ;
  return 0 ;
}
```



برنامه زیر نتیجه پرتاب دو تاس را نمایش می دهد.

```
#include <iostream.h>
#include <stdlib.h>
int main( )
{
    unsigned seed, d1, d2;
    cout << "Enter seed: " ;
    cin >> seed ;
    srand(seed) ;
    d1= 1+rand( )% 6 ;
    d2= 1+rand( )% 6 ;
    cout << d1 << " " << d2 ;
    return 0 ;
}
```



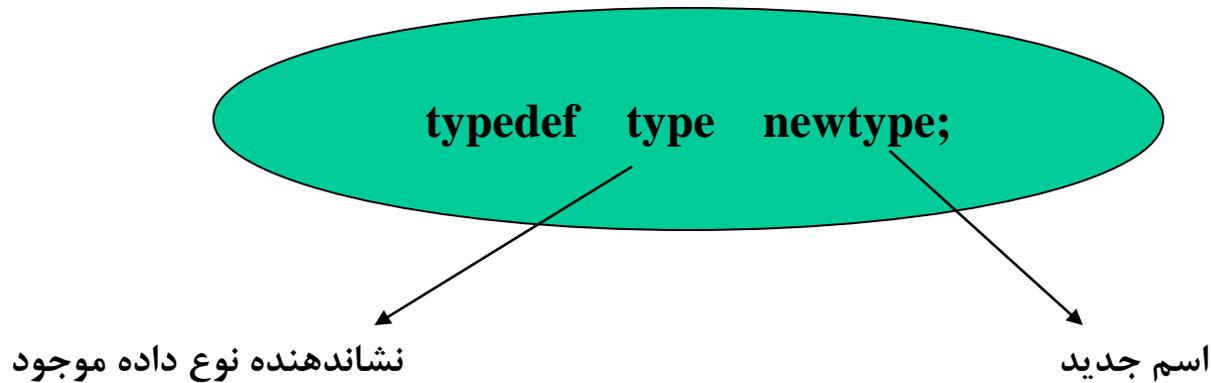
برنامه زیر 10 اعداد شانسی بین 0 و 1 را نمایش می دهد.

```
#include <stdlib.h>
#include <iostream.h>
int main( )
{
    unsigned seed ;
    cout << "Enter seed: " ;
    cin >> seed ;
    srand(seed) ;
    for(int i=1; i<=10; ++i)
    cout << rand( ) / 32768.0 << endl ;
    return 0 ;
}
```



تعریف نوع داده (typedef)

از `typedef` می‌توان برای تعریف نوع داده‌های جدید که معادل نوع داده‌های موجود باشد استفاده نمود. شکل کلی عبارتست از:



مثال :

```
typedef int integer;
```

حال می توان x و y را بصورت زیر تعریف نمود :

```
integer x,y;
```



داده‌های از نوع شمارشی

بمنظور معرفی داده‌های از نوع شمارشی از کلمه `enum` استفاده می‌گردد.

مثال :

```
0 1 2 3 4  
↑ ↑ ↑ ↑ ↑  
enum color {red, blue, green, yellow, brown} ;
```

`color` یک نوع داده شمارشی می‌باشد.

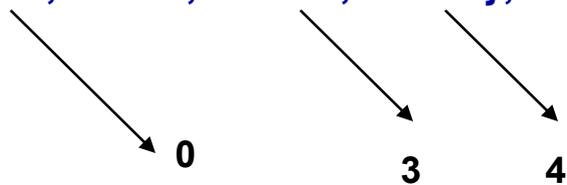


```
enum status {married, divorced, widow, single};  
status a ;  
a= single ;
```

```
enum days {sat, sun, mon, tue, wed, thr,  
           fri};
```

```
enum bread {lavash, fantezi, taftoon, barbari};
```

```
enum color { yellow, red=2, brown, white };  
color x=brown;
```



توجه :

بایستی در نظر داشت که داده‌های از نوع شمارشی در عملیات ورودی و خروجی شرکت نمی‌نمایند. بعبارت دیگر مقادیر داده‌های از نوع شمارشی بایستی در برنامه تعیین نمود. دستورالعمل‌های ورودی و خروجی مانند cin و cout در مورد داده‌های شمارشی نمی‌توان استفاده نمود.



فرمت‌های مختلفه مقادیر خروجی

مقدار X بطور غیر علمی با نقطه اعشار ثابت نمایش داده می شود.

```
include <iomanip.h>
```

```
double x=1050 ;
```

```
cout << setiosflags(ios :: fixed | ios :: showpoint ) << setw(23)  
<< setprecision(2) << x << endl ;
```

مقدار X با طول میدان ۲۳ نمایش داده می شود.

مقدار X با دو رقم اعشار نمایش داده می شود.

بنابراین مقدار X بصورت زیر نمایش داده می شود :

1050.00 شانزده ستون خالی



فصل پنجم

آرایه ها



فهرست مطالب فصل پنجم

1. آرایه یک بعدی
2. آرایه دو بعدی (ماتریس ها)



آرایه یک بعدی

آرایه یک فضای پیوسته از حافظه اصلی کامپیوتر می باشد که می تواند چندین مقدا را در خود جای دهد.

کلیه عناصر یک آرایه از یک نوع می باشند.

عناصر آرایه بوسیله اندیس آنها مشخص می شوند.

در ++C ، اندیس آرایه از صفر شروع می شود.



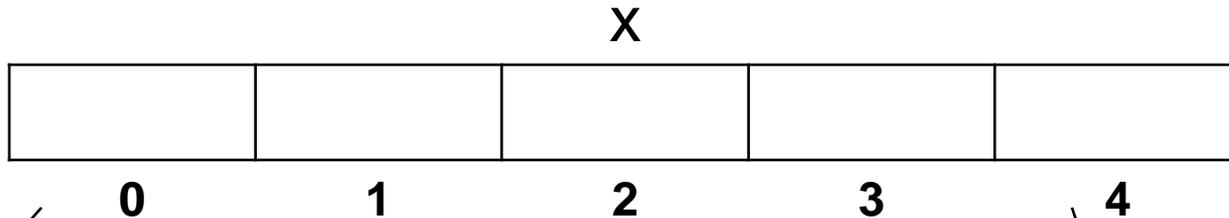
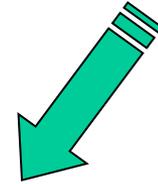
کاربرد آرایه ها

آرایه‌ها در برنامه‌نویسی در مواردی کاربرد دارند که
بخواهیم اطلاعات و داده‌ها را در طول اجرای
برنامه حفظ نمائیم.



آرایه یک بعدی از نوع *int*

```
int x[5];
```



0

1

2

3

4

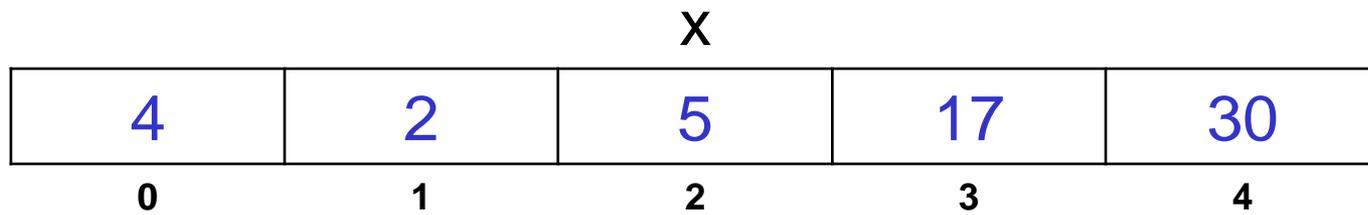
اولین عنصر $x[0]$

پنجمین عنصر $x[4]$



تخصیص مقادیر اولیه به عناصر آرایه :

```
int x[5]= {4, 2, 5, 17, 30};
```



دریافت مقادیر عناصر آرایه :

```
int x[5];  
for(int i=0; i<=4; ++i)  
cin >> x[ i ] ;
```

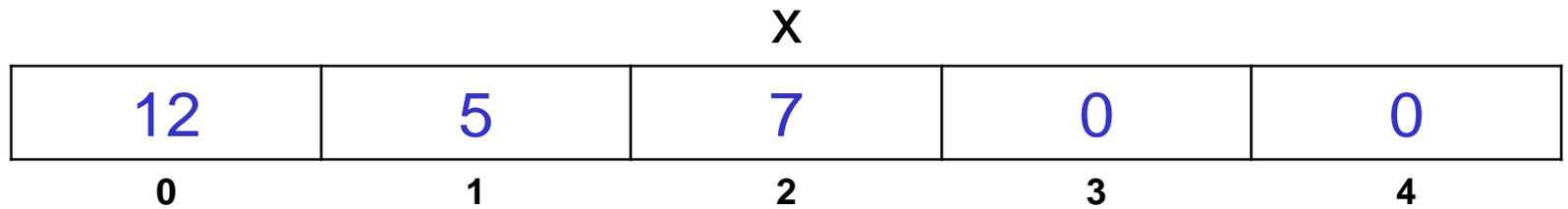
نمایش مقادیر عناصر آرایه :

```
for(int i=0; i<=5; ++i) cout << x[ i ] ;
```



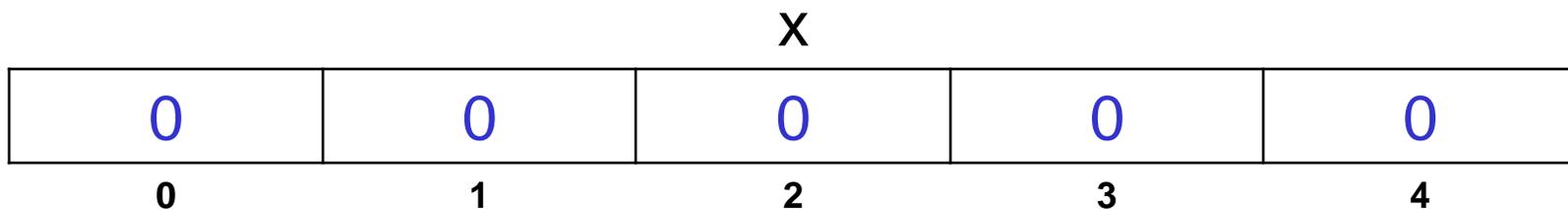
اگر تعداد مقادیر اولیه کمتر از تعداد عضوهای آرایه باشد عضوهای باقیمانده بطور اتوماتیک، مقدار اولیه صفر می‌گیرند.

