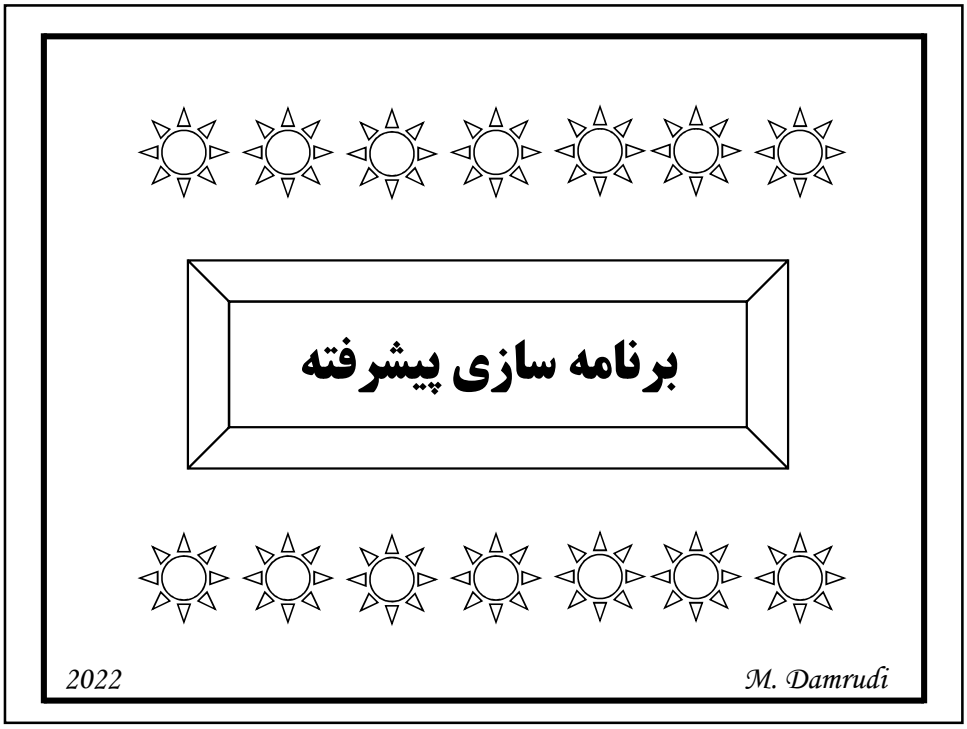


یک برنامه ساده

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
cout<<"Hello C++";
return 0;
}
```

یک برنامه ساده

```
/*
A simple program.
This program contains all of the
basic elements
*/

#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
//main is where program execution begins.
{
cout<<"Hello C++";
return 0;
}
```

یک برنامه ساده

```
/*
A simple program.
This program contains all of the
basic elements
*/

#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;

int main()
//main is where program execution begins.
{
cout<<"Hello C++";
getch();
return 0;
}
```

توضیح Comment

۱–Multiline comment : توضیحات چند خطی نامیده می‌شوند که با یک /* شروع و با */ به پایان می‌رسد.

۲– Single-line comment: توضیح یک خطی نامیده می‌شود که با // شروع شده و در همان خط پایان می‌یابد.

محتویات توضیح توسط کامپایلر نادیده گرفته می‌شود.

main( )

۱- همه برنامه‌ها در C++ ترکیبی از یک یا چند تابع می‌باشند.
۲- تنها تابعی که تمام برنامه‌ها دارند تابع main می‌باشد.
۳- main جایی است که اجرای برنامه از آن‌جا شروع می‌شود.

header files

۱- زبان C++ چندین فایل تعریف می‌کند که فایل‌های سرآمد یا header files می‌گویند.

۲- هر فایل سرآمد شامل اطلاعاتی است که برای برنامه ضروری می‌باشند.

۳- iostream برای پشتیبانی از سیستم ورودی و خروجی می‌باشد. برای استفاده از cout از این فایل سرآمد استفاده می‌شود. محتویات فایل iostream.h به برنامه اضافه می شود.

۴- conio.h برای استفاده از تابع getch مورد نیاز می‌باشد.

cout

یک شناسه از پیش تعریف شده است که مخفف console output می‌باشد و برای نمایش روی صفحه نمایش به کار می‌رود.

- ✓ انتهای تمام دستورات در C++ به سمی‌کولون (;) ختم می‌شود.

stdafx.h

یکprecompiled header file است که شامل فایل های پیش نیاز پروژه های C++ در محیط visual C++ است.

getch()

این تابع باعث می‌شود تا زمانیکه از ورودی کاراکتری دریافت نشده است اجرای برنامه متوقف گردد.

return 0

با این دستور تابع main پایان می‌یابد و مقدار صفر به پروسه فراخواننده main که معمولا سیستم‌عامل است بازگردانده می‌شود.

مقدار صفر نشان دهنده آن است که برنامه به طور عادی پایان یافته است. مقادیر دیگر نشان‌دنده آن است که برنامه به علت خطایی پایان یافته است. return یکی از کلیدواژه‌های C++ می‌باشد که برای بازگرداندن مقداری از یک تابع به کار می‌رود.

using namespace std

حوزه نام std بصورت عمومی تعریف می شود. در صورتی که این عبارت استفاده نشود باید قبل از تمام دستورات ورودی و خروجی std:: اضافه گردد.

دومین برنامه ساده

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;
int main()
{
int num;
num=99;
cout<<"The value is:";
cout<<num;
getch();
return 0;
}
```

تعریف متغیر

- متغیر محل نامگذاری شده‌ای از حافظه است که می‌توان مقداری در آن قرار داد.
- محتوای یک متغیر قابل تغییر است و ثابت نیست.
- برای تعریف متغیر ابتدا نوع متغیر و سپس نام دلخواهی برای آن تعیین می‌کنیم.

```
type   variable;
```

- با علامت = مقدار ۹۹ را داخل متغیر num قرار داده می‌شود.
- می‌توان چند متغیر را همزمان تعریف نمود.

```
int a,b;
```

نوع داده‌های اصلی

کلمه کلیدی	دامنه	شرح
bool	مقدار درست یا نادرست	true و false
char	۱۲۷ تا ۱۲۸−	کاراکترهای ۸ بیتی 'A'
int	۳۲۷۶۷ تا ۳۲۷۶۸−	عدد صحیح
float	۴/۳E+۳۸ تا ۴/۳E−۳۸	مقدار اعشاری
double	۷/۱E+۳۰۸ تا ۷/۱E−۳۰۸	مقدار اعشاری با دقت مضاعف
void	اعمال نمی‌شود	بیانگر یک عبارت بدون مقدار
wchar_t	۰ تا ۶۵۵۳۵	کاراکترهای عریض (زبان چینی)

سومین برنامه ساده

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;
int main()
{
cout<<"First line.\n";
cout<<"Second line.\n";
cout<<"Third line.\n";
getch();
return 0;
}
```

کاراکتر خط جدید
(newline)

خروجی برنامه

First line.
Second line.
Third line.

سومین برنامه ساده

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;
int main()
{
cout<<"One\nTwo\nThree";
getch();
return 0;
}
```

خروجی برنامه

One
Two
Three

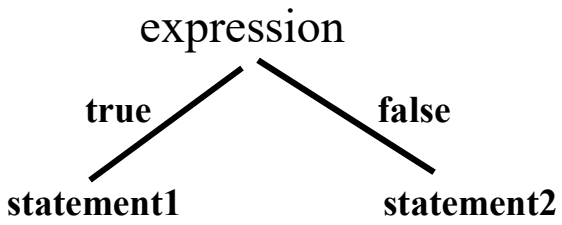
دریافت اطلاعات از کاربر

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;
int main()
{
int a,b;
cin>>a>>b;
cout<<"The value a is:"<<a<<"\n";
cout<<"The value b is:"<<b<<"\n";
getch();
return 0;
}
```

به جای "\n" می توان از **endl** استفاده کرد.

دستور if

```
if (expression) statement1;
      else statement2;
```



استفاده از if

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;
int main()
{
float num1,num2;
int choice;
cout<<"Enter values:";
cin>> num1>>num2>>choice;
if (choice==1) cout<<num1+num2;
if (choice==2) cout<<num1-num2;
getch();
return 0;
}
```

استفاده از else

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;
int main()
{
float num1,num2;
int choice;
cout<<"Enter values:";
cin>> num1>>num2>>choice;
if (choice==1) cout<<num1+num2;
else cout<<num1-num2;
getch();
return 0;
}
```

استفاده از بلوک کد

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;
int main()
{
float num1,num2;
int choice;
cout<<"Enter values:";
cin>>choice;
if (choice==1) {
cin>> num1>>num2;
cout<<num1+num2;
}
getch();
return 0;
}
```

بزرگترین و کوچکترین عدد

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

using namespace std;

int main()
{
    cout<<"Minimum int, Maximum int\n";
    cout<<INT_MIN<<", "<<INT_MAX <<"\n";
    cout<<"unsigned int \n"<<UINT_MAX <<"\n";
    system("pause");
    return 0;
}
```

Output

```
Minimum int, Maximum int
-2147483648, 2147483647
unsigned int
4294967295
Press any key to continue . . .
```

System("pause") به جای getch()
استفاده شده است.

زوج / فرد

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

using namespace std;
int main()
{
    int num;
    cout<<"Enter a number:";
    cin>>num;
    if(num%2==0)
        cout<<"The number is even.";
    else
        cout<<"The number is odd.";
    system("pause");
    return 0;
}
```

Output

```
Enter a number:11
The number is odd.Press any key to continue . . .
```

دو عدد و یک عملگر

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

using namespace std;
int main(){
    float a,b;
    char op;
    cout<<"Enter numbers:\n";
    cin>> a >> b;
    cout<<"Enter operation:\n";
    cin>>op;
    if(op=='+')
        cout<<a + b;
    if(op=='-')
        cout<<a - b;
    if(op=='*')
        cout<<a * b;
    if(op=='/')
        cout<<a / b;
    system("pause");
    return 0;}
```

Output

```
Enter numbers:
6
4
Enter operation:
/
1.5Press any key to continue . . .
```

استفاده از عملگر ?

```
if(condition) var=exp1;
else var=exp2;
```

```
var=condition ? exp1:exp2;
```

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(){
    int i;
    cout<<"Enter a number.\n";
    cin>>i;
    i=i>0 ? 1: -1;
    cout<<i<<"\n";
    system("pause");
    return 0;
}
```

عملگر sizeof

ازعملگرهای یکتائی می باشد و مشخص کننده تعداد بایت هائی است که یک نوع داده اشغال می‌کند.

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

عملگرهای منطقی

عملگر	عمل
&&	AND
||	OR
!	NOT

دستور switch

شکل کلی دستور به صورت زیر است:

```
switch(expression){
      case(val1):
      instructions
      break;
      case(val2):
      instructions
      break;
      .
      .
      .
      default:
      instructions
}
```

به کار بردن break

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;
int main()
{
int num;
cin>> num;
switch (num) {
    case 1:
    cout<< "First Case"; break;
    case 2:
    cout<< "Second Case";break;
    case 3:
    cout<< "Third Case"; break;
    case 4:
    cout<< "Forth Case"; break;
    case 5:
    cout<< "Fifth Case"; break;
    default:
    cout<<"Nothing";
    }
getch();
return 0;
}
```

تمرین

برنامه ای بنویسید که

- عددی را از ورودی دریافت کرده، قدر مطلق آن را نمایش دهد.
- عددی را از ورودی دریافت کرده، بخشپذیری آن بر ۳ و ۵ را بررسی نماید.
- دو عدد را دریافت کرده و بدون استفاده از متغیر کمکی تعویض نماید.
- یک عدد بین ۱ تا ۷ دریافت کرده معادل روز هفته را نمایش دهد.
- دو عدد را دریافت کرده، اعلام کند آیا این دو عدد متوالی هستند یا خیر؟
- شعاع دایره را دریافت کرده، محیط و مساحت دایره را محاسبه کرده و نمایش دهد.
- ضرایب یک معادله درجه۲ را دریافت کرده، ریشه‌های آن را درصورت وجود محاسبه کرده و نمایش دهد.
- یک عدد بین ۱ تا ۱۲ دریافت کرده معادل ماه مربوطه را نمایش دهد.
- نمره دانشجوئی را از ورودی دریافت کرده، با توجه به مقدار نمره یکی از خروجی های زیر را نمایش دهد.

Grade	output
17 – 20	A
14 - 17	B
12 – 14	C
10 – 12	D
0 – 10	F

استفاده از عملگرهای افزایشی و کاهشی

i=i+1;	=	i++;
i=i-1;	=	i--;

استفاده از Assignment operators

*= += -= /=

x=x + y <= x+=y

استفاده از عملگرهای افزایشی و کاهشی

می‌توان عملگرها را پیش از متغیرها هم به کار برد. اما معنای متفاوتی دارند.

j=i++; ⇨ ابتدا مقدار i به j نسبت داده شده و سپس i یک واحد اضافه می‌شود.

j=++i; ⇨ ابتدا مقدار i یک واحد اضافه می‌شود و سپس مقدار i به j نسبت داده می‌شود.

مثال

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

using namespace std;
int main()
{
int i, j, k, l;
i=15;
j=20;
k=i++;
l=++j;
cout<<i<<"\n"<<j<<"\n"<<k<<"\n"<< l<<"\n";
system("pause");
return 0;
}
```

Output

```
16
21
15
21
Press any key to continue . . .
```

مثال

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

using namespace std;
int main()
{
int i, j, k, l;
i=15;
j=20;
k=i--;
l=--j;
cout<<i<<"\n"<<j<<"\n"<<k<<"\n"<<l<<"\n";
system("pause");
return 0;
}
```

Output

```
14
19
15
19
Press any key to continue . . .
```

تقدم عملگرها

1	()
2	! ~ ++ -- sizeof
3	* / %
4	+ -
5	<< >>
6	< <= > >=
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?
14	= += -= *= /= %=
15	,

for حلقه

حالت کلی این دستور به صورت زیر می‌باشد.

```
for (initialization; expression; increment) statement;
```

مقداردهی اولیه متغیر کنترل کننده حلقه

شرط حلقه

افزایش یا کاهش مقدار متغیر کنترل کننده حلقه

حلقه for

حالت کلی این دستور به صورت زیر می‌باشد.