```
int x[5] = {12, 5, 7};
```



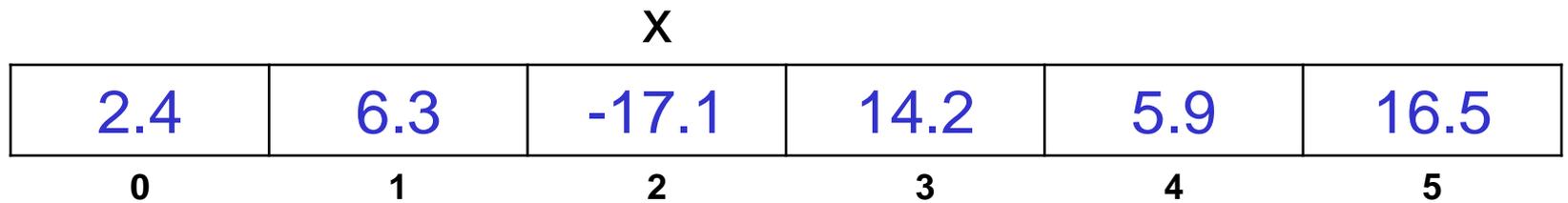
بایستی توجه داشت که آرایه‌ها به صورت ضمنی مقدار اولیه صفر نمی‌گیرند. برنامه نویس باید به عضو اول آرایه، مقدار اولیه صفر تخصیص دهد تا عضوهای باقی‌مانده بطور اتوماتیک، مقدار اولیه صفر بگیرند.

```
int x[5] = {0} ;
```



دستور زیر یک آرایه یک بعدی شش عنصری از نوع float ایجاد می نماید.

```
float x[ ] = {2.4, 6.3, -17.1, 14.2, 5.9, 16.5} ;
```



برنامه ذیل 100 عدد اعشاری و مثبت را گرفته تشکیل یک آرایه میدهد سپس مجموع عناصر آرایه را مشخص نموده نمایش می دهد.

```
#include <iostream.h>
#include <iomanip.h>
int main( )
{
  const int arrsize = 100 ;
  float x[ arrsize], tot = 0.0 ;
  for(int j=0; j<arrsize; j++)
    cin >> x[ j ];
  for(j=0; j<arrsize; j++)
    cout << setiosflags(ios::fixed ios :: showpoint ) << setw(12) <<
      setprecision(2) << x[ j ] << endl;
  for(j=0; j<arrsize; j++)
    tot += x[ j ] ;
  cout << tot ;
  return 0 ;
}
```



برنامه ذیل 20 عدد اعشاری را گرفته تشکیل یک آرایه داده سپس کوچکترین عنصر آرایه را مشخص و نمایش می دهد.

```
#include <iostream.h>
#include <conio.h>
int main( )
{
float x[20], s;
int j ;
clrscr( ) ;
for(j=0; j<20 ; ++j) cin >> x[ j ];
s = x[0 ] ;
for(j=1; j<20; ++j)
if(x[ j] <s) s = x[ j ];
cout << s << endl;
return 0;
}
```



برنامه زیر 100 عدد اعشاری را گرفته بروش حبابی (Bubble sort) بصورت صعودی مرتب می نماید.

```
#include <iostream.h>
#include <conio.h>
int main ( )
{
float x[100] , temp;
int i,j;
clrscr( );
for(i=0; i<100; ++i) cin >> x[i ];
for(i=0; i<99; i++)
for(j=i+1 ; j<100; j++)
if(x[ j ] < x[i ]){
temp = x[ j ] ;
x[ j ] = x[ i ] ;
x[ i ] = temp ;
}
for(i=0; i<=99; i++)
cout << x[ i ] << endl;
return 0 ;
}
```



آرایه‌های دوبعدی (ماتریس‌ها)

ماتریسها بوسیله آرایه‌های دوبعدی در کامپیوتر نمایش داده میشوند.

```
int a[3][4];
```

	ستون 0	ستون 1	ستون 2	ستون 3
سطر 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
سطر 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
سطر 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]



تخصیص مقادیر اولیه به عناصر آرایه :

```
int a[3][4]={ {1,2,3,4}, {5,6,7,8}, {9,10,11,12} } ;
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12



نکته ۱ :

```
int a[3][4]= { {1}, {2,3} , {4,5,6} } ;
```

	0	1	2	3
0	1	0	0	0
1	2	3	0	0
2	4	5	6	0



نکته ۲ :

```
int a[3][4]= {1, 2, 3, 4,5 } ;
```

	0	1	2	3
0	1	2	3	4
1	5	0	0	0
2	0	0	0	0



نکته ۳ :

در یک آرایهٔ دواندیزی، هر سطر، در حقیقت آرایه‌ای یک اندیزی است. در اعلان آرایه‌های دواندیزی ذکر تعداد ستونها الزامی است.

```
int a[ ][4]={1,2,3,4,5};
```

	0	1	2	3
0	1	2	3	4
1	5	0	0	0



برنامه زیر یک ماتریس 3×4 را گرفته مجموع عناصر آن را مشخص نموده و نمایش می دهد.

```
#include <iostream.h>
#include <conio.h>
int main( )
{
float  x[3][4], total= 0.0;
int  i, j ;
// generate matrix x.
for(i=0; i<3; ++i)
for (j=0; j<3; j++)
cin >> x[ i ][ j ];
// calculate the sum of elements.
for(i=0; i<3; ++i)
for(j=0; j<4; j++)
tot + = x [ i ][ j ];
cout << "total = " << total << endl;
return 0 ;
}
```



فصل نشم

توابع



فهرست مطالب فصل ششم

1. تعریف تابع
2. تابع بازگشتی
3. توابع درون خطی
4. انتقال پارامترها از طریق ارجاع
5. کلاس های حافظه (storage classes)
6. سربارگذاری توابع



تعریف توابع

استفاده از توابع در برنامه‌ها به برنامه‌نویس این امکان را می‌دهد که بتواند برنامه‌های خود را به صورت قطعه قطعه برنامه بنویسد. تا کنون کلیه برنامه‌هایی که نوشته‌ایم فقط از تابع () `main` استفاده نموده‌ایم .



شکل کلی توابع بصورت زیر می‌باشند :

لیست پارامترها جهت انتقال اطلاعات از تابع احضار کننده به تابع فراخوانده شده

نوع مقدار برگشتی

return-value-type function-name (parameter-list)

{

declaration and statements

}

نام تابع

تعریف اعلان‌های تابع و دستورالعمل‌های اجرایی



تابع زیر یک حرف کوچک را به بزرگ تبدیل می نماید.

نوع مقدار برگشتی

پارامتری از نوع char

نام تابع

```
char low_to_up (char c1)
{
char c2;
c2 = (c1>= ' a ' && c1<= ' z ')?(' A ' + c1- ' a '): c1;
return (c2) ;
}
```



برنامه کامل که از تابع قبل جهت تبدیل یک حرف کوچک به بزرگ استفاده می نماید.

```
#include <iostream.h>
char low_to_up(char c1)
{
char c2;
c2=(c1 >= ' a ' && c1 <= ' z ')?(' A ' +c1 -' a ' ) : c1;
return c2;
}
int main( )
{
char x;
x=cin.get( );
cout << low_to_up(x) ;
return 0;
}
```

X

'd'



C1

'd'



C2

'D'

آرگومان



تابع maximum دو مقدار صحیح را گرفته بزرگترین آنها را برمیگرداند.

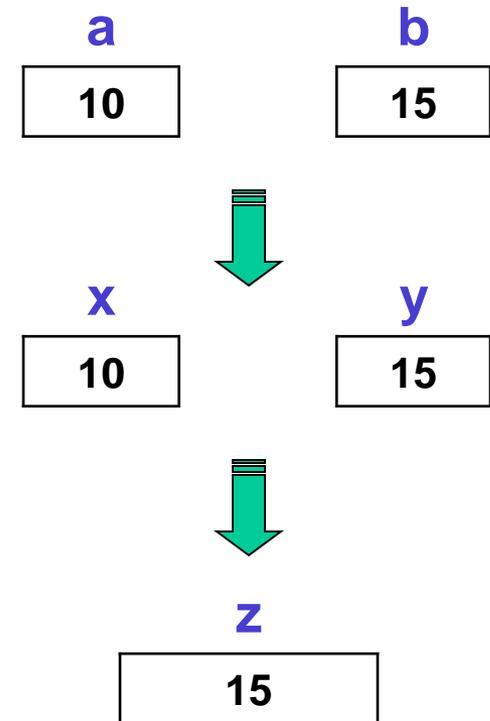
```
int maximum(int x, int y)
{
int z ;
z=(x >= y)? x : y;
return z;
}
```



برنامه کامل که از تابع maximum جهت یافتن ماکزیمم دو مقدار صحیح استفاده می نماید.

```
#include <iostream.h>
int maximum(int x , int y)
{
int z ;
z=(x > y)? x : y ;
return z;
}
int main( )
{
int a, b ;
cin >> a >> b ;
cout << maximum(a,b);
return 0;
}
```

a, b آرگومانهای تابع maximum



زنگنه () :

اسامی پارامترها و آرگومانهای یک تابع
می توانند همنام باشند.



برنامه زیر یک مقدار مثبت را گرفته فاکتوریل آنرا محاسبه نموده نمایش می دهد.

$$x! = 1 * 2 * 3 * 4 * \dots * (x-1) * x$$

```
#include <iostream.h>
long int factorial(int n)
{
    long int prod=1;
    if(n>1)
    for(int i=2; i<=n; ++i)
    prod *=i;
    return(prod);
}
int main( )
{
    int n;
    cin >> n ;
    cout << factorial(n) ;
    return 0 ;
}
```

main در n

3



factorial در n

3

factorial در i

2,3,4

factorial در prod

6



وقتی در تابعی، تابع دیگر احضار می‌گردد
بایستی تعریف تابع احضار شونده قبل از
تعریف تابع احضار کننده در برنامه ظاهر گردد.



اگر بخواهیم در برنامه‌ها ابتدا تابع **main** ظاهر گردد بایستی **prototype** تابع یعنی پیش نمونه تابع که شامل نام تابع، نوع مقدار برگشتی تابع، تعداد پارامترهایی را که تابع انتظار دریافت آنرا دارد و انواع پارامترها و ترتیب قرارگرفتن این پارامترها را به اطلاع کامپایلر برساند.

در اسلاید بعد مثالی در این زمینه آورده شده است.



```
#include <iostream.h>
#include <conio.h>
long int factorial(int); // function prototype
int main( )
{
int n;
cout << "Enter a positive integer" << endl;
cin >> n;
cout << factorial(n) << endl;
return 0 ;
}
long int factorial(int n)
{
long int prod = 1;
if(n>1)
for(int i=2; i<=n; ++i)
prod *= i;
return(prod);
}
```



نکته ۴:

در صورتی که تابع مقداری بر نگرداند نوع مقدار برگشتی تابع را void اعلان می‌کنیم. و در صورتیکه تابع مقداری را دریافت نکند بجای parameter- list از void یا () استفاده می‌گردد.

در اسلاید بعد مثالی در این زمینه آورده شده است.



```
#include <iostream.h>
#include <conio.h>
void maximum(int , int) ;
int main( )
{ int x, y;
clrscr( )
cin >> x >> y;
maximum(x,y);
return 0;
}
```

```
void maximum(int x, int y)
{
int z ;
z=(x>=y) ? x : y ;
cout << "max value \n" << z<< endl;
return ;
}
```

تابع مقداری بر نمی گرداند.



احضار بوسیله مقدار (Call By Value)

```
#include <iostream.h>
int modify(int)
int main( )
{
int a=20;
cout << a << endl;
modify(a) ;
cout << a << endl;
return 0 ;
}
int modify(int a)
{
a *= 2;
cout << a << endl;
return ;
}
```

main در a

20

modify در a

20

modify در a

40

خروجی برنامه :

20

40

20



احضار بوسیله مقدار (Call By Value)

```
#include <iostream.h>
int modify(int)
int main( )
{
int a=20;
cout << a << endl;
modify(a) ;
cout << a << endl;
return 0 ;
}
int modify(int a)
{
a *= 2;
cout << a << endl;
return ;
}
```



main در a

20

modify در a

20

modify در a

40

در این نوع احضار تابع حافظه‌های مورد استفاده آرگومانها و پارامترها از هم متمایزند و هرگونه تغییر در پارامترها باعث تغییر در آرگومانهای متناظر نمی‌گردد.

هر زمان که نوع مقدار برگشتی تابع `int` می‌باشد نیازی به ذکر آن نیست و همچنین نیازی به تعریف پیش نمونه تابع نمی‌باشد.



تابع بازگشتی (recursive functions)

توابع بازگشتی یا recursive توابعی هستند که
وقتی احضار شوند باعث می‌شوند که خود را احضار نمایند.



نحوه محاسبه فاکتوریل از طریق تابع بازگشتی

$$n! = 1 * 2 * 3 * \dots * (n-1) * n$$

$$f(n) = n!$$

$$f(n) = \begin{cases} 1 & \text{اگر } n=0 \\ n * f(n-1) & \text{در غیر اینصورت} \end{cases}$$

$$n! = 1 * 2 * 3 * \dots * (n-2) * (n-1) * n$$

$$n! = (n-1)! * n$$

در اسلاید بعد تابع بازگشتی مورد نظر پیاده سازی شده است.



```
#include <iostream.h>
long int factorial(int) ;
int main( )
{
    int n ;
    cout << " n= " ;
    cin >> n ;
    cout << endl << " factorial = " << factorial(n) << endl;
    return 0 ;
}
long int factorial(int n)
{
    if(n<=1)
        return(1);
    else
        return(n *factorial(n-1) ) ;
}
```



نحوه محاسبه n امین مقدار دنباله فیبناکی از طریق تابع بازگشتی

دنباله فیبناکی : 0 , 1, 1, 2, 3, 5, 8, 13, 21 , 34, ...

$$\text{fib}(n) = \begin{cases} 0 & \text{اگر } n=1 \\ 1 & \text{اگر } n=2 \\ \text{fib}(n-1)+\text{fib}(n-2) & \text{در غیر اینصورت} \end{cases}$$

در اسلاید بعد تابع بازگشتی مورد نظر پیاده سازی شده است.



برنامه‌زیر n امین مقدار دنباله فیبناکی (fibonacci) را مشخص و نمایش می‌دهد.

```
#include <iostream.h>
long int fib(long int); // forward declaration
int main( )
{
    long int r ;
    int n ;
    cout << " Enter an integer value " << endl ;
    cin >> n ;
    r = fib(n) ;
    cout << r << endl ;
    return 0 ;
}
long int fib(long int n)
{
    if(n == 1 || n == 2)
        return 1 ;
    else
        return(fib(n-1) + fib(n-2) ) ;
}
```



برنامه زیر یک خط متن انگلیسی را گرفته آنرا وارون نموده نمایش می دهد.

```
#include <iostream.h>
void reverse(void) ; // forward declaration
int main( )
{
    reverse( ) ;
    return 0 ;
}
void reverse(void)
// read a line of characters and reverse it
{
    char c ;
    if(( c=cin.get( )) != ' \n ') reverse( ) ;
        cout << c ;
    return ;
}
```



استفاده از آرایه‌ها بعنوان پارامتر تابع مجاز است.

در اسلاید بعد به یک مثال توجه نمایید.



در برنامه زیر تابع `modify` آرایه `a` را بعنوان پارامتر می‌گیرد.

```
#include <iostream.h>
void modify(int [ ]); // forward declaration
int main( )
{
int a[5];
for(int j=0; j<=4; ++j)
a[ j ] = j+1 ;
modify(a) ;
for(j=0; j<5; ++j)
cout << a[ j ] << endl ;
return 0 ;
}
void modify(int a[ ]) // function definition
{
for(int j=0; j<5; ++j)
a[ j ] += 2 ;
for(j=0; j<5; ++j)
cout << a[ j ] << endl ;
return ;
}
```

خروجی :

1
2
3
4
5
3
4
5
6
7



نکته :

در صورتیکه آرایه بیش از یک بعد داشته باشد
بعدهای دوم به بعد بایستی در تعریف تابع و پیش
نمونه تابع ذکر گردد.

در اسلاید بعد به یک مثال توجه نمایید.



```

#include <iostream.h>
void printarr(int [ ][ 3 ]);
int main( )
{
    int arr1 [2][3] = { {1,2,3}, {4,5,6} };
    arr2 [2][3]= {1,2,3,4,5};
    arr3 [2][3]={ {1,2}, {4} };
    printarr(arr1);
    cout << endl ;
    printarr(arr2);
    cout << endl ;
    printarr(arr3);
    return 0 ;
}
void printarr(int a[ ][3] )
{
    for(int i=0; i<2; i++)
    {
        for(int j=0; j<3; j++)
            cout << a[ i ][ j ] << '
';
        cout << endl ;
    }
}

```

خروجی :

1	2	3
4	5	6
1	2	3
4	5	0
1	2	0
4	0	0



توابع درون خطی (inline)

کلمه **inline** بدین معنی است که به کامپایلر دستور می‌دهد که یک کپی از دستورات العلمهای تابع در همان جا (در زمان مقتضی) تولید نماید تا از احضار تابع ممانعت بعمل آورد.