```
for (initialization; expression; increment) statement;
```

- initialization: تنها یکبار و پیش از شروع حلقه اجرا می‌گردد.
- expression: در آغاز هر تکرار انجام می‌شود.
- increment: پس از اجرای بدنه حلقه، اجرا می‌شود.

برنامه ای بنویسید که اعداد بین ۱ تا ۱۰ را نمایش دهد.

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

using namespace std;
int main()
{
int num;
for (num=1; num<11; num= num+1)
    cout<< num<<" ";
system("pause");
return 0;
}
```

Output

```
1 2 3 4 5 6 7 8 9 10 Press any key to continue . . .
```

حلقه for

کاهش و افزایش متغیر کنترل کننده می‌تواند بیش از یک واحد باشد.

```
for (num=10; num>0; num= num-4)
```

```
for (num=0; num<11; num= num+4)
```

عدد صحیح و مثبت n را دریافت کرده فاکتوریل آنرا محاسبه ونمایش دهد.

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

using namespace std;
int main()
{
int  n, i ;
long fact = 1 ;
cout << "Enter a positive integer number\n";
cin >> n;
for( i=1; i<=n; ++i)
    fact *= i;
cout << fact<<"\n" ;
system("pause");
return 0;
}
```

Output

```
Enter a positive integer number
5
120
Press any key to continue . . .
```

دوعدد را دریافت کرده و عدد اول را به توان عدد دوم برساند و نتیجه را نمایش دهد. x^y

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

using namespace std;
int main()
{
int x,y,p=1;
cin>>x>>y;
for(int n=1;n<=y;n++)
    {
        p=p*x;
    }
cout<<"Power is:"<<p<<"\n";
system("pause");
return 0;
}
```

Output

```
2
3
Power is:8
Press any key to continue . . .
```

به کار بردن for

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

using namespace std;
int main()
{
int num;
for (num=11; num<11; num= num+1)
    cout<< num<<" ";
system("pause");
return 0;
}
```

Output

?

به کار بردن بلوک کد در for

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

using namespace std;
int main()
{
int num, sum, prod;
sum=0;
prod=1;
for (num=1; num<11; num= num+1){
    sum= sum + num;
    prod= prod * num;
}
cout<< "product:"<< prod<< "\tsum:"<< sum <<"\n";
system("pause");
return 0;
}
```

Output

```
product:3628800 sum:55
Press any key to continue . . .
```

کاهش به جای افزایش در for

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

using namespace std;
int main()
{
int num, sum, prod;
sum=0;
prod=1;
for (num=10; num>0; num= num-1){
    sum= sum + num;
    prod= prod * num;
}
cout<< "product:"<< prod<< "\tsum:"<< sum <<"\n";
system("pause");
return 0;
}
```

Output

```
product:3628800 sum:55
Press any key to continue . . .
```

حلقه های تو در تو

```
#include "stdafx.h"
#include <iostream>
 using namespace std;

int main()
{
    system("color 5");
    for(int i=1; i<=10; i++)
        for(int j=1; j<=10; j++)
            cout<<i*j<<"\t";
        cout<<"\n";

    system("pause");
    return 0;
}
```

رنگی کردن خروجی

Output

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Press any key to continue . . .

مقادیر منطقی در زبان C

==0 ⇒ false

!=0 ⇒ true

استفاده از مقادیر منطقی

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
bool p,q;
p=true;
q=true;
cout<<"p=true,"<<"q=true"<<"\t";
cout<<"(p&&q):"<<(p&&q)<<" ";
cout<<"(p||q):"<<(p||q)<<"\n";
p=true;
q=false;
cout<<"p=true,"<<"q=false"<<"\t";
cout<<"(p&&q):"<<(p&&q)<<" ";
cout<<"(p||q):"<<(p||q)<<"\n";
p=false;
q=true;
cout<<"p=false,"<<"q=true"<<"\t";
cout<<"(p&&q):"<<(p&&q)<<" ";
cout<<"(p||q):"<<(p||q)<<"\n";
p=false;
q=false;
cout<<"p=false,"<<"q=false"<<"\t";
cout<<"(p&&q):"<<(p&&q)<<" ";
cout<<"(p||q):"<<(p||q)<<"\n";
system("pause");
return 0;
}
```

Output

```
p=true,q=true   (p&&q):1 (p||q):1
p=true,q=false  (p&&q):0 (p||q):1
p=false,q=true  (p&&q):0 (p||q):1
p=false,q=false (p&&q):0 (p||q):0
Press any key to continue . . .
```

عدد دریافتی اول است؟

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

using namespace std;
int main()
{
    int num, i, is_prime;
    cin>>num;
    is_prime=1;
    for (i=2; i<=num/2; i++)
        if(!(num%i)) is_prime=0;
    if(is_prime) cout<<"The number is prime.";
    else cout<<"The number is not prime.";
    system("pause");
    return 0;
}
```

فرض می‌شود عدد اول است.

حلقه while

حالت کلی این دستور به صورت زیر می‌باشد.

while (expression) statement;

شرط حلقه

تا زمانیکه شرط برقرار باشد دستورات اجرا می‌شود.
شرط حلقه در ابتدای حلقه کنترل می‌شود.

اعداد فرد بین num و صفر

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

using namespace std;
int main()
{
    int num;
    cin>> num;
    while(num) {
    if (num%2) cout<<num<< " ";
    num--;
    }
    system("pause");
    return 0;
}
```

Output
5
5 3 1 Press any key to continue . . .

حلقه do

حالت کلی این دستور به صورت زیر می‌باشد.

```
do {
        statements
 } while (expression);
```

شرط حلقه

تا زمانیکه شرط برقرار باشد دستورات اجرا می‌شود.
شرط حلقه در انتهای حلقه کنترل می‌شود.
حلقه حداقل یکبار اجرا خواهد شد.

استفاده از break برای خروج از حلقه

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

using namespace std;
int main()
{
    int i;
    for(i=1; i<100;i++){
        cout<< i<< " ";
    if(i==10) break;
    }
    system("pause");
    return 0;
}
```

Output
1 2 3 4 5 6 7 8 9 10 Press any key to continue . . .

استفاده از continue برای اجرای دور بعدی حلقه

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

using namespace std;
int main()
{
    int x;
    for(x=1; x<100;x++){
        continue;
        cout<< x;
    }
    system("pause");
    return 0;
}
```

Output
Press any key to continue . . .

حلقه goto

حالت کلی این دستور به صورت زیر می‌باشد.

goto label;

label:

برچسب

با دیدن goto به خطی می‌رود که برچسب یکسانی دارد.

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

using namespace std;
int main()
{
    int i;
    i=1;
    again:
        cout<<i<<" ";
        i++;
    if(i<10) goto again;
    system("pause");
    return 0;
}
```

Output
1 2 3 4 5 6 7 8 9 Press any key to continue . . .

نگاهی دقیق‌تر به نوع داده‌ها

با استفاده از توصیف‌کننده‌های نوع (type modifiers) می‌توان نوع داده‌های char, int, float, double را تعدیل کرد. توصیف‌کننده‌های نوع عبارتند از :

signed, unsigned, long, short

توصیف کننده نوع پیش از نوع داده آورده می‌شود. long int i;

signed (علامت‌دار)

signed را می‌توان در مورد انواع int و char به‌کار برد.

استفاده از signed برای اعداد صحیح کار زایدی است زیرا به طور پیش فرض اعداد صحیح علامت‌دار هستند.

بسته به نوع پیاده‌سازی یک char می‌تواند علامت‌دار یا بدون علامت باشد. در صورتی که بدون علامت باشد اعداد مثبت بین ۰ تا ۲۵۵ را نگاه می‌دارد. درصورتی که به عنوان signed باشد می‌تواند اعدادی در دامنه ۱۲۸- تا ۱۲۷ را نگاه می‌دارد. signed char x;

long

long را می‌توان در مورد نوع داده‌های int و double به‌کار برد.

long بر روی int طول آن را بر حسب بیت دوبرابر می‌کند. اگر در محیطی طول int ۱۶ بیت باشد با استفاده از long طول آن ۳۲ بیت می‌شود.

long بر روی double طول double را نیز بر حسب بیت دوبرابر می‌کند.

long int g;

short

short را می‌توان تنها در مورد نوع داده int به‌کار برد.

short بر روی int طول آن را بر حسب بیت نصف می‌کند. اگر در محیطی طول int ۳۲ بیت باشد با استفاده از short طول آن ۱۶ بیت می‌شود.

unsigned (بدون علامت)

unsigned را می‌توان در مورد نوع داده‌های int و char به‌کار برد.
این توصیف‌کننده را همراه توصیف‌کننده‌های short و long نیز می‌توان مورد استفاده قرار داد.

signed & unsigned

۱- تفاوت اعداد صحیح علامت‌دار در نحوه تفسیر بیت با مرحله بالاتر می‌باشد.
۲- اگر عدد صحیح بدون علامت باشد از تمامی بیت‌ها برای نگه‌داری مقادیر استفاده می‌شود.
۳- اگر عدد صحیح علامت‌دار باشد در آن صورت کامپایلر کدی تولید می‌کند که فرض می‌نماید از بیت با مرتبه بالاتر به عنوان علامت استفاده می‌شود. اگر ۰ باشد عدد مثبت و اگر ۱ باشد عدد منفی است.
۴- عموما اعداد منفی با روش two's complement (مکمل دو) بیان می‌شوند.

ترکیبات ممکن از توصیف‌کننده‌ها

نوع	بیت	دامنه
char	۸	۱۲۸– تا ۱۲۷
unsigned char	۸	۰ تا ۲۵۵
signed char	۸	۱۲۸– تا ۱۲۷
int	۱۶	۳۲۷۶۸– تا ۳۲۷۶۷
unsigned int	۱۶	۰ تا ۶۵۵۳۵
signed int	۱۶	۳۲۷۶۸– تا ۳۲۷۶۷
short int	۱۶	مثل int
unsigned short	۱۶	مثل unsigned int
signed short int	۱۶	مثل short int
unsigned long	۳۲	۰ تا ۴۲۹۴۹۶۷۲۹۵
long	۳۲	۲۱۴۷۴۸۳۶۴۸ – تا ۲۱۴۷۴۸۳۶۴۷

ترکیبات ممکن از توصیف‌کننده‌ها

نوع	عرض
long int	۳۲
unsigned long	۳۲
signed long int	۳۲
float	۳۲
double	۶۴
long double	۸۰

نکته

ممکن است در محیط کاری شما long و short اصلا تاثیری نداشته باشد. بستگی به کامپایلر دارد.

const

اگر پیش از نوع داده یک متغیر توصیف کننده const آورده شود محتوای آن از سوی برنامه تغییر نمی‌کند. (مگر سخت‌افزاری)

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(){
    const int ten=10;
    cout<<ten*5<<"\n";
    system("pause");
    return 0;
}
```

typedef

typedef OldDataType NewName;

با استفاده از typedef برای یک نوع داده موجود نام تازه‌ای ایجاد کرد.

```
typedef int length;
typedef length depth;
depth d;
typedef short int myint;
```

define

از define جهت تعریف یک شناسه و یک دنباله کاراکتری استفاده می‌شود طوری که در source برنامه هرجا با آن شناسه برخورد شود آن دنباله کاراکتری جانشین آن گردد. این دستور از دستورات preprocessor است و نیازی به ; ندارد.

```
#define macro-name character-sequence
```

```
#define UP 1
#define GETFILE "Enter File Name"
#define myfor for(int k=10;k<15;k++)
```

Preprocessor برنامه سیستم است که قبل از ترجمه برنامه توسط کامپایلر تغییراتی در برنامه ایجاد می کند.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(){
    #define myfor for(int k=10;k<15;k++)
    myfor
        cout<<k<<"\t";
    system("pause");
    return 0;
}
```

تمرین

برنامه ای بنویسید که

- **دو عدد را دریافت کرده، اعداد مابین این دو عدد را نمایش دهد. a<b**
- **مضارب ۷ بین ۱ تا ۱۰۰ را نمایش دهد.**
- **اعداد فرد کوچکتر از ۲۰ را چاپ کند.**
- **۱۰۰ عدد را دریافت کرده، میانگین آن‌ها را محاسبه کرده و نمایش دهد.**
- **n جمله اول سری فیبوناچی را نمایش دهد.**
- **۵۰ عدد را دریافت کرده، بزرگترین و کوچکترین عدد را نمایش دهد.**
- **دو عدد m و n را دریافت کرده و یک مستطیل رنگی با یک کاراکتر نمایش دهید.**
- **یک عدد را دریافت کرده مقلوب عدد را نمایش دهد.**
- **تعداد و مجموع ارقام یک عدد دریافتی را نمایش دهد.**
- **بزرگترین و کوچکترین رقم یک عدد دریافتی را نمایش دهد.**
- **تعدادی عدد را دریافت کرده، مجموع اعداد, میانگین اعداد، بزرگترین عدد و کوچکترین عدد را محاسبه کرده و نمایش دهد. عدد آخر صفر باشد.**

آرایه‌ها و رشته‌ها

یک آرایه مجموعه‌ای از متغیرهای به‌هم مرتبط است که به وسیله یک نام مشترک مشخص می‌شود.

آرایه یک بعدی فهرستی از متغیرهایی از یک نوع واحد است که با استفاده از یک نام مشترک به همه آنها مراجعه می‌شود.

به هر متغیر یک آرایه یک عنصر آرایه گفته می‌شود.

حالت کلی آرایه‌ها به صورت زیر می‌باشد:

```
type var_name[size];
```

برای دسترسی به هر عنصر آرایه باید از شماره آن عنصر به عنوان index استفاده نمود.

آرایه‌ها و رشته‌ها

اندیس تمام آرایه‌ها از صفر شروع می‌شود و تا size-1 ادامه دارد.

```
int d[20];
```

```
d[0]=100;
```
مقدار عنصر صفر آرایه را برابر با ۱۰۰ قرار می‌دهد.

برای مقداردهی آرایه‌ها نیاز به حلقه‌های for داریم. مقادیر عناصر را باید تک به تک قرار داد.

مشکلی که وجود دارد این است که بر روی ایندکس آرایه‌ها عمل بررسی دامنه (bound checking) صورت نمی‌پذیرد.

به عنوان برنامه‌نویس خود باید مراقب این مساله باشید.

مثال:

```
int t[20];
t[25]=14;
```

برنامه‌ای بنویسید که ۳۰ عدد صحیح مثبت را دریافت کرده و max آنها را بیابید، نمایش دهید.

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

using namespace std;
int main()
{
    int i, max=0;
    int n[30];
    for(i=0; i<30;i++)
    {
        cin>>n[i];
        if(n[i]>max) max=n[i];
    }

    cout<<"Maximum is:"<<max;
    system("pause");
    return 0;
}
```

در زمان معرفی یک متغیر می‌توان به آن مقدار اولیه داد.