اشکال توابع inline

بجای داشتن تنها یک کپی از تابع ، چند کپی از دستورالعملهای تابع در برنامه اضافه می شود که باعث بزرگ شدن اندازه یا طول برنامه می شود. بنابراین از inline برای توابع کوچک استفاده می گردد.



مثالی از توابع درون خطی

```
#include <iostream.h>
inline float cube(const float s) {return s*s*s; }
int main( )
{
float side ;
cin >> side ;
cout << side << cube(side) << endl ;
return 0 ;
}
```



انتقال پارامترها از طریق ارجاع

تاکنون وقتی تابعی را احضار می کردیم یک کپی از مقادیر آرگومانها در پارامترهای متناظر قرار می گرفت . این روش احضار بوسیله مقدار یا **call by value** نامیده شد.
در انتقال پارامترها از طریق ارجاع در حقیقت حافظه مربوط به آرگومانها و پارامترهای متناظر بصورت اشتراکی مورد استفاده قرار می گیرد. این روش **call by reference** نامیده می شود .



انتقال پارامترها از طریق ارجاع

در این روش پارامترهایی که از طریق `call by reference` عمل می‌نمایند در پیش نمونه تابع قبل از نام چنین پارامترهایی از `&` استفاده می‌شود. واضح است که در تعریف تابع نیز بهمین طریق عمل می‌شود.



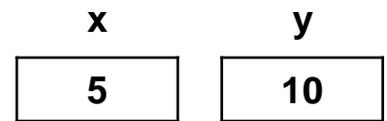
```
#include <iostream.h>
int vfunc(int); // for
void rfunc (int &);
int main( )
```

```
{
    int x=5, y=10;
    cout << x << endl << vfunc(x) << endl << x << endl ;
    cout << y << endl ;
    rfunc(y);
    cout << y << endl ;
    return 0 ;
}
```

```
int vfunc(int a)
{
    return a *= a ;
}
```

```
void rfunc(int &b)
{
    b *= b ;
}
```

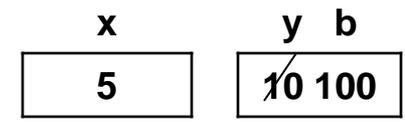
مثال :



خروجی :

5
25
5
10

مقدار آرگومان x تغییر نمی کند.



ادامه خروجی :

100



نکته :

وقتی پارامتری بصورت `call by reference` اعلان می‌گردد این بدان معنی است که با تغییر مقدار این پارامتر در تابع احضار شده مقدار آرگومان متناظر نیز تغییر می‌نماید.



برنامه‌زیر با استفاده از fswap دو مقدار اعشاری را مبادله می‌نماید.

```
#include <iostream.h>
void fswap(float & , float & );
int main( )
{
    float a=5.2, b=4.3;
    cout << a << endl << b ;
    fswap( a , b ) ;
    cout << a << endl << b ;
    return 0 ;
}
void fswap(float &x , float & y)
{
    float t;
    t = x ;
    x = y ;
    y = t ;
}
```



کلاس‌های حافظه (storage classes)

متغیرها بدو طریق متمایز مشخص می‌شوند یکی بوسیله نوع (type) آنها و دیگری بوسیله کلاس حافظه آنها. نوع متغیر قبلاً اشاره شده بعنوان مثال `int` ، `float` ، `double` ، ... ولی کلاس حافظه یک متغیر در مورد طول عمر و وسعت و دامنه متغیر بحث می‌نماید.

در اسلاید بعد به انواع کلاس حافظه می‌پردازیم.



بطور کلی کلاس حافظه متغیرها به چهار دسته تقسیم می گردد :

- .1 automatic
- .2 static
- .3 external
- .4 register



automatic

متغیرهای **automatic** در درون یک تابع تعریف می‌شوند و در تابعی که اعلان می‌شود بصورت متغیرهای محلی برای آن تابع می‌باشند. حافظه تخصیص داده شده به متغیرهای **automatic** پس از اتمام اجرای تابع از بین می‌رود بعبارت دیگر وسعت و دامنه متغیرهای از نوع **automatic** تابعی می‌باشد که متغیر در آن اعلان گردیده است.



static

متغیرهای **static** نیز در درون توابع تعریف میشوند و از نظر وسعت و دامنه شبیه
متغیرهای **automatic** هستند ولی در خاتمه اجرای تابع، حافظه وابسته به این نوع
متغیرها از بین نمی‌رود بلکه برای فراخوانی بعدی تابع باقی می‌ماند.

در اسلاید بعد به یک مثال از کاربرد این نوع کلاس حافظه می‌پردازیم.



مثال :

```
#include <iostream.h>
// program to calculate successive fibonacci numbers
long int fib(int) ;
int main( )
{
int n ;
cout << " how many fibonacci numbers?" ;
cin >> n ;
cout << endl ;
for(int j=1; j<=n; ++j )
cout << j << " " << fib(j) << endl ;
return 0 ;
}
long int fib(int count)
{
static long int t1 = 1, t2=1;
long int t ;
t =(count <3) ?1 : t1 + t2 ;
t2 = t1 ;
t1 = t ;
return(t) ;
}
```



بایستی توجه داشت که اگر در توابع به متغیرهای از نوع static مقدار اولیه تخصیص ندهیم مقدار صفر بصورت اتوماتیک برای آنها در نظر گرفته می شود.

external

متغیرهای از نوع **external** متغیرهائی هستند که در بیرون از توابع اعلان میشوند و وسعت و دامنه فعالیت آنها کلیه توابعی می باشد که در زیر دستور اعلان متغیر قرار دارد.

در اسلاید بعد به یک مثال از کاربرد این نوع کلاس حافظه می پردازیم.



مثال :

```
#include <iostream.h>
int w; // external variable
functa(int x, int y)
{
cout << w ;
w = x + y ;
cout << endl << w << endl;
return x%y ;
}
int main( )
{
int a, b, c, d;
cin >> a >> b ;
c=functa(a, b) ;
d=functa(w, b+1);
cout << endl << c << endl << d << endl << w ;
return 0 ;
}
```

بایستی توجه داشت که اگر در توابع به متغیرهای از نوع external مقدار اولیه تخصیص ندهیم مقدار صفر بصورت اتوماتیک برای آنها در نظر گرفته می شود.



register

وقتی متغیری از نوع register اعلان می شود از کامپیوتر عملاً درخواست می شود که به جای حافظه از یکی از رجیسترهای موجود استفاده نماید.



کاربرد کلاس register

معمولاً از نوع رجیستر برای شاخص‌های دستور تکرار و یا اندیس‌های آرایه‌ها استفاده می‌شود. بایستی توجه داشت که متغیرهای از نوع رجیستر قابل استفاده در دستور cin نمی‌باشند



سربارگذاری توابع (function overloading)

در C++ این امکان وجود دارد که در یک برنامه بتوانیم از چند توابع هم نام استفاده نمائیم مشروط بر این که پارامترهای این توابع متفاوت باشند. (از نظر تعداد پارامتر و یا نوع پارامترها و ترتیب آنها)



مثال :

```
#include <iostream.h>
float addf(float , int);
int addf(int , int);
int main( )
{
int a=5, b=10 ;
float d=14.75 ;
cout << addf(a , b) << endl;
cout << addf(d , b) << endl;
return 0 ;
}
int addf(int x, int y)
{
return x+y ;
}
float addf(float x, int y)
{
return x+y ;
}
```



فصل هفتم

ساختارها و اشاره گرها



فهرست مطالب فصل هفتم

1. ساختارها
2. Union ها
3. اشاره گرها (Pointer)
4. تعریف آرایه
5. آرایه های دو بعدی و اشاره گرها
6. تخصیص حافظه بصورت پویا (عملگر new)
7. رشته ها و توابع مربوطه



ساختارها

ساختارها شبیه آرایه‌ها بوده بدین صورت که یک نوع داده گروهی است که فضای پیوسته از حافظه اصلی را اشغال می‌نماید. اما عناصر ساختار الزاماً از یک نوع نمی‌باشند بلکه اعضای یک ساختار می‌توانند از نوع‌های مختلفه از قبیل **char** ، **float** ، **int** ... باشند.



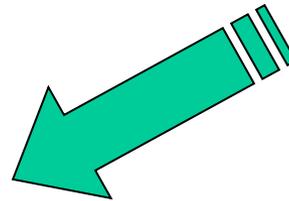
تعريف ساختار

نام ساختار

struct time

```
{  
int hour ; // 0 – 23  
int minute ; // 0 – 59  
int second; //  
};
```

اعضا ساختار



مثال :

```
struct account {  
int acc_no ;  
char acc_type;  
char name[80] ;  
float balance ;  
};
```

ساختار **account** دارای چهار عضو می باشد.
acc_no شماره حساب از نوع **int**
acc_type نوع حساب از نوع **char**
name مشخصات صاحب حساب از نوع رشته **80** کارکتری
balance مانده حساب از نوع **float**



به دو صورت می توان اعلان یک متغیر از نوع ساختار را نمایش داد :

روش اول :

```
struct account {  
int acc_no;  
char acc_type;  
char name[80];  
float balance;  
} cust1, cust2, cust3;
```

روش دوم :

```
struct account {  
int acc_no ;  
char acc_type;  
char name[80];  
float balance;  
};  
account cust1, cust2, cust3;
```



به ساختارها می توان مقدار اولیه نیز تخصیص داد

```
account cust = {4236, 'r', "Nader Naderi" , 7252.5};
```



دسترسی به عناصر یک ساختار

بمنظور دسترسی به عناصر یک ساختار از عملگر . استفاده
می گردد . عملگر . جزء عملگرهای یکتائی می باشد.



مثال :

cust .acc_no = 4236

cust .acc_type = 'r'

cust . name = "Nader Naderi"

cust . balance = 7252.5



نکته :

عضو یک ساختار خود می تواند یک ساختار دیگر باشد.

```
struct date {  
int month;  
int day;  
int year;  
};  
struct account {  
int acc_no ;  
char acc_typer;  
char name[80];  
float balance ;  
date lastpay ; };
```

```
account x, y ;
```

```
x.lastpay.day  
x.lastpay.month  
x.lastpay.year
```

اگر داشته باشیم

آنگاه عضو lastpay بوسیله

مشخص می گردد.



نکته :

می توان آرایه‌ای تعریف نمود که هر عضو آن یک ساختار باشد و حتی به آنها مقادیر اولیه تخصیص نمود.

```
struct struc1 {  
char name[40];  
int pay1;  
int pay2; } ;  
struc1 cust[ ]= {"nader", 3000 , 40000,  
"sara", 4200, 6000,  
"susan", 3700, 25000,  
"saman", 4800 , 2000, };
```



برنامه زیر هر عدد مختلط را بصورت یک ساختار در نظر گرفته، دو عدد مختلط را می‌گیرد و مجموع آنها را مشخص و نمایش می‌دهد.

```
#include <iostream.h>
int main( )
{
struct complex{
float a;
float b; } x, y, z;
cout << "enter 2 complex numbers" << endl ;
cin >> x.a>>x.b ;
cout << endl;
cin >> y.a >> y.b;
z.a = x.a + y.a ;
z.b = x.b + y.b ;
cout << endl << z.a << " " << z.b;
return 0 ;
}
```

بطوریکه $i^2 = -1$

$$x = a + ib$$
$$y = c + id$$
$$x+y = (a+c) + i(b+d)$$

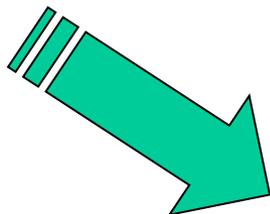

union

union از نظر ساختاری شبیه **struct** می‌باشد. با این تفاوت که عضوهای **union** که تشکیل **union** میدهد همگی از حافظه مشترکی در کامپیوتر استفاده می‌نمایند. بنابراین استفاده از **union** باعث صرفه‌جویی در حافظه می‌گردد.



مثال :

```
union id  
{  
char color [10];  
int size;  
} x , y;
```

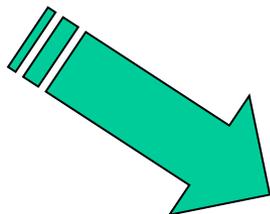


هر کدام از متغیرهای x و y یک رشته 10 کرکتری یا یک مقدار از نوع int می باشد و کامپیوتر یک بلوک حافظه که بتواند رشته 10 کرکتری را در خود جای دهد ، برای color و size در نظر می گیرد .



مثال :

```
union xpq  
{  
  int x ;  
  char y[2] ;  
} p ;
```



اشاره‌گرها (Pointers)

داده‌هایی که در کامپیوتر در حافظه اصلی ذخیره میشوند
بایت‌های متوالی از حافظه بسته به نوع data اشغال می‌کنند.

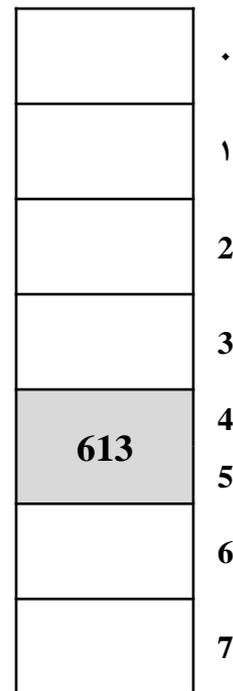
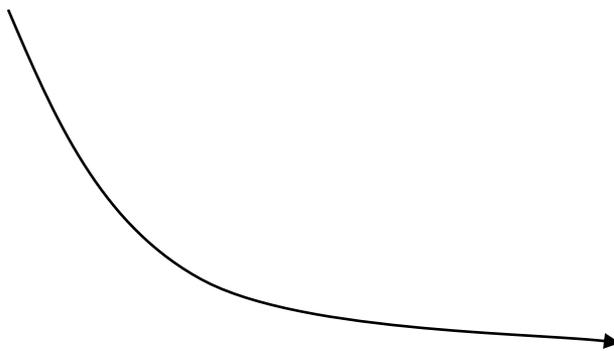
نوع داده	مقادیر	حافظه لازم
int	32767 تا -32768	۲ بایت
long int	2147483647 تا -2147483648	۴ بایت
char	یک کارکتر	۱ بایت
float	3.4e38 تا 1.2e-38	۴ بایت
double	1.8e308 تا 2.2e-308	۸ بایت



اشاره‌گرها (Pointers)

با داشتن آدرس داده در حافظه اصلی می‌توان براحتی به آن داده دسترسی پیدا نمود و از طرف دیگر آدرس هر داده در حافظه آدرس بایت شروع آن داده می‌باشد.

`int x = 613;`



نکته :

در کامپیوتر آدرس‌ها معمولاً دو بایت اشغال می‌نمایند. اگر آدرس x را در px قرار دهیم آنگاه می‌گوئیم که px به x اشاره می‌نماید.



آدرس متغیر x را بوسیله $\&x$ نشان می‌دهیم و عملگر $\&$ را عملگر آدرس می‌نامند.

```
int x , *px  
px = &x ;
```



مثال :

```
int y , x , *px ;
```

```
x = 26 ;
```

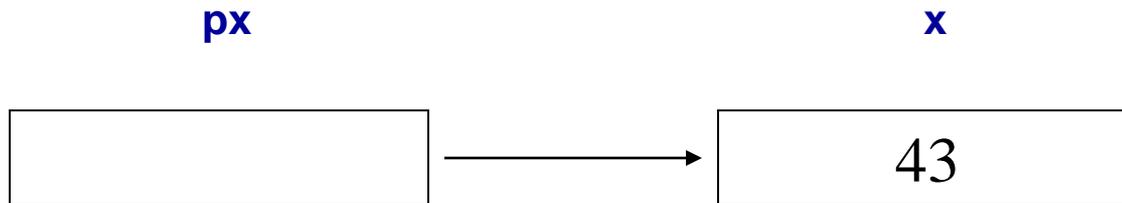
```
px = &x ;
```



حال اگر دستور العمل $x += 10$ را بدهیم :



حال اگر دستور العمل $*px = *px + 7$ بدهیم.



آرایه یک بعدی و اشاره گرها

0	26.5	x
1	24.7	
2	5.8	
3	-73.2	
4	69.0	
5	100.5	
6	-13.24	
7	424.3	
8	187.8	
9	358.2	

اولین عنصر آرایه بوسیله $x[0]$ مشخص می شود.

آدرس اولین عنصر آرایه بوسیله $\&x[0]$ یا بوسیله x مشخص می شود.

آدرس i امین عنصر آرایه بوسیله $\&x[i-1]$ یا بوسیله $x(i-1)$ مشخص می شود.

دو دستورات عمل زیر با هم معادلند .

```
x[ i ] = 82.5 ;  
*(x + i) = 82.5 ;
```

از طرف دیگر اگر داشته باشیم

```
float x[10];  
float *p;
```

دو دستورات عمل زیر معادلند.

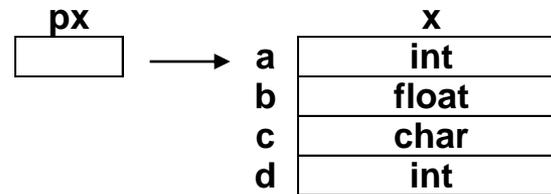
```
p = &x[2] ;  
p = x + 2 ;
```



ساختارها و اشاره گرها

می توان اشاره گری را تعریف نمود که به اولین بایت یک ساختار (struct) اشاره نماید.

```
struct struct1
{
    int a ;
    float b ;
    char c ;
    int d ;
} x, *px ;
```



```
px = &x ;
```

عبارت x.a معادل px->a معادل (*px).a می باشد.