برنامه‌ای بنویسید که ۵۰ عدد صحیح مثبت را دریافت کرده و آنها را به ترتیب عکس دریافت نمایش دهید.

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

using namespace std;
int main()
{
    int i;
    int n[50];
    for(i=0; i<50;i++)
        cin>>n[i];

    for(i=49; i>=0;i--)
        cout<<n[i]<<"\t";
    system("pause");
    return 0;
}
```

برنامه‌ای بنویسید که دو آرایه ۵ عنصری را دریافت کرده مجموع دو آرایه را در آرایه سوم قرار دهد.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
    int a[5], b[5], c[5], i;
    for(i=0;i<5;i++)
    {
        cin>>a[i];
        cin>>b[i];
    }
    for(i=0;i<5;i++)
    {
        c[i]=a[i]+ b[i];
        cout<<a[i]<<"+"<<b[i]<<"="<<c[i]<<"\t";
    }
    system("pause");
    return 0;
}
```

تمرین

برنامه‌ای بنویسید که

- یک آرایه ۲۰ عنصری را دریافت کرده بزرگترین عنصر را به همراه اندیس آن نشان دهد.
- یک آرایه ۱۰ عنصری را دریافت کرده اعلام کند هر عنصر زوج است یا فرد.
- یک آرایه ۵ عنصری دریافت کرده یک واحد به آنها اضافه کرده و نمایش دهد.
- توسط آرایه، نمودار افقی برای اعداد {۵،۱۳،۸،۱۰،۱۵،۱۱} رسم کند.
- یک آرایه را دریافت و در آرایه دیگر به صورت وارونه کپی کرده، آرایه دوم را نمایش دهد.
- عناصر آرایه را گرفته مکعب آنها را محاسبه و در همان عنصر ذخیره نمایید.
- ۳۰ عدد را دریافت کرده، تعداد اعدادی را که به ۴ ختم می‌شوند، نمایش دهد.
- ۱۰ عدد را از ورودی دریافت کرده و در یک آرایه قرار دهد. سپس یک عدد دیگر را خوانده و در آرایه جستجو کند و وجود یا عدم وجود آنرا در آرایه مشخص کند.
- ۱۰ عدد مرتب شده را از ورودی دریافت کرده و در یک آرایه قرار دهد. سپس یک عدد دیگر را خوانده و در جای مناسب در آرایه قرار دهید.
- ۵ عدد را دریافت و در آرایه قرار دهد، اختلاف مجموع اعداد زوج و فرد را نمایش دهد.
- برنامه ای بنویسید که یک عدد را از مبنای ۱۰ به مبنای دو برده و نمایش دهد(با آرایه).

رشته‌ها (strings)

متداول‌ترین کاربرد آرایه‌های یک بعدی رشته‌ها می‌باشند.

هر رشته به صورت یک آرایه کاراکتری ختم شده به تهی null-terminated تعریف می‌شود.

آرایه‌ای که قرار است یک رشته را نگه دارد باید به کاراکتر تهی ختم شود به این معنی که باید یک بایت بزرگتر در نظر گرفته شود تا کاراکتر تهی در آن قرار گیرد. مقدار کاراکتر تهی صفر است.

```
char name[11];
```

حداکثر ۱۰ کاراکتر را نگه می‌دارد.

هر ثابت رشته‌ای نیز به یک کاراکتر تهی ختم می‌شود که کامپایلر به طور اتوماتیک آن را به انتهای رشته اضافه می‌کند.

چند تابع رشته‌ای استاندارد

```
strcpy(to,from);
```

این تابع رشته موجود در from رابه to کپی می‌کند. محتوای from تغییری پیدا نمی‌کند.

```
char str[80];
strcpy(str,"hello");
cout<<str;
```

این تابع دامنه آرایه مقصد را چک نمی‌کند و باید خود اندازه آن را بررسی کنید. (به همراه کاراکتر ختم‌کننده تهی null)

```
strcpy(str,"");
```

رشته‌ای با طول صفر ایجاد می‌کند. به چنین رشته‌ای رشته تهی (null string) می‌گویند.

```
strcat(to,from);
```

این تابع رشته موجود در from رابه to می‌چسباند . محتوای from تغییری پیدا نمی‌کند.

چند تابع رشته‌ای استاندارد

```
char str[80];
strcpy(str,"hello");
strcat(str,"Dear");
cout<<str;
```

این تابع دامنه آرایه مقصد را چک نمی‌کند و باید اندازه آن را بررسی کنید.

```
strcmp(s1,s2);
```

این تابع دو رشته را مقایسه می‌کند
اگر دو رشته یکسان باشد، تابع مقدار صفر می‌گیرد.
اگر s1<s2، مقداری کوچکتر از صفر برمی‌گرداند.
اگر s1>s2، مقداری بزرگتر از صفر برمی‌گرداند.

```
cout<<strcmp("one","one");
```

```
strlen(str);
```

این تابع طول یک رشته را برحسب تعداد کاراکترهای آن باز می‌گرداند.
کاراکتر ختم کننده تهی را به حساب نمی‌آورد.

مثال

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

using namespace std;
int main()
{
    char str1[80], str2[80];
    int i;
    cout<<"Enter two strings.\n";
    cin>>str1>>str2;
    cout<<"length of str1 is:"<<strlen(str1)<<"\n";
    cout<<"length of str2 is:"<<strlen(str2)<<"\n";
    i=strcmp(str1,str2);
    if(!i) cout<<"the strings are equal";
    if (strlen(str1)+strlen(str2)<80){
        strcat(str1, str2);
        cout<<str1<<"\n";}
    strcpy(str1,str2);
    cout<<str1<<"\n";
    system("pause");
    return 0;
}
```

cin.get()

این تابع یک کاراکتر را از صفحه کلید می‌گیرد. برای استفاده از این تابع در ابتدای برنامه باید از فایل سرآمد iostream استفاده شود.

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

```
cin.get (طول آرایه و نام آرایه) ;
cin.get (S,15);
```

در این روش کامپایلر آنقدر از کاربر حرف می گیرد تا به طول آرایه تعریف شده برسد. ممکن است کلمه مورد نظر کاربر ۳ حرفی باشد برای حل این مشکل از روش زیر استفاده کنیم:

```
cin. get (S,15,'*');
```

کاراکتر جدا کننده در واقع شرط پایان است که خودمان تعیین می کنیم و یادمان باشد همواره ۱کاراکتر از کاراکترهای داده شده کم می شود چون از آنجا شرط تمام است.

gets/puts

- برای دریافت متن از تابع gets می توان استفاده کرد.
- برای نمایش متن از تابع puts می توان استفاده کرد.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
 char t[10];
 gets(t);
 puts(t);
 system("pause");
 return 0;
}
```

toupper() , tolower()

کاراکتر ورودی را به معادل بزرگ خود تبدیل می‌کند.

```
int toupper(int ch);
```

کاراکتر ورودی را به معادل کوچک خود تبدیل می‌کند.

```
int tolower(int ch);
```

باید از header file ، ctype.h استفاده نمود.

یکی از کاربردهای متداول آن پشتیبانی از interface است.

به این معنی که در دریافت ورودی از کاربر مفید است.

مثال

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
    char command[80];
    int i, j;
    for(;;){
        cout<<"operation:add,sub,mul,div,mod,quit\n";
        cin>>command;
        for(i=0;i<strlen(command);i++)
            command[i]=tolower(command[i]);
        if(!strcmp(command, "quit"))
           break;
        cout<<"Enter two No.\n";
        cin>>i>>j;
        if(!strcmp(command, "add"))
            cout<<i+j<<"\n";
        if(!strcmp(command, "sub"))
            cout<<i-j<<"\n";
        if(!strcmp(command, "mul"))
            cout<<i*j<<"\n";
        if(!strcmp(command, "div"))
            cout<<i/j<<"\n";
        if(!strcmp(command, "mod"))
            cout<<i%j<<"\n" ;}
    system("pause");
    return 0;
}
```

string

این نوع داده برای نگهداری رشته ای از یک یا چند کاراکتر مورد استفاده قرار می گیرد. کلاس `string` با متد هایی که در اختیار می گذارد کار با کاراکتر ها و مدیریت متن ها را آسان تر می کند. برای استفاده از `string` باید هدرفایل `string` را به برنامه اضافه کنیم.

```
string str1 = "This class is fun.";
string str2 = "Learn C++.";

//.size() show length of string (18)
cout<< str1.size();

//.empty() if string is empty return 1 else return 0. (0)
cout<< str1.empty();

//oprator [] :you can access to i th character.(c)
cout<< str1[5];
```

string

```
string str1 = "This class is fun.";
 string str2 = "Learn C++.";

//.append(string ) :append string to end of other string.
//(This class is fun.Learn C++.)
 str1.append(str2);

//.erase(int i,int j) :Delet form charcter i th and reply that j time.
//(This class++.)
 str1.erase(10,15);
 cout<< str1<<"\n";

 //.insert(int n,const sting &) : from character n program start embed new
string in other string. (This class+Java is good.+.)
 str1.insert(11,"Java is good.");
 cout<< str1;

 //.find(string) search string form left in other string and return position of
it. (2)
 cout<< str1.find("is");
```

string

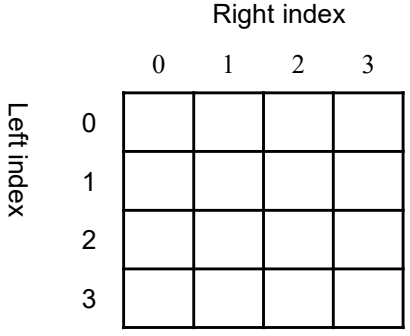
```
//.rfind(string) search string form right in other string and return position of it. (16)
 cout<< str1.rfind("is");

 //operator + : you can connect too string or more by + (ab)
 string str3 = "a";
 string str4 = "b";
 string str5 = "";
 str5 = str3 + str4;
cout<< str5;


 //.compare(string) : do compration operator on to string if to sting equal return 0 if first was biger than other return -1 else return +1. (8)
 cout<< str1.compare(str2);

 //.clear() :assign string to "".
 str1.clear();
 cout<< str1;
```

آرایه های چندبعدی

```
int count[10][12];
```

سطر ستون

یک آرایه دوبعدی از چپ به راست و یک ردیف یک ردیف مورد دستیابی قرار می‌گیرد.



نمایش ذهنی از آرایه‌های دوبعدی

مثال

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
    int two_d[4][5],i,j;
    for(i=0;i<4;i++)
     for(j=0;j<5;j++)
        two_d[i][j]=i*j;
    for(i=0;i<4;i++)
    {
     for(j=0;j<5;j++)
        cout<<two_d[i][j]<<"\t";
     cout<<"\n";
    }
    system("pause");
    return 0;
}
```

Sizeof(int)

4*5*4=80

Output

```
0   0   0   0   0
0   1   2   3   4
0   2   4   6   8
0   3   6   9   12
Press any key to continue . . .
```

مقداردهی اولیه آرایه ها

```
type array-name[size]={value-list};
```

```
char k[5]={1,4,9,16,25};
```

0	1	2	3	4
1	4	9	16	25

```
int B[2][3]={{1},{4,9}};
```

0	1	2
1	0	0
4	9	0

```
k[0] =>  1
```

```
k[4] =>  25
```

```
char m[5]={0};
```

0	1	2	3	4
0	0	0	0	0

مثال

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
    double radians[5][2]={
        1.0, 0.0175,
        2.0, 0.0349,
        3.0, 0.0524,
        4.0, 0.0698,
        5.0, 0.0873
    };
    double degrees;
    int i;
    cin>>degrees;
    for(i=0; i<5; i++)
        if (radians[i][0]==degrees)
        {
            cout<<radians[i][1]<<"\n";;
            break;
        }
    system("pause");
    return 0;
}
```

Output
4
0.0698
Press any key to continue . . .

آرایه‌های بدون اندازه

```
int pwr[]={1,2,4,8,16,32,64,128};
char prompt[]="Enter your name";
```

آرایه‌هایی که صراحتا ابعاد آنها مشخص نشده است.تعداد کاراکترها و یا اعداد را شمرده و اندازه آرایه را آن تغییر می‌دهد.

```
int sq[][3]={
1, 2, 3,
4, 5, 6,
7, 8, 9
};
```

مزیت این نحوه معرفی آرایه‌ها این است که می‌توان جدول را بدون آنکه ابعاد آرایه را تغییر دهیم، بلندتر یا کوتاهتر نماییم.

جداول رشته‌ای

```
char names[10][40];
```

۱۰ رشته هر کدام با طول حداکثر ۴۰ کاراکتر(با احتساب کاراکتر ختم کننده تهی)

```
cout<<names[0];
```
اولین رشته جدول

```
cin>>names[2];
```
دریافت سومین رشته جدول

ثابت‌های کاراکتری خاص

کد	معنا
\t	tab
\n	newline
\\	back slash **کاراکتر**

توابع کاراکتری ...is

تابع	کاری که انجام می‌دهد
isalnum(ch)	حروف الفبا یا یک رقم باشد، true برمی‌گرداند.
isalpha(ch)	حروف الفبا باشد، true برمی‌گرداند.
isdigit(ch)	یک رقم باشد، true برمی‌گرداند.
isspace(ch)	فاصله خالی باشد، true برمی‌گرداند.

ctype.h

برنامه‌ای بنویسید که تعداد فاصله‌های یک متن را اعلام می‌کند.