آرایه‌های دوبعدی و اشاره‌گرها

یک آرایه دوبعدی بصورت تعدادی آرایه یک بعدی می‌توان تعریف نمود.
اگر x یک ماتریس 5 سطری و 4 ستونی از نوع اعشاری باشد قبلاً این ماتریس را با

```
float x[5][4];
```

معرفی کردیم. حال با استفاده از اشاره‌گرها بصورت زیر معرفی نمائیم:

```
float (*x)[4];
```



آرایه‌های دوبعدی و اشاره‌گرها

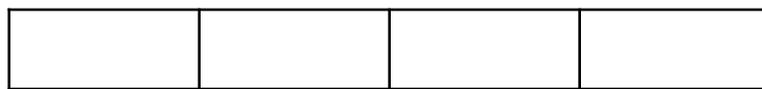
```
float (*x)[4];
```

x



آرایه يك بعدي اول

(x+1)



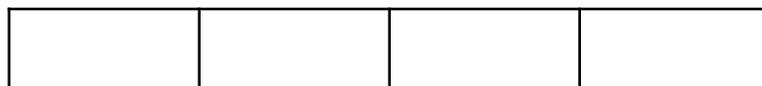
آرایه يك بعدي دوم

(x+2)



آرایه يك بعدي سوم

(x+3)



آرایه يك بعدي چهارم

(x+4)



آرایه يك بعدي پنجم



ایجاد شده و مقادیر عناصر آرایه را به چهار طریق نمایش `int` عنصری از نوع 5 در برنامه زیر یک آرایه می دهد.

```
#include <iostream.h>
#include <conio.h>
int main( )
{
int x[ ]={12, 25, 6, 19, 100};
clrscr( );
int *px=x;
// نام آرایه بدون اندیس، اشاره به عنصر اول آرایه می نماید
for(int i=0; i<=4; i++)
cout << *(x+i) << endl;
//the second method
for(i=0; i<5; i++)
cout << x[ i ] << '\n';
//the third method
for(i=0; i<=4; i++)
cout << px[ i ]<<endl;
//the forth method
for(i=0; i<=4; i++)
cout << *(px+i)<<endl;
return 0; }
```



تخصیص حافظه به صورت پویا یا (عملگر new)

از عملگر **new** برای تخصیص حافظه به صورت پویا می توان استفاده نمود ، در ضمن می توان برای بلوکی از حافظه که تخصیص یافته مقدار اولیه تعیین نمود.

new

delete

dynamic memory allocation



برای تخصیص حافظه باندازه 20 مقدار از نوع `int` که اشاره گر `ptx` به آن اشاره نماید بصورت زیر عمل می شود.

```
int *ptx;  
ptx = new int [20];
```

`ptx` به اولین داده از نوع `int` اشاره می نماید .

`ptx+i` به `i+1` امین عنصر از فضای پیوسته اشاره می نماید.



برنامه زیر یک فضای n عنصری از نوع اعشاری در حافظه ایجاد نموده، سپس آنرا مقدار داده و مجموع مقادیر رامشخص و نمایش می دهد.

```
#include <iostream.h>
int main( )
{
int n;
float *ptr, tot = 0.0;
cout << "enter a value for n " << endl;
cin >> n;
ptr=new float [n];
for(int j=0; j<n; ++j)
{
    cin >> *(ptr + j);
    cout << '\n ';
}
for(j=0; j<=n-1; ++j)
    tot += *(ptr + j);

cout << tot ;
// in order to free the space use
delete[] ptr ;
return 0;
}
```



برنامه زیر آرایه‌های n عنصری از ساختار را ایجاد می‌نماید.

```
#include <iostream.h>
#include <conio.h>
int main( )
{
    struct rec {
        float a;
        int b; } ;
    int n;
    rec *ptr;
    clrscr( );
    cout << " how many records? \n"
    cin >> n ;
    ptr = new rec[n];
    for(int i=0; i<n; ++i) {
        cout <<((ptr+i) ->b=i) << " ";
        cout <<((*(ptr +i)).a = i+0.5) << endl ;
        delete [ ] ptr;
    }
    return 0 ; }
```



برنامه زیر دو مقدار اعشاری را گرفته مقادیر آنها را بکمک تابع `swap` جابه جا می نماید.

```
#include <iostream.h>
#include <conio.h>
void swap(float *, float *);
int main( )
{
float a,b;
cin >> a >> b;
cout << a << endl << b << endl ;
return 0;
}
void swap(float *px , float *py)
{
float t;
t = *px;
*px = *py;
*py = t ;
return;
}
```



رشته‌ها و توابع مربوطه

رشته‌ها در C++ ، آرایه‌ای از کرکترها می‌باشند که با کرکتر '\0' ختم میشوند.

```
char name[ ]= " sara";
```

s
a
r
a
\0



رشته و اشاره گر

هر رشته از طریق اشاره‌گری به اولین کرکتر آن در دسترس قرار می‌گیرد. آدرس یک رشته، آدرس کرکتر اول آن می‌باشد. به رشته‌ها می‌توان مقدار اولیه تخصیص داد.

```
char *name = "sara";
```



برنامه ذیل پنج اسم را بصورت 5 رشته در نظر گرفته آنها را بترتیب حروف الفباء مرتب نموده نمایش می دهد.

```
#include <iostream.h>
#include <string.h>
void sort(char * [ ]);
int main( )
{
char *name[5] = {"sara", "afsaneh", "babak", "saman", "naser" };
sort(name);// display sorted strings
for(int i=0; i<5; ++i)
    cout << name[ i ] << endl;
return 0; }
sort(char *name[ ])
{
char *t;
for(int i=0; i<4; ++i)
for(int j=i+1; j<5; ++j)
if(strcmpi(name[ i ], name[ j ]> 0)
{// interchange the two strings
t= name[ i ];
name[ j ] = name[ i ];
name[ i ] = t ;}
return ; }
```



تابع strcmpi(s1, s2)

رشته‌های s1 و s2 را با هم مقایسه نموده (بدون توجه به حروف کوچک و بزرگ) اگر رشته s1 برابر با رشته s2 باشد مقدار صفر و اگر رشته s1 کوچکتر از رشته s2 باشد یک مقدار منفی در غیر اینصورت یک مقدار مثبت برمی‌گرداند.



تابع (s1, s2) strcmp

رشته‌های s1 و s2 را با هم مقایسه نموده اگر s1 برابر با s2 باشد مقدار صفر و اگر رشته s1 کوچکتر از رشته s2 باشد یک مقدار منفی در غیر اینصورت یک مقدار مثبت برمی‌گرداند.



تابع (strncmp(s1, s2,n))

حداکثر n کرکتر از رشته $s1$ را با n کرکتر از رشته $s2$ مقایسه نموده در صورتیکه $s1$ کوچکتر از $s2$ باشد یک مقدار منفی، اگر $s1$ مساوی با $s2$ باشد مقدار صفر در غیر اینصورت یک مقدار مثبت برمیگرداند.



تابع strcat(s1, s2)

دو رشته s1 و s2 را بعنوان آرگومان گرفته رشته s2 را به انتهای رشته s1 اضافه می‌نماید. کرکتر اول رشته s2 روی کرکتر پایانی '0' رشته s1 نوشته می‌شود و نهایتاً رشته s1 را برمیگرداند.



تابع strncat(s1, s2,n)

دو رشته s1 و s2 و مقدار صحیح و مثبت n را بعنوان آرگومان گرفته، حداکثر n کرکتر از رشته s2 را در انتهای رشته s1 کپی می‌نماید. اولین کرکتر رشته s2 روی کرکتر پایانی '0' رشته s1 می‌نویسد و نهایتاً مقدار رشته s1 را برمیگرداند.



تابع strlen(s)

رشته S را بعنوان آرگومان گرفته طول رشته را مشخص می نماید.



تابع strlen(s)

رشته S را بعنوان آرگومان گرفته طول رشته را مشخص می نماید.



تابع strcpy(s1,s2)

دو رشته s1 و s2 را بعنوان آرگومان گرفته رشته s2 را در رشته s1 کپی می نماید و نهایتاً مقدار رشته s1 را بر می گرداند.



تابع strncpy(s1, s2,n)

دو رشته $s1$, $s2$ و مقدار صحیح و مثبت n را بعنوان آرگومان گرفته، حداکثر n کرکتر را از رشته $s2$ در رشته $s1$ کپی نموده، نهایتاً مقدار رشته $s1$ را برمیگرداند.



نکته مهم

برای استفاده از توابع مربوط به رشته‌ها بایستی حتماً در ابتدا برنامه `#include <string.h>` را قرار دهیم.



مثال :

```
#include <iostream.h>
#include <string.h>
#include <conio.h>
int main( )
{
char *s1= "happy birthday";
char *s2= "happy holidays ";
clrscr( );
cout << strcmp(s1, s2) << endl;
cout << strncmp(s1, s2, 7) << endl ;
return 0;
}
```



مثال :

```
#include <iostream.h>
#include <string.h>
#include <conio.h>
int main( )
{
char *s = "sara";
clrscr( );
cout << strlen(s);
return 0; }
```



تابع زیر معادل تابع کتابخانه strcmp می باشد.

```
int nikstrcmp(char s[] , char t[] )  
{  
    int i=0;  
    while (s[i]==t[i] )  
        if ( s[i++]=='\0' )  
            return 0;  
    return (s[i]-t[i]);  
}
```



فصل هشتم

برنامه نویسی گرا



فهرست مطالب فصل هشتم

- | | |
|--|--------------------------------------|
| 12. <u>عضوهای static</u> | 1. <u>تعریف شی گرای</u> |
| 13. <u>کلاسهای تودرتو</u> | 2. <u>چند ریختی (polymorphism)</u> |
| 14. <u>کلاس های محلی</u> | 3. <u>خاصیت ارث بری</u> |
| 15. <u>استفاده از object ها بعنوان پارامترهای تابع</u> | 4. <u>پشته (stack)</u> |
| 16. <u>برگشت اشیاء</u> | 5. <u>ایجاد شی</u> |
| 17. <u>انتساب اشیاء</u> | 6. <u>ارث بری</u> |
| 18. <u>آرایه اشیاء</u> | 7. <u>سازنده ها و نابود کننده ها</u> |
| 19. <u>اشاره گر به اشیاء</u> | 8. <u>توابع دوست</u> |
| 20. <u>اشاره گر this</u> | 9. <u>کلاس های دوست</u> |
| 21. <u>توابع مجازی و پلی مرفیسم</u> | 10. <u>توابع سازنده پارامتر دار</u> |
| | 11. <u>توابع سازنده یک پارامتری</u> |



تعریف شی گرای

برنامه نویسی شی گرا یا oop یک روش جدید برنامه نویسی می باشد که در آن از ویژگی ساختیافته همراه با چند ویژگی های قوی جدید استفاده می شود. زبان برنامه نویسی ++C امکان استفاده از oop را به راحتی فراهم می نماید.



تعریف شی گرای

در oop ، بطور کلی مساله به تعدادی زیرگروه قطعات مربوط بهم شکسته می شود که برای هر زیر گروه code و data تهیه شده و نهایتاً این زیرگروه ها تبدیل به unit ها یا واحدهائی می شود که objects (یا اشیاء) نامیده میشوند.



نکته مهم:

تمام زبانهای برنامه نویسی شی گرا دارای سه
خصوصیت مشترک زیر می باشند:

الف: **encapsulation** (محصورسازی)

ب: **polymorphism** (چندریختی)

ج: **inheritance** (ارث بری)



محصورسازی (Encapsulation)

محصورسازی مکانیزمی است که **code** و **data** را بهم وصل نموده و هر دوی آنها را از استفاده‌های غیرمجاز مصون نگه می‌دارد. شی یک مؤلفه منطقی است که **code** و **data** را محصور نموده و **code** باعث دستکاری و پردازش **data** می‌شود.



polymorphism (چند ریختی)

چند ریختی مشخصه‌ای است که بیک وسیله امکان میدهد که با تعدادی از سیستمها یا عمیات یا دستگاهها، مورد استفاده قرار گیرد.



inheritance (ارث بری)

ارث بری فرآیندی است که بوسیله آن یک شی (object) می تواند
خاصیت های شی دیگری را دارا شود.



پشته (stack)

پشته ساختاری است که دارای خاصیت **last in first out** می باشد. پشته فضای پیوسته در حافظه اشغال می نماید. عملیاتی که روی پشته انجام می شوند عبارتند از :

الف: **push** ، که باعث می شود یک عنصر وارد پشته شده.

ب: **pop** ، که باعث می شود یک عنصر از پشته خارج گردد.



ایجاد شی (object)

بمنظور ایجاد یک شی بایستی از کلمه رزرو شده **class** استفاده نمود. **class** از نظر ظاهر شبیه ساختار یا **struct** می باشد. پشته را بعنوان یک **object** می توان در نظر گرفت که **data** آن شامل یک آرایه و یک **tos** ، و عملیاتی که روی این **object** انجام می شود عبارتست از **push** ، **initialize** ، **pop** کردن پشته.

در اسلاید بعد مثالی از نحوه ایجاد شی آورده شده است.



مثال :

```
#define SIZE 100
// this creates the class stack.
class stack {
    private :
        int stck[SIZE];
        int tos;
public:
    void init( );
    void push(int i);
    int pop( );
};
```

بدین معنی است که `stck` و `tos` بوسیله توابعی که عضو `object` نباشند غیر قابل دسترسی هستند. و این یکی از روش‌های محصور سازی ارقام داده‌هاست.

بدین معنی است که بوسیله سایر قطعات برنامه قابل دسترسی می‌باشد.



نکته :

فقط توابع عضو می‌توانند به متغیرهای عضو از نوع **private** دسترسی داشته باشند. بایستی توجه داشت که اگر نوع عضوی مشخص نگردد آن عضو به صورت اتوماتیک **private** می‌باشد.

Private



نحوه تعریف تابع عضو یک کلاس

```
void stack :: push(int i)
{
    if(tos == SIZE ) {
        cout << "stack is full.";
        return;
    }
    stck[tos]= i ;
    tos ++ ;
}
```

عملگر :: مشخص می نماید که تابع متعلق به کدام object می باشد. عملگر :: عملگر
scope resolution نامیده می شود.



: stack برنامه کامل

```
#include <iostream.h>
#define SIZE 100
// this creates the class stack.
class stack {
    int stck[SIZE];
    int tos;
public:
    void init(int i);
    int pop( );
    void push(int i);
};
void stack :: init( )
{
    tos = 0 ;
}
void stack :: push(int i)
{
    if(tos == size) {
        cout << "stack is full." ;
        return ;
    }
    stck[tos] = i ;
    tos ++ ;
}
```

```
int stack :: pop( )
{
    if(tos == 0) {
        cout << "stack underflow." ;
        return 0 ; }
    tos -- ;
    return stck[tos];
}

int main( )
{
    stack st1, st2; // create two objects
    st1. init( );
    st2.init( );
    st1.push(1);
    st2.push(2);
    st1.push(3);
    st2.push(4);
    cout << st1.pop( ) << endl;
    cout << st1.pop( ) << endl;
    cout << st2. pop( ) << endl;
    cout << st2. pop( ) << endl;
    return 0; }
```



ارث بری

ارث بری فرآیندی است که بوسیله آن یک شی (object) می تواند
خاصیت های شی دیگری را دارا شود.

در اسلاید بعد مثالی از ارث بری آورده شده است.



مثال :

```
class building {  
int rooms;  
int floors;  
int area;  
public:  
void set_rooms(int num);  
int get_rooms( );  
void set_floors(int num);  
int get_floors( );  
void set_area(int num);  
int get_area( );  
};
```

```
// house is derived from building  
class house : public building {  
int bedrooms;  
int baths;  
public:  
void set_bedrooms(int num);  
int get_bedrooms( );  
void set_baths(int num);  
int get_baths( );  
};
```



در روبرو **object** ای بنام ساختمان یا **building** تعریف گردیده است. هر ساختمان دارای تعدادی اتاق، تعدادی طبقه و سطح زیر بنا نیز می باشد. از طرف دیگر توابعی که برای شی تعریف شده :

حال **object** دیگری بنام **house** تعریف می نمائیم که نه تنها دارای تمام اعضای شی **building** می باشد بلکه دارای دو اقلام داده اضافی و چهار تابع اضافی می باشد. در اینجا عملاً شی **house** از شی **building** ارث می برد :

نکته :

در مثال قبل `building` را `base class` و `house` را `derived class` می نامند . شی `house` تمام اعضای `building` را دارا است بعلاوه اینکه دارای متغیرهای عضوی اضافی `bedrooms` ، `baths` و همچنین توابع عضوی `set_baths()` ، `set_bebrooms()` ، `get_baths()` ، `get_bedrooms()` نیز می باشد..

Inheritance



سازنده‌ها و نابودکننده‌ها (constructors and destructors)

Initialization یا مقدار اولیه دادن بصورت اتوماتیک از طریق تابعی انجام می‌شود بنام تابع **constructor** یا تابع سازنده. تابع سازنده تابع مخصوصی است که عضوی از کلاس بوده و همانم با کلاس می‌باشد.



سازنده‌ها و نابودکننده‌ها (constructors and destructors)

تابع نابود کننده یا **destructor** ، عکس عمل تابع سازنده را انجام می‌دهد. وقتی که شی‌ای از بین می‌رود بصورت اتوماتیک تابع نابود کننده آن فراخوانی می‌گردد.



توابع دوست (friend functions)

با استفاده از کلمه **friend** این امکان وجود دارد که به تابعی که عضو کلاس نمی‌باشد اجازه دسترسی به متغیرهای **private** کلاس را داد. برای آنکه تابعی را دوست اعلان نمائیم از کلمه **friend** قبل از تعریف تابع استفاده می‌نمائیم.

در اسلاید بعد مثالی آورده شده است.



مثال :

```
#include <iostream.h>
#include <conio.h>
class myclass {
int a,b;
public :
friend int sum(myclass x);
void set_ab(int i, int j);
};
void myclass :: set_ab(int i, int j)
{
    a=i;
    b=j;
}
//sum is not a member function
int sum(myclass x)
{
return s.a + x.b;
}
int main( )
{myclass n;
clrscr( );
n. set_ab(5,8);
cout << sum(n);
return 0 ; }
```



۱- کلاسی که از کلاس دیگر ارث می‌برد ، توابع دوست آن کلاس را به ارث نمی‌برند. بعبارت دیگر یک **derived class** ، توابع دوست را به ارث نمی‌برد.

۲- توابع دوست دارای مشخصه نوع ذخیره نمی‌باشند یعنی توابع دوست را نمی‌توان بصورت **static** یا **external** تعریف نمود.



کلاسهای دوست (friend classes)

این امکان وجود دارد که یک کلاس دوست کلاس دیگری باشد. در چنین وضعیتی تابع دوست به کلیه اسامی **private** تعریف شده در کلاس دیگر دسترسی دارد.

Friend Class

در اسلاید بعد مثالی آورده شده است.