```
#include "stdafx.h"
#include <iostream>
#include <ctype.h>
#include <string.h>

using namespace std;
int main()
{
    char str[]="This is a test";
    int i, spaces;
    spaces=0;
    for(i=0; i<strlen(str); i++)
        if(isspace(str[i]))
            spaces++;
    cout<<spaces<<"\n";
    system("pause");
    return 0;
}
```

```
Output
3
Press any key to continue . . .
```

تمرین

برنامه‌ای بنویسید که

- آرایه ۲ بعدی ۱۰ در ۱۰ را با مقادیر جدول ضرب، مقدار دهی کرده و نمایش دهد.
- دو ماتریس ۴*۴ را از ورودی دریافت کرده، تفاضل آن دو را در خروجی نمایش دهد.
- در یک ماتریس ۴*۴ عناصر قطر اصلی را یک و بقیه را صفر قرار دهد و نمایش دهد.
- دو ماتریس ۴*۳ و ۵*۴ را دریافت کرده، حاصلضرب آن دو را نمایش دهد.
- نام ۴۰ دانش‌آموز و نمره آن‌ها را از ورودی دریافت کرده، نام و نمره دانش‌آموزانی که نمره آن‌ها بیشتر از ۱۸ است را نمایش دهد.
- یک رشته راخوانده و تمام حروف بزرگ به کوچک تبدیل نماید و نمایش دهد.
- نمایش کاراکترهای **a** تا **d** به همراه کد اسکی آنها.
- مقداری را به عنوان رمز دریافت کند، در صورتی که رمز درست بود اعلام کند به برنامه خوش آمدید در غیراینصورت اعلام کند رمز اشتباه است.
- یک متن حداکثر ۱۰۰ کاراکتری را در یافت کرده، تعداد تکرار **A**، تعداد تکرار **DA** را نمایش دهد و تمام **K**ها را با **L**جایگزین کند.
- ۵۰ اسم را دریافت کرده، تعداد اسامی که با **B** شروع می‌شود را نمایش دهد.
- دو رشته را از ورودی دریافت کرده، به هم پیوند دهد و در خروجی نمایش دهد.
- رشته ای را از ورودی دریافت کرده، تعداد ارقام موجود در رشته را محاسبه کرده و نمایش دهد.

تابع

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

using namespace std;
void Areas(int base,int height);
int main()
{
    Areas(10, 20);
    Areas(5, 6);
    Areas(8, 9);
    system("pause");
    return 0;
}
(void) Areas(int base, int height)
{
    cout<<"Area is:"<<base*height/2<<"\n";
}
```

Prototype

argument

Parameters

تابع

تابع شامل یک یا چند دستور بوده و عمل بخصوصی را انجام می‌دهد.

تابع دارای نامی است و با همین نام فراخوانی می‌شود.

در یک تابع نمی‌توان تابع دیگری ایجاد کرد اما می‌توان تابع دیگری را فراخوانی کرد.

برای استفاده مجدد تابع نوشته شده، بهتر است برای تابع header file ساخته و با پسوند h. در فولدر include که مخصوص header file ها است ذخیره نمود.

prototype شامل سه چیز است: نوع مقدار بازگشتی تابع، تعداد پارامترهای تابع، نوع داده پارامترهای آن تابع

پیش‌الگوی یک تابع (function prototype) برای معرفی آن تابع، پیش از آنکه تعریف شود، استفاده می‌گردد.

تابع

مقدار بازگشتی توابع باید مشخص شود. اگر مشخص نشود اکثرا به عنوان int درنظر گرفته می‌شود. درصورتی‌که تابع دارای مقدار بازگشتی نباشد نوع مقدار بازگشتی void تعریف می‌شود(تابع return ندارد).

به مقداری که به یک تابع ارسال می‌شود argument می‌گویند.

حد بالای آرگومانها توسط کامپایلری که از آن استفاده می‌شود تعیین می‌گردد، اما هر کامپایلر استانداردی حداقل ۲۵۶ آرگومان را قبول می‌کند.

متغیرهایی که آن آرگومانها را دریافت می‌کنند نیز باید معرفی شوند، به این متغیرها parameters گفته می‌شود.

تابع

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

using namespace std;
float Areas(float base,float height);
int main()
{
    float S,x,y;
    cin>>x>>y;
    S=Areas(x,y);
    cout<<"The area is:"<<S<<"\n";
    system("pause");
    return 0;
}

float Areas(float base, float height)
{
 return base*height/2;
}
```

```
Output
3
4
The area is:6
Press any key to continue . . .
```

توابع ریاضی

استفاده از توابع ریاضی با فایل سرآمد math.h

عملیات ریاضی	تابع در زبان C
cos t	double cos(double t)
e^t	double exp(double t)
x%y	double fmod(double x, double y)
$\log^t$	double log10(double t)
x^y	double pow(double x, double y)
Sin t	double sin(double t)
$\sqrt{t}$	double sqrt(double t)

در C++ می‌توان توابعی به صورت inline تعریف کرد که واقعا فراخوانی نشوند، بلکه در هر نقطه‌ای که فراخوانی شده‌اند کد آنها قرار داده شود. این توابع سریعتر اجرا می‌شوند. پیش از استفاده حتما باید تعریف شود.

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

using namespace std;

inline int even(int x)
{
    return !(x%2);
}
int main()
{
    if(even(10)) cout<<"10 is even \n";
    if(even(11)) cout<<"11 is even \n";
    if(even(12)) cout<<"12 is even \n";
    system("pause");
    return 0;
}
```

```
Output
10 is even
12 is even
Press any key to continue . . .
```

تابع

زمانی که به انتهای تابع و } یا به دستور return برسیم تابع به روتین فراخواننده خود باز می‌گردد. return می‌تواند مقداری را به روتین فراخواننده‌اش بازگرداند.

یک تابع می‌تواند چندین دستور return داشته باشد.

تابعی که خودش را فرابخواند recursive نامیده می‌شود.

تابع بازگشتی (Recursive)

تابعی که خود را به صورت مستقیم یا از طریق تابع دیگر به صورت غیرمستقیم فراخوانی می کند.

بازگشتی مستقیم:

```
f1(int a)
{
   int b;
   .
   f1(b);
   .
}
```

بازگشتی غیرمستقیم:

```
f1(int a)        f2(int c)
{                {
   int b;           int d;
   .                .
   f2(b);           f1(d);
   .                .
}                }
```

هر تابع بازگشتی دارای یک یا چند مقدار اولیه است که آنرا حالت توقف تابع می نامند. تابع بازگشتی از طریق فراخوانی خودش (حالت بازگشتی) به حالت توقف می رسد.

جملات فیبوناچی تا جمله n

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int fibo(int x);
int main()
{
    int i, n;
    cin>>n;
    for (i=1;i<=n;i++)
        cout<<fibo(i)<<"\t";
     system("pause");
     return 0;
}
int fibo(int x)
{
    if (x<=2) return 1;
    else return fibo(x-1)+fibo(x-2);
}
```

میدان دید (scope)

```
int myabs(int x);
int main()
{
    int i;
    for (i=1;i<=8;i++)
    {
        int n;
        cin>>n;
        cout<<myabs(n)<<"\n";
    }
    system("pause");
    return 0;
}
int myabs(int x)
{
    if (x<=0) return -1 * x;
    else return x;
}
```

روشهای انتقال پارامترهای تابع

call by value, call by reference

value: در این روش مقدار آرگومان به پارامتر صوری آن سابروتین کپی می‌شود بنابراین تغییراتی که روی مقادیر پارامترهای آن سابروتین اعمال می‌شود روی آرگومانهایی که جهت فراخوانی آن سابروتین به کار برده می‌شود تاثیری **نخواهد داشت**.

reference: در این روش به جای مقدار آرگومان آدرس آن به پارامتر مورد نظر کپی می‌شود بنابراین تغییراتی که روی مقادیر پارامترهای آن سابروتین اعمال می‌شود روی آرگومانهایی که جهت فراخوانی آن سابروتین به کار برده می‌شود تاثیر **خواهد داشت**.

call by value

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void f(double x, double y);
int main(){
    double i,j;
    i=6.45;
    j=9.35;
    f(i,j);
    cout<<i<<"\t"<<j<<"\t";
    system("pause");
    return 0;
}
void f(double x, double y){
    x=x+3;
    y=y-3;
    cout<<x<<"\t"<<y<<"\t";;
}
```

Output

```
9.45    6.35    6.45    9.35    Press any key to continue . . .
```

call by reference

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void f(double &x, double &y);
int main(){
    double i,j;
    i=6.45;
    j=9.35;
    f(i,j);
    cout<<i<<"\t"<<j<<"\t";
    system("pause");
    return 0;
}
void f(double &x, double &y){
    x=x+3;
    y=y-3;
    cout<<x<<"\t"<<y<<"\t";
}
```

Output

```
9.45    6.35    9.45    6.35    Press any key to continue . . .
```

تمرین

برنامه‌ای بنویسید که

- مقادیر **k** و **n** را دریافت کرده، و تابع زیر را با فراخوانی تابع فاکتوریل محاسبه کرده و نمایش دهد. **y=n!k!/(k!-(n-k)!)**
- با استفاده از فراخوانی تابع مقدار تابع زیر را محاسبه کرده و نمایش دهد.

$$y = \begin{cases} x^3 + 4x + 5 & x \geq 0 \\ 1 & x < 0 \end{cases}$$

- با استفاده از تابع بازگشتی فاکتوریل عدد دریافتی را محاسبه کرده و نمایش دهد.
- با استفاده از تابع محیط و مساحت مستطیل را محاسبه نماید.
- بزرگترین عدد بین ۲ عدد ورودی با تابع را نمایش دهد.
- با تابع مقدار X^n را برای مقادیر دریافت شده محاسبه نماید.
- مقدار X^n را با تابع بازگشتی محاسبه کرده و نمایش دهد.
- **n** امین جمله سری فیبوناچی را با استفاده از تابع بازگشتی محاسبه کرده و نمایش دهد.
-

کد ASCII

- هر کاراگتر یک کد ASCII دارد که می توان در برنامه نویسی از آن استفاده نمود.
- کد اسکی ۲۷ مربوط به کلید Esc است.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>

int main()
{
    char x;
    for(;;)
    {
        x=getch();
        if(x==27)
            return 1;
    }
    system("pause");
    return 0;
}
```

حلقه بی نهایت

مثال

یک سطر متن انگلیسی که به CTRL Z ختم می شود را دریافت کرده و نمایش می‌دهد.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(){
    char  x;
    while((x = cin.get( )) !=EOF)
        cout << x ;
    system("pause");
    return 0;
}
```

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

اعدا تصادفی

- ✓ مقادیر تصادفی در اکثر برنامه‌های کاربردی در زمینه شبیه سازی و بازیهای کامپیوتری نقش مهمی را ایفا می‌نمایند.
- ✓ برای ایجاد یک عدد تصادفی صحیح بین ۰ و ۳۲۷۶۷ از تابع `rand()` استفاده می شود.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
 for(int  j=1; j<=10; j++)
    cout << rand() << "\n";
 system("pause");
 return 0;
}
```

output

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
Press any key to continue . . .
```

اعدا تصادفی

✓ هر چند بار برنامهٔ قبل را اجرا نمائیم جواب یکسانی را می گیریم. برای تصادفی کردن اعداد باید از تابع ( ) srand استفاده نمائیم. این تابع به یک آرگومان صحیح از نوع unsigned نیاز دارد که به آن seed گفته می‌شود.

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

output

Enter seed value : 4
51
17945
27159
386
17345
27504
20815
20576
10960
6020
Press any key to continue . . .

نتیجه پرتاب ۲ تاس

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

output

Enter seed: 16
1 3Press any key to continue . . .

اعداد تصادفی بین ۰ و ۱

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
 unsigned seed ;
 cout<< "Enter seed: " ;
 cin >> seed ;
 srand(seed) ;
 for(int i=1; i<=10; ++i)
 cout<< rand( ) / 32768.0 <<"\n";
 system("pause");
 return 0;
}
```

output

```
Enter seed: 14
0.00256348
0.827789
0.280518
0.355316
0.677094
0.0376892
0.585052
0.734589
0.373413
0.307465
Press any key to continue . . .
```

struct

هر ساختار از دو یا چند عضو که به همراه هم یک واحد منطقی را می‌سازند، تشکیل می‌شود.

```
struct name{
    type member1;
    type member 2;
    .
    .
    .
    .
    .
}variables;
```

ساختارها شبیه آرایه ها هستند، بدین صورت که یک نوع داده گروهی(جمعی) است که فضای پیوسته از حافظه اصلی را اشغال می نماید. اما عناصر ساختار الزاماً از یک نوع نمی باشند بلکه اعضای یک ساختار می توانند از نوع های مختلف مانند float، int و ... باشند. اعضای ساختار ربط منطقی دارند. اعضای ساختار را field یا elementگویند.

مثال

نام ساختار

```
struct time
{
int hour ; // 0 to 23
int minute ; // 0 to 59
int second; // 0 to 59
};
```

struct

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

روش اول

```
struct Info{
int   Id;
char  Id_type;
char  name[80];
} St1, St2, St3;
```

روش دوم

```
struct Info{
int   Id;
char  Id_type;
char  name[80];
};
Info St1, St2, St3;
```

struct

مقدار اولیه برای ساختارها:

```
struct Info{
int   Id;
char  Id_type;
char  name[80];
};
Info St={13, 'p', "Asadi"};
```

به منظور دسترسی به عناصر یک ساختار از عملگر (dot). استفاده می‌گردد.

```
struct Info{
int   Id;
char  Id_type;
char  name[80];
};
Info St;
St.Id=145;
St.Id_type='p';
strcpy(St.name,"Asadi“);
```

struct

عضو یک ساختار خود می‌تواند یک ساختار دیگر باشد. (ساختارهای تو در تو)

```
struct date {
int  month;
int  day;
int  year;
};

struct Info{
int Id;
char name[80];
date entrance; };
```

```
Info x;
x.entrance.month=10;
x.entrance.day=22;
x.entrance.year=1378;
```

struct

عضو یک ساختار خود می‌تواند یک ساختار دیگر باشد.
(روش دیگر)

```
struct Info{
int Id;
char name[80];
  struct date {
      int  month;
      int  day;
      int  year;
      } entrance;
};
```

آرایه ای از ساختارها:

```
struct Info{
int   Id;
char  Id_type;
char  name[80];
};
Info St[30];
```

مثال

نام خانوادگی دانشجویان بزرگتر از ۱۸ سال

```
#include "stdafx.h"
#include <iostream>
using namespace std;
struct students{
  char name[40];
  char family[50];
  int  age;
}st1, st2;
int main()
{
    students st3[100];
    int i;
    for(i=0;i<100;i++)
        cin>>st3[i].name>>st3[i].family>>st3[i].age;
    for(i=0;i<100;i++)
        if(st3[i].age>=18) cout<<st3[i].family<<"\n";
    system("pause");
    return 0;
}
```

تمرین

برنامه‌ای بنویسید که

- چهار تاس برای دو بازیکن (هر کدام دو تاس) در نظر بگیرد. هر بازیکن دو تاس خود را می اندازد. مجموع دو تاس هر بازیکن بیشتر بود آن بازیکن را برنده اعلام کند. در صورتی که مجموع دو تاس هر دو بازیکن برابر بود تساوی اعلام شود.
- با استفاده از ساختار، طول و عرض یک مستطیل را دریافت کرده، مساحت چهار مستطیل را محاسبه کرده و نمایش دهد.
- ساختاری شامل اطلاعات نام، نام خانوادگی، سال ورود، نمره ۱، نمره ۲ و نمره ۳ دانشجویان را در ساختاری ذخیره کند. سپس اطلاعات ۲۰ دانشجو را دریافت کرده و معدل هر یک را به همراه نام خانوادگی نمایش دهد.
- ساختاری شامل اطلاعات نام، نام خانوادگی، سال استخدام، حقوق پایه، بیمه و مالیات کارمندان را در ساختاری ذخیره کند. سپس اطلاعات ۱۰ کارمند را دریافت کرده و حقوق دریافتی هر یک را به همراه نام و نام خانوادگی نمایش دهد.

enum

enum type-name {enumeration-list} variable-list;

می‌توان نوع داده مجتمعی ساخت که از لیستی از ثابت‌های صحیح دارای اسم تشکیل شده باشد.

enum colors {red,green,yellow} mycolor;

ذکر کردن نام enum یا variable list اختیاری است، اما یکی حتما باید ذکر شود.