مثال :

```
#include <iostream.h>
class coins {
enum units {penny, nickel, dime, quarter, half_ dollar};
friend class amount;
};

class amount {
coins :: units money;
public:
void setm( );
int getm( );
} ob;
void amount :: setm( )
{
money = coins :: dime;
}
int amount :: getm( )
{
return money;
}
int main( )
{
ob.setm( );
cout << ob.getm( );
return 0 ;
}
```



توابع سازنده پارامتردار

امکان انتقال آرگومانها به توابع سازنده وجود دارد. معمولاً از این آرگومانها برای `initialize` نمودن شی در زمان ایجاد آن استفاده میگردد.

در اسلاید بعد مثالی آورده شده است.



مثال :

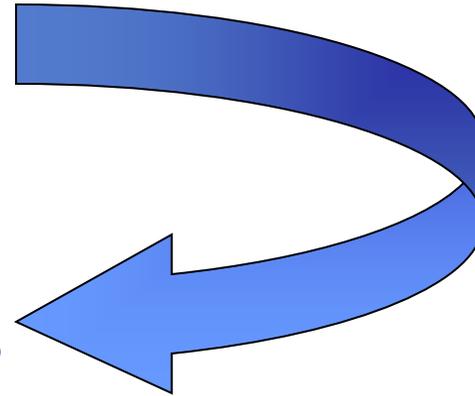
```
#include <iostream.h>
#include <conio.h>
class myclass {
    int x, y;
public :
    myclass(int i, int j) {x = i; y=j; }
    void show( ) {cout << x << endl << y; }
};
int main( )
{
    myclass obj( 3 , 5);
    clrscr( );
    obj.show( );
    return 0;
}
```



توابع سازنده یک پارامتری

```
#include <iostream.h>
#include <conio.h>
class myclass{
    int x;
    public:
    myclass(int i) {x=i;}
    int getx( ) {return x;}
};
```

```
int main( )
{
    clrscr( );
    myclass obj=126; // 126 را به i منتقل کن
    cout << obj.getx( );
    return 0 ;
}
```



عضوهای static

اگر عضو داده‌ای بصورت static اعلان کرده این بدین معنی است که کامپایلر فقط یک کپی از متغیر مذکور را نگهداری نموده و تمام object ها بایستی بصورت اشتراکی از آن کپی استفاده نمایند. برای اینکار می‌بایستی از کلمه static قبل از اعلان عضو استفاده نمود.

در اسلاید بعد مثالی آورده شده است.



مثال :

```
#include <iostream.h>
class shared{
    static int a;
    int b;
public :
    void set(int i, int j) {a=i; b=j; }
    void show( );
};

int shared :: a ;           // define a
void shared :: show( )
{
    cout << "static a: " << a << endl;
    cout << "nonstatic b:  << b << endl";
}

int main( )
{
    shared x,y;
    x.set(1,1); // set a to 1
    x.show( );
    y.set(4,4); // change a to 4
    y.show( );
    x.show( );
    return 0;
}
```



nested classes (کلاسهای تودرتو)

می توان یک کلاس را در یک کلاس دیگر تعریف نمود.
اما بعلت اینکه در C++ برای کلاسها خاصیت ارث
بری وجود دارد نیازی معمولاً به تعریف نمودن یک
کلاس در کلاس دیگر نیست

Nested Classes
Nested Classes



local classes (کلاسهای محلی)

وقتی که کلاسی در درون یک تابع تعریف می شود این کلاس فقط برای آن تابع شناخته شده است و برای توابع دیگر ناشناخته می باشد. چنین کلاسی را کلاس محلی می نامند.

Local Classes
Local Classes



در مورد کلاسهای محلی رعایت نکات زیر ضروری است :

1. تمام توابع عضو بایستی در درون کلاس تعریف گردند.
2. از متغیرهای محلی، تابعی که کلاس در آن تعریف شده نمی تواند استفاده نماید.
3. از متغیرهای عضوی `static` نمی توان استفاده نمود.



استفاده از object ها بعنوان پارامترهای توابع

از **object** ها می توان بعنوان پارامترهای توابع استفاده نمود و مکانیزم انتقال آرگومانها و پارامترها بصورت **call by value** می باشد.



برگشت اشیاء (returning objects)

مقدار برگشتی یک تابع می تواند یک **object** باشد.

در اسلاید بعد مثالی آورده شده است.



مثال :

```
#include <iostream.h>
class myclass {
int i ;
public :
void set_ i(int n) { i=n;}
int get_ i() {return i;}
};
myclass funct( ); // return an object
int main( )
{
    myclass ob;
    ob=funct( );
    cout << ob.get_ i( ) << endl;
    return 0;
}
myclass funct( )
{
    myclass x ;
    x.set_ i(1);
    return x;
}
```



انتساب اشیاء (object assignment)

در صورتیکه دو تا object از یک نوع باشند
می توان یک object را بدیگری انتساب نمود.

در اسلاید بعد مثالی آورده شده است.



مثال :

```
#include <iostream.h>
#include <conio.h>
class myclass{
    int i;
    public:
    void set_i(int n) {i=n;}
    int get_i( ) {return i;}
};
int main( )
{
    myclass ob1, ob2;
    ob1.set_i(126);
    ob2= ob1; // assign data from ob1 to ob2
    clrscr( );
    cout << ob2.get_i( );
    retrun 0 ;
}
```



آرایه اشیاء (array of objects)

امکان استفاده از آرایه در مورد اشیاء می باشد.
بعبارت دیگر می توان در برنامه ها آرایه ای از object ها
داشته باشیم.

در اسلاید بعد مثالی آورده شده است.



مثال :

```
#include <iostream.h>
#include <conio.h>
class myclass{
    int i;
    public:
    void set_i(int j) {i=j;}
    int get_i( ) {return i;}
};
int main( )
{
    clrscr( );
    myclass ob[3];
    int i;
    for(i=0; i<3; i++) ob[ i ].set_i(i+1);
    for(i=0; i<3; i++)
        cout << ob[ i ].get_i( ) << endl;
    return 0;
}
```



اشاره‌گر به اشیاء (pointers to objects)

در مورد اشیاء نیز از اشاره‌گرها نیز می‌توان استفاده نمود.
از عملگر >- در این مورد استفاده می‌شود.

در اسلاید بعد مثالی آورده شده است.



مثال :

```
#include <iostream.h>
#include <conio.h>
class myclass{
    int i ;
    public:
    myclass( ) {i=0;}
    myclass(int j) {i=j;}
    int get_i( ) {return i;}
};
int main( )
{
    myclass ob[3]= {1, 2, 3};
    myclass *p;
    int i;
    p=ob; // get start of array
    for(i=0; i<3; i++)
    {
        cout << p -> get_i( ) << endl;
        p++; // point to next object
    }
    return 0;
}
```



اشاره گر this (this pointer)

هر تابع عضو یک کلاس دارای یک پارامتر مخفی
بنام **this pointer** می باشد. **this** اشاره به **object**
خاصی می نماید.

در اسلاید بعد مثالی آورده شده است.



مثال :

```
#include <iostream.h>
class pwr {
double b;
int e;
double val;
public:
pwr(double base, int exp);
double get_pwr( ) {return val;}
};
pwr :: pwr(double base, int exp)
{
this -> b=base;
    this -> e=exp;
    this -> val =1;
    if(exp == 0) return;
    for(; exp > 0 ; exp --)
this -> val = this -> val *this -> b;
}
int main( )
{pwr x(4.0, 2) , y(2.5, 1), z(5.7,0);
cout << x.get_pwr( ) << " ";
cout << y.get_pwr( ) << " ";
cout << z.get_pwr( ) << " \n ";
return 0; }
```



توابع مجازی و پلی مرفیسم (virtual functions)

تابع مجازی، تابعی است که در **base class** تعریف شد و
بوسیله **derived class** تغییر داده میشود. برای اعلان یک
تابع مجازی بایستی از کلمه **virtual** استفاده نمائیم.
تغییر تابع در کلاس مشتق روی تابعی که در کلاس پایه (**base class**)
تعریف شده انجام می شود.

در اسلاید بعد مثالی آورده شده است.



مثال :

```
#include <iostream.h>
class base {
public :
virtual void vfunc( ){cout << " this is base 's vfunc( ) \n" ;}
};
class derived1 : public base {
public:
void vfunc( ) {cout << " this is derived1 's vfunc( ) " << endl ; }
};
class derived2: public derived1 {
public:
/*vfunc( ) not overridden by derived2.In this case, since derived2 is derived
from derived1, derived1 's vfunc( ) is used */ ;
int main( )
{
base *p, b;
derived1 d1;
derived2 d2;
//point to base
p = &b;
p -> vfunc( ); // access base's vfunc
// point to derived1
p=&d1;
p -> vfunc( ); //access derived1's vfunc( )
//point to derived2
p = &d2;
p -> vfunc( ); // use derived1 's vfunc( )
return 0 ; }
```



بایان



keywords and alternative tokens.

asm	enum	protected	typedef
auto	explicit	public	typeid
bool	extern	register	typename
break	false	reinterpret_cast	union
case	float	return	unsigned
catch	for	short	using
char	friend	signed	virtual
class	goto	sizeof	void
const	if	static	volatile
const_cast	inline	static_cast	wchar_t
continue	int	struct	while
default	long	switch	xor
delete	mutable	template	xor_eq
do	namespace	this	or_eq
double	new	throw	not
dynamic_cast	operator	true	bitand
else	private	try	and_eq
And -- or	bitor	not_eq	compl