ثابت های شمارشی رشته نیستند بلکه مقادیر ثابت صحیح با نام هستند.

enum

```
enum colors {red,green=9,yellow} mycolor;
```

به متغیر شمارشی فقط مقادیری را می توان نسبت داد که در آن شمارش تعریف شده باشند. (در مثال بالا red, green, yellow)

کامپایلر از سمت چپ مقادیر صحیح را نسبت می دهد، و یک واحد یک واحد اضافه می کند.

در صورتی که یه یک عنصر مقداری نسبت داده شود، عنصر بعدی یک واحد بزرگتر از قبلی خواهد بود.

enum

```
#include "stdafx.h"
#include <iostream>
using namespace std;
enum language{C,CPP,Pascal,BASIC,Ada};

int main()
{
    language p;
    p=BASIC;
    switch(p)
    {
        case C:cout<<"C language";break;
        case CPP:cout<<"C++ language";break;
        case Pascal:cout<<"Pascal language";break;
        case BASIC:cout<<"BASIC language";break;
        case Ada:cout<<"Ada language";break;
    }
    system("pause");
    return 0;
}
```

bit fields

bit-fields عضوی از یک ساختار است که از یک یا چند بیت تشکیل می گردد. با استفاده از آن می توان به کمک یک اسم به یک یا چند بیت موجود در داخل یک byte یا word دسترسی پیدا نمود.

Type name:size;

type: int یا unsigned است. اگر نوع علامت دار باشد، بیت با بالاترین مرتبه به عنوان علامت در نظر گرفته می شود.

کامپایلر عموما میدان های بیتی را در کوچکترین واحدی از حافظه ذخیره می کنند که می تواند آن ها را در خود جای دهد.

```
struct Info{
char name[50];
unsigned department:3;
unsigned month:4;
unsigned vacancy:1;
}I[100];
```

bit fields

لزومی ندارد برای هر بیت نامی مشخص کرد. می توان برای دستیابی به اولین و آخرین بیت یک بایت از میدان بیتی استفاده کرد.

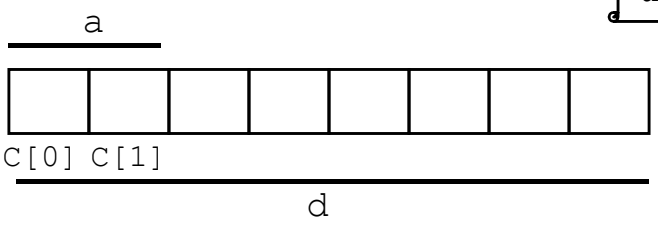
```
struct Info{
unsigned first:1;
unsigned :6;
unsigned last:1;
};
```

union

- union از نظر ساختاری شبیه struct می‌باشد. با این تفاوت که عضوهائی که تشکیل union می دهند همگی از حافظه مشترکی در کامپیوتر استفاده می‌نمایند. بنابراین استفاده از union باعث صرفه‌جوئی در حافظه می‌گردد.
- union از یک مکان ثابت حافظه که بین یک یا چند متغیر به اشتراک گذاشته شده باشد، تشکیل می‌گردد.
- این متغیرها که در آن محل حافظه با هم مشترک هستند، ممکن است انواع متفاوتی باشند. در هر لحظه تنها از یکی از آن متغیرها می‌توان استفاده کرد.
- ذکر نام union یا variable list اختیاری است، اما حتما یکی باید ذکر شود.

```
union type_name{
type member1;
type member2;
:
type memberN;
}variable_list;
```

union




```
union examp1{
int a;
char c[2];
double d;
} sample;
sample.d=14.6;
```

encoder، decoder با union

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int encode(int i);
int main()
{
int i,n;
cin>>n;
i=encode(n);
cout<<n<<" encoded is:"<<i;
cout<<"\n";
i=encode(i);
cout<<i<<" decoded is:"<<i<<"\n";
system("pause");
return 0;
}
```

```
int encode(int i){
union crypt{
int num;
char c[2];
}crypt1;
unsigned char ch;
crypt1.num=i;
//swap bytes
ch=crypt1.c[0];
crypt1.c[0]=crypt1.c[1];
crypt1.c[1]=ch;
return crypt1.num;
 }
```

کلاس های حافظه

✓ متغیرها بوسیله نوع (type) و کلاس حافظه متمایز می‌شوند. نوع متغیر می تواند int، float، double و ... باشد ولی کلاس حافظهٔ یک متغیر در مورد طول عمر و وسعت و دامنهٔ متغیر بحث می‌نماید.

- automatic
- external
- static
- 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 می رود. بعبارت دیگر وسعت و دامنهٔ متغیرهای از نوع automatic تابعی می‌باشد که متغیر در آن اعلان گردیده است. با توجه به اینکه متغیرهای محلی (local) همگی به صورت پیش فرض اتوماتیک هستند به ندرت کلمه auto در برنامه ها دیده می شود.

کلاس های حافظه

static

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void f();
int main()
{
    int i;
    for (i=1;i<=10;i++)
        f();
    system("pause");
    return 0;
}
void f(){
    static int count=0;
    count++;
    cout<<count<<"\n";
}
```

Output
1
2
3
4
5
6
7
8
9
10
Press any key to continue . . .

- متغیرهای static نیز می توانند محلی (local) یا سراسری (global) باشند.
- اگر در درون توابع تعریف شوند از نظر وسعت و دامنه شبیه متغیرهای automatic هستند ولی در خاتمهٔ اجرای تابع، حافظه وابسته به این نوع متغیرهـــا از بین نمی‌رود بلکه برای فراخوانی بعدی تابع باقی می‌ماند.

static باعث می‌شود که یک متغیر محلی بین دو فراخوانی تابع دست نخورده باقی بماند. تنها یکبار مقداردهی اولیه می‌شود آن هم زمانیکه برای نخستین بار وارد بلوک می‌شود.

کلاس های حافظه

static

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void func(void);
static int count = 10; /* Global variable */
int main()
{
  while(count--)
    {
       func();
    }
 system("pause");
 return 0;
}
// Function definition
void func( void )
{
    static int i = 5; // local static variable
    i++;
    cout << "i is " << i ;
    cout << " and count is " << count << std::endl;
}
```

Output
i is 6 and count is 9
i is 7 and count is 8
i is 8 and count is 7
i is 9 and count is 6
i is 10 and count is 5
i is 11 and count is 4
i is 12 and count is 3
i is 13 and count is 2
i is 14 and count is 1
i is 15 and count is 0
Press any key to continue . . .

کلاس های حافظه

```
#include "stdafx.h"
#include <iostream>
using namespace std;
long int fib(int count);
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;
 system("pause");
 return 0;
}
long int fib(int count)
{
static long int t1 = 1, t2=1;
long int  t;
t =(count <3) ?1 : t1 + t2;
t1 = t2;
t2 = 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 می‌شود.

کلاس های حافظه

external

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int functa(int x, int y);
int w; // external variable
int main()
{
 int a, b, c, d;
 cin >> a >> b ;
 c=functa(a, b) ;
 d=functa(w, b+1);
 cout << "\n" << c << "\n" << d << "\n" << w ;
 system("pause");
 return 0;
}
int functa(int x, int y)
{
 cout << w ;
 w = x + y ;
 cout << "\n" << w << "\n";
 return x%y ;
}
```

Output

```
4
3
0
7
7
11

1
3
11Press any key to continue . . .
```

- متغیرهای از نوع external متغیرهائی هستند که بیرون از توابع اعلان می شوند و وسعت و دامنه فعالیت آنها کلیه توابعی می‌باشد که در زیر دستور اعلان متغیر قرار دارد.
- زمانیکه متغیر یا تابعی تعریف می شود که قرار است چند فایل از آن استفاده کنند از این نوع استفاده می شود.
- در مثال روبرو w می تواند در فایل دیگری به صورت زیر استفاده شود:
- `extern int w;`

کلاس های حافظه

register

- ✓ زمانیکه متغیری از نوع `register` اعلان می‌شود از کامپیوتر عملاً درخواست می‌شود که به جای حافظه (RAM) از یکی از رجیسترهای موجود استفاده نماید.
- ✓ معمولاً از نوع رجیستر برای شاخص‌های دستور تکرار و یا اندیسهای آرایه‌ها استفاده می‌شود. متغیرهای از نوع رجیستر قابل استفاده در دستور `cin` نمی‌باشند.
- ✓ متغیرهای از نوع `register` عمدتا متغیرهایی هستند که تناوبا به وجود آمده و از بین می روند. این متغیرها نیز متغیرهای محلی هستند.
- ✓به عنوان مثال زمانی که در برنامه یک `counter` وجود دارد یا زمانی که در برنامه می خواهید مجموع حساب کنید، در این صورت متغیری دارید که خیلی زیاد مورد استفاده قرار می گیرد. در این صورت به کامپایلر اعلام می کنید در صورت امکان این متغیرها را در قسمتی از سخت افزار قرار دهد که سرعت بیشتری از حافظه دارد، تا برای خواندن و نوشتن این متغیر زمان کمتری تلف شود.
- ✓امروزه کامپایلرها خود تشخیص این امر را داده و چنین متغیرهای را به صورت `register` تعریف می کنند و نیازی به دستور از طرف برنامه نویس ندارند.

:: Scope Resolution operator

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

using namespace std;

int x = 10;
int main()
{
    int x = 14;
    x+=5;
    cout<<x<<"\n";
    {
        int x = 20;
        cout<<x<<"\n";
        cout<<::x<<"\n";
    }
    system("pause");
    return 0;
}
```

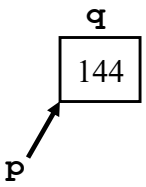
```
Output
19
20
10
Press any key to continue . . .
```

اشاره‌گر

type *var_name;

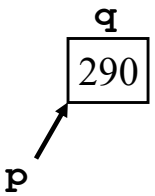
- ✓اشاره‌گر متغیری است که آدرس حافظه یک شی دیگر را در خود نگاه می دارد.
- ✓type نوع داده شیئی است که این اشاره گر به آن اشاره می کند.
- ✓*: محتوای آدرسی که قبل از آن قرار گرفته است.
- ✓&: آدرس متغیری که پیش از آن قرار گرفته است.
- ✓ رجوع به مقدار یک متغیر با اشاره گر رجوع غیرمستقیم indirection گفته می شود.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 int *p,q;
 q=144;
 p=&q;
 cout<<*p;
 cout<<p;
 system("pause");
 return 0;
}
```

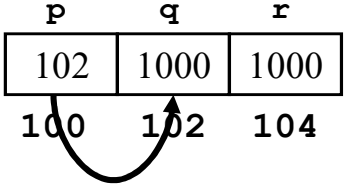


اشاره‌گر

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 int *p,q;
 p=&q;
 *p=290;
 cout<<q;
 system("pause");
 return 0;
}
```




```
p=&q;
*p=1000;
r=*p;
```



اشاره‌گر

✓از آنجایی که کوچکترین واحدی از حافظه که می تواند دارای آدرس باشد یک بایت است، نمی توان آدرس یک متغیر بیتی را بدست آورد. (نمی توان اشاره گری به یک میدان بیتی داشته باشیم)

✓یک متغیر اشاره گر در آغاز دارای مقدار اولیه بامعنایی نیست. اگر سعی کنید از اشاره گری، پیش از آنکه آدرس یک متغیر را به آن بدهید، استفاده کنید، خطا است.

✓اگر اشاره گری دارای مقدار `null` باشد، به این معنی است که آن اشاره گر مورد استفاده قرار نگرفته و به چیزی اشاره نمی کند.

✓ اشاره گر `null` معادل `false` هم است.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 int *p=NULL;
 if (!p) cout<<"P is null";
 system("pause");
 return 0;
}
```

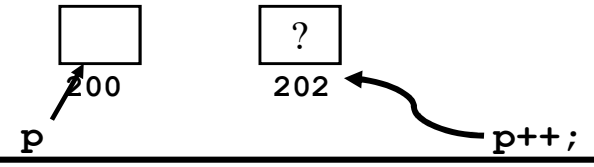
اشاره‌گر void

اشاره گر `void` با استفاده از `*void` تعریف می شود. این اشاره گر به هر نوع شیء می تواند اشاره کند. در صورتی می توان یک اشاره گر را به اشاره گر دیگری نسبت داد که نوع داده آنها با هم سازگار باشند. اما بدون نیاز به استفاده از قالب بندی نوع هر اشاره گری را می توان به یک اشاره گر `void` نسبت داد.

```
int *p;
void *v;
v=p;
```

اعمال حسابی روی اشاره گرها

فقط روی مقادیر صحیح می توان +، ++، -، -- را انجام داد. این اعمال با توجه به نوع مبنای هر اشاره گر انجام می گیرد.



```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 int *p;
 int q;
 system("cls");
 p=&q;
 cout<<p<<"\n";
 p++;
 cout<<p;
 system("pause");
 return 0;
}
```

output
0041F828
0041F82CPress any key to continue . . .

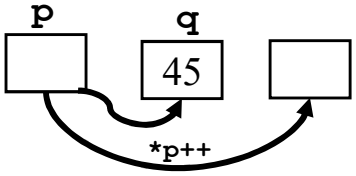
output
003FF808 003FF7F0 003FF7D4
003FF80C 003FF7F4 003FF7DC
Press any key to continue . . .

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 int *pi,i;
 float *pf,f;
 double *pd,d;
 pi=&i; pf=&f; pd=&d;
 cout<<" "<<pi<<" "<<pf<<" "<<pd<<"\n";
 pi++; pf++; pd++;
 cout<<" "<<pi<<" "<<pf<<" "<<pd<<"\n";
 system("pause");
 return 0;
}
```

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 int q, *p;
 q=45;
 p=&q;
 cout<<"\np is:"<<p;
 cout<<"\n*p is:"<<*p;
 *p++;
 cout<<"\np is:"<<p;
 cout<<"\n*p is:"<<*p;
 cout<<"\nq is:"<<q;
 system("pause");
 return 0;
}
```

نکته

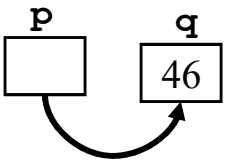
output
p is:0017FF08
***p is:45**
p is:0017FF0C
***p is:-858993460**
q is:45Press any key to continue . . .



```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 int q, *p;
 q=45;
 p=&q;
 cout<<"\np is:"<<p;
 cout<<"\n*p is:"<<*p;
 (*p)++;
 cout<<"\np is:"<<p;
 cout<<"\n*p is:"<<*p;
 cout<<"\nq is:"<<q;
 system("pause");
 return 0;
}
```

نکته

```
output
p is:0039FB38
*p is:45
p is:0039FB38
*p is:46
q is:46Press any key to continue . . .
```



اشاره گر و آرایه

❁ زمانیکه نام آرایه بدون اندیس استفاده شود، اشاره گری داریم که به ابتدای آرایه اشاره می کند.

```
int num[3]={1, 2, 3};
int *p;
p=num;
p=&num[0];
```

} Do the same thing.

❁ برای استفاده از یک اشاره گر جهت دستیابی به یک آرایه چندبعدی، باید همان عملیاتی را که کامپایلر به طور اتوماتیک انجام می دهد، انجام دهیم.

```
float b[10][5];
```

برای دسترسی به b[3][1] با اشاره گر باید از روش زیر استفاده کرد.

```
*(p+(3*5)+1)
```

شماره سطر * تعداد عناصر هر سطر + شماره ستون

اشاره گر و آرایه

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 int num[10]={1,2,3,4,5,6,7,8,9,10};
 int *p;
 p=num;
 cout<<*p<<" "<<*(p+1)<<" "<<*(p+2)<<"\n";
 cout<<num[0]<<" "<<num[1]<<" "<<num[2];
 system("pause");
 return 0;
}
```

output

1 2 3

1 2 3Press any key to continue . . .

کپی کردن محتویات یک رشته به ترتیب عکس ورودی با اشاره‌گر

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 char str1[]="Our class is advanced";
 char str2[80],*p1,*p2;
 p1=str1+strlen(str1)-1; //end of str1
 p2=str2;                //start of str2
 while(p1>= str1){
 *p2=*p1;
 p1--;
 p2++;}
 *p2='\0';        // null terminate str2
 cout<<"string1:"<<str1<<"string2:"<<str2;
 system("pause");
 return 0;
}
```

کپی کردن محتویات یک رشته به ترتیب عکس ورودی با اشاره‌گر

```
output
string1:Our class is advanced
string2:decnavda si ssalc ruO
Press any key to continue . . .
```

- در ابتدا p1 به انتهای رشته str1 و p2 به ابتدای رشته str2 اشاره می کنند.
- در حلقه از این شرط استفاده شده است که هرگاه p1 به ابتدای str1 اشاره کرد، عملیات کپی خاتمه پیدا کند.
- برای مشخص نمودن انتهای رشته از کاراکتر مربوطه '\0' استفاده می شود.

آرایه ای از اشاره گرها

برای ساختن جداول رشته ای هم به کار می رود.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
char *p[]={
"Disk error\n",
"Out of range\n",
"Paper out\n",
"Disk full\n"};
void error(int num);
int main()
{
 int i;
 for(i=0;i<4;i++)
    error(i);
 system("pause");
 return 0;
}
void error(int num)
{
cout<< p[num];
}
```

```
output
Disk error
Out of range
Paper out
Disk full
Press any key to continue . . .
```

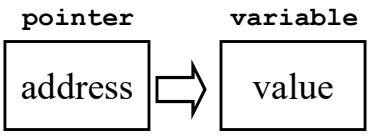
اشاره گر به جای ایندکس کردن آرایه ها

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 int i;
 char s[]="test in";
 for(i=0; s[i];i++)
 {
  s[i]=toupper(s[i]);
  cout<<s[i];
 }
 system("pause");
 return 0;
}
```

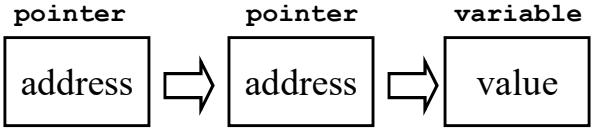
```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 char s[]="test in";
 char *p;
 p=s;
 while(*p)
 {
  *p=toupper(*p);
   cout<<*p;
   p++;
 }
 system("pause");
 return 0;
}
```

رجوع غیر مستقیم چندگانه multiple indirection

اشاره گری که به اشاره گر دیگری اشاره کند.



Single Indirection



Multiple Indirection

multiple indirection

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
 char **mp,*p,ch;
 system("cls");
 p=&ch;
 mp=&p;
 **mp='A';
 cout<<"ch:"<<ch;
 cout<<"\n*p:"<<*p;
 cout<<"\n**mp:"<<**mp;
 system("pause");
 return 0;
}
```

output

```
ch:A
*p:A
**mp:APress any key to continue . . .
```

توابعی که اشاره گر برمی گردانند

❁ اگر نوع بازگشتی یک تابع از نوع اشاره گر باشد در آن صورت مقدار بکار رفته در دستور return آن نیز باید یک اشاره گر باشد.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
char *findblank(char blank, char *s);
int main()
{
 char *p;
 p=findblank(' ',"This is a Test");
 if(p)cout<<p;
 system("pause");
 return 0;
}
char *findblank(char blank, char *s)
{
 while(blank !=*s && *s) s++;
 return s;
}
```

output

```
is a TestPress any key to continue . . .
```

ارسال اشاره گر به تابع

۱- پارامتر تابع از نوع اشاره گر باشد.
۲- درآرگومان تابع به جای یک مقدار از آدرس استفاده می شود.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void tavan2(double *a);
int main()
{
 double x=14.4;
 cout<< "x is:"<<x;
 tavan2(&x);
 cout<<"\nx squared:"<<x<<"\n";
 system("pause");
 return 0;
}
void tavan2(double *a)
{
 *a= *a * *a;
}
```

زمانیکه به جای مقدار اشاره‌گر به تابع ارسال شود، هر چه را که اشاره‌گر به آن اشاره می کند، تغییرخواهد کرد.

فراخوانی با رفرنس

- پارامتر تابع از نوع رفرنس (&) باشد.
- به تابع آدرس آرگومان ارسال می شود.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void tavan2(double &a);
int main()
{
 double x=14.4;
 cout<< "x is:"<<x;
 tavan2(x);
 cout<<"\nx squared:"<<x<<"\n";
 system("pause");
 return 0;
}
void tavan2(double &a)
{
 a= a * a;
}
```

توابعی که آدرس برمی گردانند

```
#include "stdafx.h"
#include <iostream>
using namespace std;
double &f();
double val=13.5;
double hist[20];
int top=0;
int main()
{
 f()=14;   f()=145;
 f()=132;  f()=124;
 for(int i = 0; i < top; i++)
     cout<<hist[i]<<endl;
 system("pause");
 return 0;
}
double &f()
{
    hist[top++] = val;
    return val;
}
```

```
hist[top++] = val;
val=14;
hist[top++] = val;
val=145;
hist[top++] = val;
val=132;
hist[top++] = val;
val=124;
```

output

13.5
14
145
132
Press any key to continue . . .

اشاره گر به ساختار

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

using namespace std;
struct students
{
    char name[40];
    int  age;
};
int main()
{
    students *st1,st2;
    st1=&st2;
    cin>>st1->name>>st1-> age;
    if(st1->age==18)
        cout<<st1->name;
    system("pause");
    return 0;
}
```

❖ برای دسترسی به عناصر یک ساختار با اشاره گر از عملگر <- (arrow operator) استفاده می‌گردد.

تخصیص حافظه پویا

ذخیره سازی اطلاعات در حافظه به دو صورت است:

۱- استفاده از متغیرها

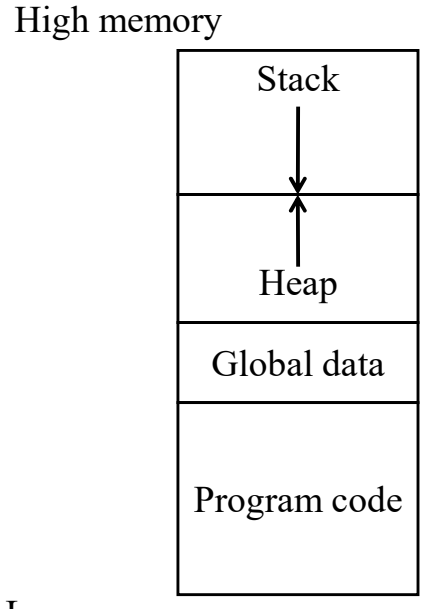
فضای ذخیره سازی که برای هر متغیر در نظر گرفته می شود، در زمان کامپایل ثابت است و در طول اجرای برنامه تغییر نمی کند.

۲- تخصیص حافظه پویا(دینامیک)

در این حالت برنامه می تواند متغیرهایی را که نیاز دارد در طول اجرای خود بسازد و یا رها (آزاد) نماید.

در این روش فضای ذخیره سازی یک داده در صورت نیاز از حافظه آزادی که بین برنامه و پشته (stack) قرار دارد تخصیص داده می شود. به این ناحیه heap گفته می شود.

تخصیص حافظه پویا



- ✓ فضای تخصیص یافته پویا در زمان اجرا مشخص می شود. معمولا تخصیص دهی پویا برای پیاده سازی ساختمان داده هایی مانند linked listed، binary trees، sparse arrays استفاده می شود.
- ✓ باید اطمینان حاصل شود، که حافظه پر نبوده باشد و حافظه تخصیص داده شده است.

تخصیص حافظه پویا

برای تخصیص دهی حافظه پویا از new استفاده می شود.

```
pointer= new type;
```

برای آزادسازی حافظه پویا از delete استفاده می شود. حافظه ای که آزاد شود را می توان توسط یک دستور new دیگر مجددا تخصیص دهی کرد.

```
delete pointer;
```

مثالی از تخصیص دهی و آزادسازی حافظه

```
#include "stdafx.h"
#include "iostream"
using namespace std;
int main()
{
 int *p;
 p= new int;
 if(!p){ // check if there is problem to allocate space
   cout<<"Allocation Error\n";
   return 1;
 }
 *p=100;
 cout<<*p;
 delete p;
 system("pause");
 return 0;
}
```

```
#include "stdafx.h"
#include "iostream"
using namespace std;
int main()
{
 int *p;
 p= new int(500);
 if(!p){ // check if there is problem to allocate space
   cout<<"Allocation Error\n";
   return 1;
 }
 cout<<*p;
 delete p;
 system("pause");
 return 0;
}
```

مقداردهی اولیه به صورت تخصیص پویا

تخصیص دهی آرایه ها

تخصیص دهی آرایه:

```
pointer= new type[size];
```

Size تعداد عناصر آرایه است.

```
delete[] pointer;
```

آزادسازی حافظه:

ذخیره تعدادی داده در آرایه و نمایش میانگین

```
#include "stdafx.h"
#include "iostream"
using namespace std;
int main()
{
 int *p;
 int n,i,s=0;
 cin>>n;
 p= new int[n];
 if(!p) { // check if there is problem to allocate
    cout<<"Allocation Error\n";
    return 1;}
 for(i=0; i<n;i++){
    cin>>p[i];
    s=s+p[i];}
 cout<<s/n;
 delete[ ] p;
 system("pause");
 return 0;
}
```

شی گرایی

☜ برنامه هایی که تا کنون نوشتید برنامه های ساخت یافته نامیده می شدند که دارای ساختارهای کنترلی well defined، بلوکهای کد، حداقل استفاده (عدم استفاده) از goto و procedure هایی که recursionو متغیرهای محلی را پشتیبانی می کردند.

☜این برنامه ها را procedural می نامند که دارای متغیرها و توابع هستند. اما در دنیای واقعی object وجود دارد نه متغیر.

☜ برنامه نویسی شی گرا (OOP) این امگان را می دهد که یک مساله به زیرگروه های به هم مرتبطی تجزیه شوند، هر کدام از این زیرگروه ها به یک شی تبدیل می شوندکه کدها و داده های مربوط به خود را دارند، پیچیدگی کاهش می یابد و برنامه نویس می تواند برنامه های بزرگتری را مدیریت نماید.

☜ object ها نقش اشیا را در دنیای واقعی دارند. همه زبانهای OOP در مفاهیم زیر مشترک هستند:

کپسوله سازی، پلی مورفیسم، ارث بری

شی گرایی

⇦ کپسوله سازی Encapsulation

☜ مکانیسمی است که کدها و داده هایی که این کدها با آنها کار می کنند، در کنار هم قرار داده و آنها را از تداخل های بیرونی و سوء کاربرد محفوظ می دارد. کدها و داده های ضروری را به همان طریقی که یک جعبه سیاه خودکفا ساخته می شود، می توان کنار هم قرار داد. زمانیکه کدها و داده ها به اینگونه کنار هم قرار می گیرند، یک شی ساخته می شود. Object مفهومی است که از کپسوله سازی پشتیبانی می کند.

☜ در یک شیء، کدها و دادها ممکن است نسبت به آن شیء private یا public باشند. کدهای خصوصی فقط برای دیگر بخشهای همان شیء شناخته شده و قابل دسترسی است. به این معنی که کد یا داده خصوصی نمی تواند از خارج شیء مورد دسترسی قرار گیرد. اما قسمت های دیگر برنامه می توانند به کد یا داده عمومی دسترسی داشته باشند. به این عمل data hiding گفته می شود.

☜ هر زمان که یک کلاس تعریف می شود، یک نوع داده جدید ایجاد می شود و هر مورد از این نوع داده یک متغیر مرکب (شیء) است.

شی گرایی

⇦ پلی مورفیسم (چندریختی) Polymorphism

☜ با این قابلیت می توان چندتابع با یک نام با تعداد، ترتیب، نوع و ترکیب پارامترهای متفاوت داشت. این خاصیت به کاهش پیچیدگی برنامه کمک می کند. کامپایلرخود با توجه به شرایط، عمل(تابع) مورد نظر را انتخاب می کند. (نمی توان تنها با نوع بازگشتی متفاوت سربارگذاری توابع را انجام داد.)

☜ اگر در توابع هم نام فقط نوع و تعداد پارامترها متفاوت باشد، سربارگذاری توابع (operator overloading) می گویند.

```
int  abs(int a);
int  abs(int a , int b);
long  abs(long a);
float abs(float a);
```

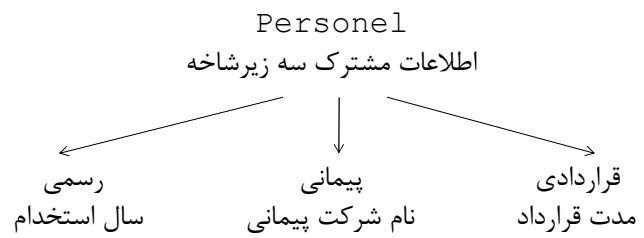
☜unique signature

علائمی که باعث می شوند دو تابع توسط کامپایلر متمایز باشند. تعداد، ترتیب، نوع و ترکیب پارامترهای متفاوت

شی گرایی

⇦ ارث بری Inheritance

☜ یک کلاس جدید می تواند خواص (properties) کلاس دیگر را به ارث ببرد. در واقع اشیا می توانند داده ها و توابع عضو اشیا دیگر را به ارث ببرند و به دیگر خصوصیات منحصر به خود اضافه نمایند.



class

کلاس ها ارکان برنامه نویسی شی گرا را تشکیل می دهند. از کلاس برای ساخت object استفاده می شود.

ساختار کلاس به صورت زیر می باشد:

```
class name{
private functions and variables
public:
public functions and variables
}objects;
```

همانند ساختارها ذکر کردن نام اشیا اختیاری است و می توان بعدا آنها را معرفی کرد. به طور پیش فرض همه توابع و متغیرهای کلاس خصوصی هستند مگر آنکه از public پیش از معرفی آنها استفاده شود.

class

نمونه ای از یک کلاس:

```
class myclass{
int a;
public:
void seta(int x);
int calca();
};
```

a یک متغیر private است پس توسط هیچ کدی خارج از myclass قابل دستیابی نمی باشد.

متغیر خصوصی a و دو تابع عمومی seta و calca.

توابعی که به عنوان جزئی از یک کلاس معرفی می شوند توابع عضو (member functions) می گویند.

به متغیری که از کلاس گرفته می شود، object می گویند.

class

تکمیل شده کلاس فوق:

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class myclass{
int a;
public:
void seta(int x){a=x;}
int calca(){return a*a;}
};
int main(){
    myclass test;
    int k;
    cin>>k;
    test.seta(k);
    cout <<test.calca();
    system("pause");
    return 0;
}
```

class

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class myclass{
int a;
public:
void seta(int x);
int calca();
};
void  myclass::seta(int x){a=x;}
int  myclass:: calca(){return a*a;}
int main(){
    myclass test;
    int k;
    cin>>k;
    test.seta(k);
    cout <<test.calca();
    system("pause");
    return 0;
}
```

روش دوم:

عملگر تعیین میدان
scope resolution operator ::

class

۱- **توابع و متغیرهای درون یک کلاس member نامیده می شوند.**

۲- **یک شیء در حافظه فضا اشغال می کند اما یک کلاس که یک نوع داده جدید است فضایی اشغال نمی کند.**

۳- **هر شیء یک کلاس از تمام متغیرهای عادی معرفی شده در آن کلاس یک کپی مخصوص به خود خواهد داشت.**

class

❖سازنده ها در کلاس

- ✓ می توان در معرفی یک کلاس تابع سازنده (constructor function) قرار داد، هر بار که یک شیء از کلاس ساخته می شود، تابع سازنده به طور خودکار فراخوانی می شود.
- ✓ از این تابع می توان برای مقداردهی اولیه لازم روی شیء استفاده کرد. این تابع باید در قسمت public نوشته شود.
- ✓ تابع سازنده هم نام با نام کلاس خود است و نوع مقدار بازگشتی هم ندارد.
- ✓ این تابع می تواند پارامتر ورودی داشته باشد.
- ✓ این تابع اختیاری است و در صورت نیاز می توان توابع سازنده مختلفی با تعداد و نوع پارامترهای متفاوت داشت.

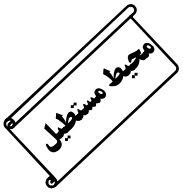
class

❖مخرب ها در کلاس

- ✓ مخرب ها مکمل سازنده ها هستند.
- ✓ زمانی که قرار است شیئی از بین برود، این تابع فراخوانی می شود. این تابع باید در قسمت public نوشته شود.
- ✓ شیئی که زمان ایجاد شدن خود فضایی را از حافظه به خود اختصاص داده است، زمانی که باید از بین برود، باید آن حافظه را آزاد نماید. این تابع به طور خودکار این کار را انجام می دهد.
- ✓ تابع مخرب هم نام با نام کلاس خود است با این تفاوت که پیش از نام خود علامت ~ را دارد و نوع مقدارباز‌گشتی هم ندارد.
- ✓ این تابع پارامتر ورودی ندارد. (تخصیص حافظه پویا)
- ✓ این تابع اختیاری است اما در صورت وجود فقط و فقط یک تابع مخرب می تواند وجود داشته باشد.

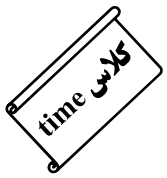
class

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class rectangle{
int x,y;
public:
 rectangle(int a, int b);
 ~rectangle();
 int show();
};
rectangle::rectangle(int a, int b){ x=a;  y=b;}
rectangle:: ~rectangle(){cout<<"no free allocation";}
int  rectangle:: show(){return (x+y)*2;}
int main(){
    int m,n;
    cin>>m>>n;
    rectangle ob(m,n);
    cout<<ob.show();
    system("pause");
    return 0;
}
```



class

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class time{
public:
    time(int h, int m, int s);
    void reset();
    void show();
private:
    int hour, min, sec;};
time::time(int h,int m,int s){hour=h;min=m;sec=s;}
void time::reset(){hour=0; min=0; sec=0;}
void time::show(){
cout<<"time is:\n\t"<<hour<<":"<<min<<":"<<sec<<"\n";
}
int main()
{
    int saat,dagh,san;
    cin>>saat>>dagh>>san;
    time o(saat,dagh,san);
    o.show();
    system("pause");
    return 0;
}
```



class

تخصیص حافظه پویا

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class myclass{
int *p;
public:
 myclass();
 ~myclass();
 void  show(int x);
};
myclass::myclass(){
      p=new int;
     if (!p){cout<<"error"; exit(1);}
}
myclass::~myclass(){delete p;}
void  myclass:: show(int x){*p=x;  cout<<++(*p);}
int main(){
    myclass  ob;
    int k;
    cin>>k;
    ob.show(k);
    system("pause");
    return 0;
}
```

class

❖توابع inline

- ✓ توابعی که داخل بدنه کلاس تعریف می شوند inline درنظر گرفته می شود.
- ✓ توابع inline توابعی هستند که در هر نقطه ای که فراخوانی شوند، در اجرا کد آنها قرار داده می شود و فراخوانی نمی شوند.
- ✓ توابع inline سریعتر از توابع عادی اجرا می شوند اما حجم برنامه را افزایش می دهند.
- ✓ این توابع فقط برای توابع کوتاه استفاده می شوند.
- ✓ در صورتی که تابع کلاس خارج از بدنه کلاس تعریف شود، همانند توابع inline معمولی، کلمه inline پیش از نوع بازگشتی تابع قرار می گیرد.

class

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class rectangle{
int x,y;
public:
 rectangle(int a, int b);
 ~rectangle();
 int show();
};
rectangle::rectangle(int a, int b){ x=a;  y=b;}
rectangle:: ~rectangle(){cout<<"no free allocation";}
inline int rectangle:: show(){return (x+y)*2;}
int main(){
    int m,n;
    cin>>m>>n;
    rectangle ob(m,n);
    cout<<ob.show();
    system("pause");
    return 0;
}
```

class

❖ اشاره گر به اشیا

- ✓ اشاره گر به شی دقیقا مانند اشاره گر به هر متغیر دیگر است.
- ✓ برای به دست آوردن آدرس (&) و محتوا (*) نیز همانند قبل عمل می شود (مانند اشاره گر به ساختار)
- ✓ برای دسترسی به عناصر کلاس در این حالت از عملگر <- استفاده می شود.

class

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class test{
  int i, j ;
public:
  test(int n, int m);
  int meanij();
};
test::test(int n, int m){
i=n;
j=m;
}
int test::meanij(){
return (i+j)/2;
}
int main(){
    int a,b;
    cin>>a>>b;
    test O1(a,b);
    test *p;
    p=&O1;
    cout<<O1.meanij()<<"\n";
    cout<< p->meanij();
    system("pause");
    return 0;
}
```

class

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class zarb{
int x,y;
public:
 zarb(int a, int b);
int show();
};
zarb::zarb(int a, int b){ x=a;  y=b;}
int  zarb:: show(){return  x*y;}
int main(){
    int m,n;
    cin>>m>>n;
    zarb ob(m,n);
    zarb *p;
    p=&ob;
    cout<<ob.show()<<"\n";
    cout<<p->show();
    system("pause");
    return 0;
}
```

اشاره گر به شیء
ضرب دو عدد

class

❖ ساختار و کلاس

- ✓ تعریف ساختار و کلاس مشابه است.
- ✓ در کلاس اگر نوع اعضای کلاس مشخص نشود، private در نظر گرفته می شود.
- ✓ در ساختار اگر نوع اعضای ساختار مشخص نشود، public در نظر گرفته می شود.
- ✓ در ساختار هم می توان همانند کلاس توابع عضو داشت.
- ✓ این تغییرات پس از ظهور شیء گرایی در ساختار به وجود آمده است.

class

❖ ارث بری در کلاس

- ✓ زمانی که یک کلاس از کلاس دیگر به ارث می برد، به کلاسی که از آن ارث برده شده است کلاس مبنا (base class) و به کلاسی که از آن ارث می برد کلاس مشتق شده یا (derived class) گفته می شود.
- ✓ در مثال بعد کلاس D از کلاس B مشتق شده است. Public مشخص می کند که کلاس D به کونه ای از B به ارث می برد که همه اعضای عمومی کلاس مبنا، اعضای عمومی کلاس مشتق شده خواهند بود.

class

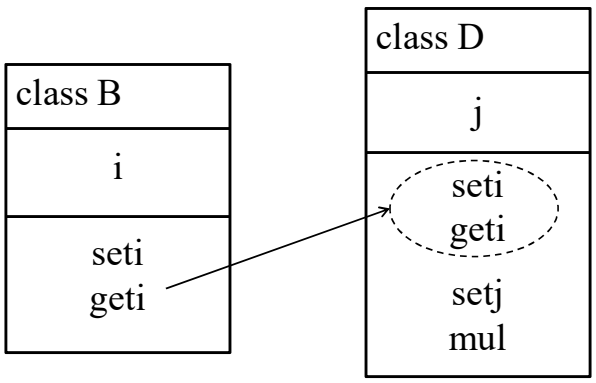
```
#include "stdafx.h"
#include <iostream>
using namespace std;
class B{
   int i;
public:
   void seti(int n);
   int geti();
};
class D: public B {
   int j;
public:
   void setj(int n);
   int mul();
};
void B::seti(int n) {i=n;}
int B::geti(){return i;}
void D::setj(int n){ j=n;}
int D::mul() {return j*geti();}
```

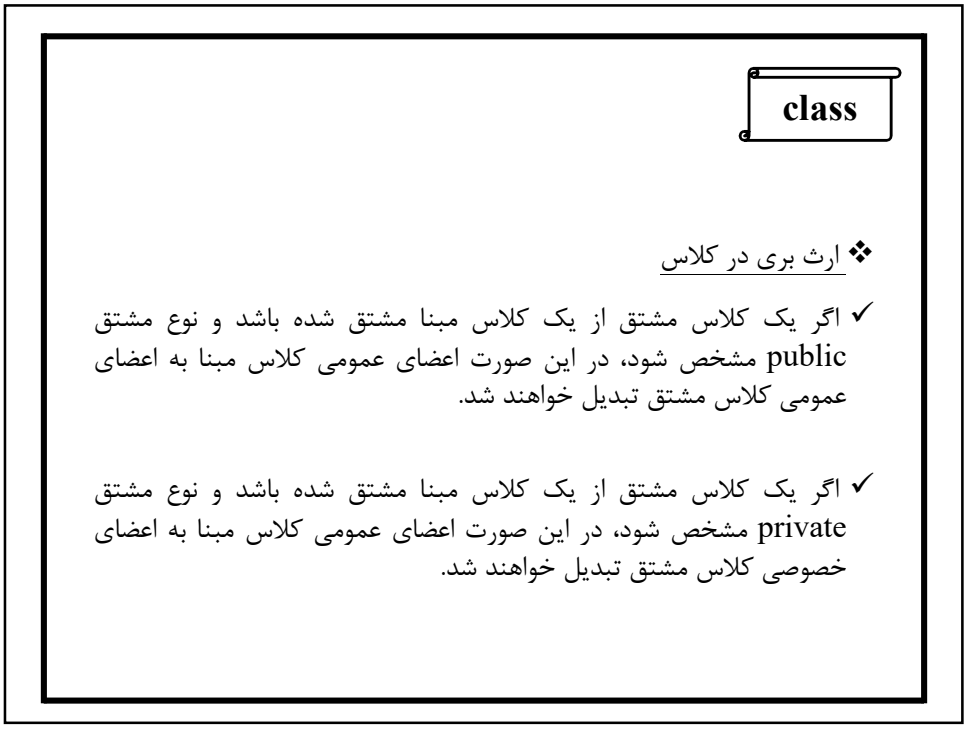
```
int main()
{
 D ob;
 ob.seti(20);
 ob.setj(4);
 cout<<ob.mul();
 system("pause");
 return 0;
}
```

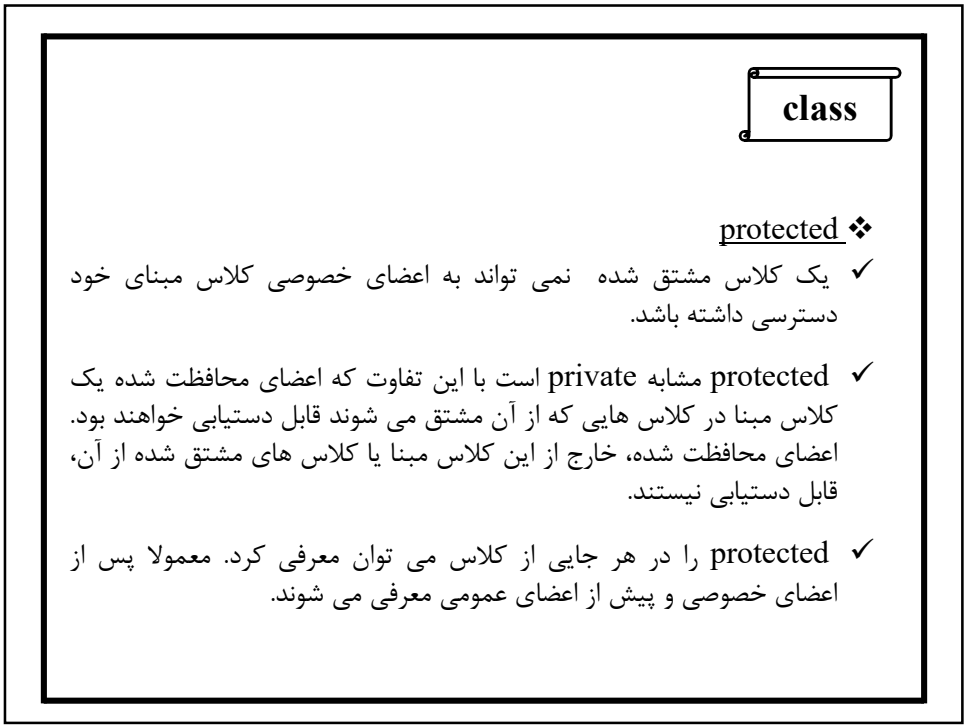
output

80Press any key to continue . . .

class




class
❖ ارث بری در کلاس
✓ اگر یک کلاس مشتق از یک کلاس مبنا مشتق شده باشد و نوع مشتق public مشخص شود، در این صورت اعضای عمومی کلاس مبنا به اعضای عمومی کلاس مشتق تبدیل خواهند شد.
✓ اگر یک کلاس مشتق از یک کلاس مبنا مشتق شده باشد و نوع مشتق private مشخص شود، در این صورت اعضای عمومی کلاس مبنا به اعضای خصوصی کلاس مشتق تبدیل خواهند شد.


class
❖ protected
✓ یک کلاس مشتق شده نمی تواند به اعضای خصوصی کلاس مبنای خود دسترسی داشته باشد.
✓ protected مشابه private است با این تفاوت که اعضای محافظت شده یک کلاس مبنا در کلاس هایی که از آن مشتق می شوند قابل دستیابی خواهند بود. اعضای محافظت شده، خارج از این کلاس مبنا یا کلاس های مشتق شده از آن، قابل دستیابی نیستند.
✓ protected را در هر جایی از کلاس می توان معرفی کرد. معمولا پس از اعضای خصوصی و پیش از اعضای عمومی معرفی می شوند.

class

❖ protected

- ✓ وقتی عضو محافظت شده ای از یک کلاس مبنا به صورت public، توسط یک کلاس مشتق به ارث برده می شود به عضو محافظت شده آن کلاس مشتق تبدیل می گردد. اگر کلاس مبنا به صورت private به ارث برده شود در آن صورت اعضای محافظت شده آن به اعضای خصوصی کلاس مشتق تبدیل خواهند شد.
- ✓ کلاس مبنا را یک کلاس مشتق می تواند به صورت protected نیز به ارث ببرد. در چنین مواردی اعضای عمومی و محافظت شده کلاس مبنا به اعضای محافظت شده کلاس مشتق شده تبدیل می شوند.

```
class classname{
//private members
protected:
//protected members
public:
//public members};
```

class

مثال

```
#include "stdafx.h"
#include <iostream>
using namespace std;

class base {
protected:
    int a,b;
public:
    void setab(int n, int m){a=n; b=m;}
};
class derived: public base {
    int c;
public:
    void setc(int n){c=n;}
    void showabc(){cout<<a<<"\n"<<b<<"\n"<<c;}
};
int main()
{
    derived ob;
    ob.setab(2,8);
    ob.setc(5);
    ob.showabc();
    system("pause");
    return 0;
}
```

class

❖ تابع های دوست friend functions

- ✓ زمانی که بخواهیم تابعی بتواند به اعضای خصوصی یک کلاس دسترسی داشته باشد بدون آنکه واقعا عضو کلاس باشد، از تابع دوست استفاده می شود.
- ✓ تابع دوست همانند یک تابع معمولی غیر عضو تعریف می شود. پیش از پیش الگوی تابع friend قرار داده می شود.
- ✓ در تعریف تابع کلمه friend قرار داده نمی شود.
- ✓ تابع های دوست می توانند با بیش از یک کلاس دوست شوند.

class

```
#include "stdafx.h"
#include <iostream>
using namespace std;
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;
}
int sum(myclass  x){
    return x.a + x.b;
}
int main()
{
    myclass n;
    n.set_ab(5,8);
    cout << sum(n);
    system("pause");
    return 0;
}
```

class

مثال

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class test{
  int i, j ;
public:
  test(int n, int m);
  friend int isfactor(test ob);
};
test::test(int n, int m){
    i=n; j=m;
}
int isfactor(test ob){
if  (!(ob.i %ob.j)) return 1;
else return 0;
}
```

```
int main()
{
    test o1(18,6),o2(13,6);
    if (isfactor(o1))
        cout<<" 6 is factor of 18\n";
     else
        cout<<" 6 is n't factor of 18\n";
     if (isfactor(o2))
        cout<<" 6 is factor of 13";
    else
        cout<<" 6 is n't factor of 13";
    system("pause");
    return 0;
}
```

class

❖ تابع های دوست friend functions

✓ تابع دوست تنها در پیوند با شیئی که داخل وی معرفی می شود، یا به آن ارسال می گردد، می تواند به این اعضا دسترسی داشته باشد.

✓ فراخوانی تابع دوست مانند تابع عضو کلاس غلط است.

```
O1.isfactor();
```

✓ تابع isfactor تابع دوست است پس می تواند به متغیرهای خصوصی دسترسی پیدا کند.

✓ کلاسی که از کلاس دیگر ارث می‌برد، توابع دوست آن کلاس را به ارث نمی‌برند. به عبارت دیگر یک (کلاس مشتق) derived class، توابع دوست را به ارث نمی‌برد.

class

❖ اشاره گر this

- ✓ اشاره گر ویژه ای مختص کلاس ها است.
- ✓ this اشاره گر ویژه ای است که وقتی یک تابع عضو فراخوانی می شود، به طور اتوماتیک به آن ارسال می گردد و به شیئی اشاره می کند که آن فراخوانی را تولید نموده است. یک تابع دوست دارای اشاره گر this نیست.
- ✓ اگر ob یک شی باشد:

```
ob.f1( );
```

- ✓ در اینجا به طور اتوماتیک به تابع f1( ) اشاره گری به نام this ارسال می شود که به شیء ob اشاره خواهد کرد.

class

مثال

```
#include "stdafx.h"
#include <iostream>
using namespace std;

class  invent{
    char item[20];
    double cost;
    int available;
public:
    invent(char *i, double c, int o){
        strcpy(item, i);
        cost=c;
        available=o;}
    void show(); };
void  invent:: show(){
    cout<<item<<"\n "<<cost;}

int main()
{
    invent ob("coca",5.5,8);
    ob.show();
    system("pause");
    return 0;
}
```

class

مثال با this

```
#include "stdafx.h"
#include <iostream>
using namespace std;

class  invent{
    char item[20];
    double cost;
    int available;
public:
    invent(char *i, double c, int o){
        strcpy(this->item, i);
        this->cost=c;
        this->available=o;}
    void show(); };
void  invent:: show(){
    cout<<this->item<<"\n"<<cost;}

int main()
{
    invent ob("coca",5.5,8);
    ob.show();
    system("pause");
    return 0;
}
```

class

❖ کلاس های دوست friend classes

✓ یک کلاس می تواند دوست کلاس دیگری باشد. در این حالت تابع کلاس دوست به کلیه اسامی private تعریف شده در کلاس دیگر دسترسی دارد.

✓ در این مثال کلاس دوست friendclass در کلاس CPP_Test تعریف شده است.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

class CPP_Test{
    int  privatedata;
    friend class friendclass;
public:
    CPP_Test(){privatedata = 5;}
};
class friendclass{
public:
    int subtract(int x){
        CPP_Test ob2;
        return ob2.privatedata - x;}
};
int main()
{
    friendclass ob3;
    cout << ob3.subtract(2);
    system("pause");
    return 0;
}
```

class

❖ تابع های مجازی virtual functions

- ✓ توابع مجازی جهت پشتیبانی از پلی مورفیسم در زمان اجرای برنامه استفاده می شود.
- ✓ در C++ به دو روش از پلی مورفیسم پشتیبانی می شود:

  ۱- در زمان کامپایل از طریق توابع و سربارگذاری

  ۲- در زمان اجرا از طریق به کار بردن توابع مجازی
- ✓ تابع مجازی، تابعی است که در یک کلاس مبنا تعریف شده و توسط کلاسی که از آن کلاس مبنا مشتق خواهد شد مجددا تعریف می شود (تغییر داده می شود).

class

❖ تابع های مجازی virtual functions

- ✓ برای معرفی تابع مجازی از کلمه virtual استفاده می شود.
- ✓ کلاس مشتق شده تابع مجازی را با توجه به نیازهای خود مجددا تعریف می کند. تابع های مجازی فلسفه یک رابط چند کار که همان مفهوم پلی مورفیسم است پیاده سازی می کند.
- ✓ هر بار که تابع مجازی فراخوانی می شود، نوع شیئی که به آن اشاره می کند مشخص می کند که کدام نسخه تابع مجازی استفاده شود.
- ✓ کلاسی که شامل یک تابع مجازی باشد کلاس پلی مورفیک (polymorphic class) گویند.

class

```
#include "stdafx.h"
#include <iostream>
using namespace std;

class base {
public:
    int i;
    base (int x){i=x;}
    virtual void func(){cout<<"\n in base :"<<i;}
};
class derived1:public base{
public:
    derived1 (int x): base(x){}
    void func(){cout<<"\n in derived1: "<<i*i;}
};
class derived2:public base{
public:
    derived2 (int x) : base(x){}
    void func(){cout<<"\n in derived2: "<<i+i;}
};
```

```
int main()
{
    base *p;
    base ob(10);
    derived1 ob1(10);
    derived2 ob2(10);
    p=&ob;
    p->func();
    p=&ob1;
    p->func();
    p=&ob2;
    p->func();
    system("pause");
    return 0;
}
```

file

فایل

☜ برای دریافت اطلاعات و یا نوشتن اطلاعات در محلی به جز console از فایل ها استفاده می شود. در برنامه های قبل داده ها در RAM ذخیره می شدند که با قطع جریان برق و خاتمه برنامه از بین می روند. برای ذخیره دائمی ورودیها و خروجیها از فایل استفاده می شود.

☜ برای انجام عملیات ورودی خروجی در فایل از کتابخانه fstream استفاده می‌شود.

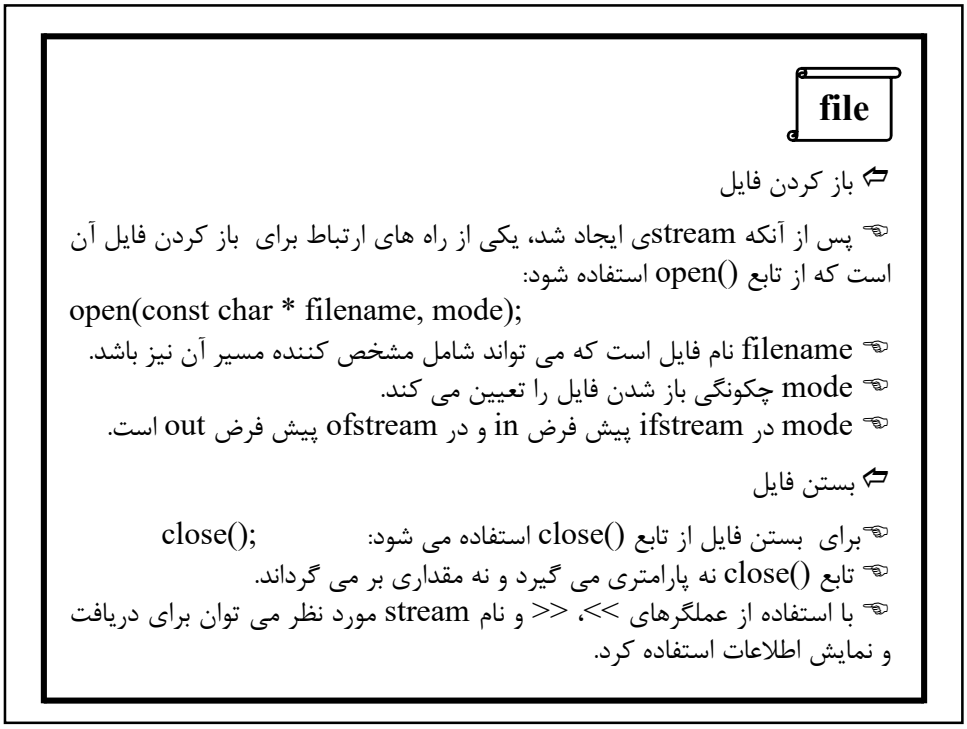
☜ این فایل کلاس های

ifstream (ورودی)،

ofstream (خروجی)،

fstream (ورودی و خروجی)

☜را تعریف می کند.


file
باز کردن فایل
پس از آنکه streamی ایجاد شد، یکی از راه های ارتباط برای باز کردن فایل آن است که از تابع ()open استفاده شود:
open(const char * filename, mode);
filename نام فایل است که می تواند شامل مشخص کننده مسیر آن نیز باشد.
mode چکونگی باز شدن فایل را تعیین می کند.
mode در ifstream پیش فرض in و در ofstream پیش فرض out است.
بستن فایل
برای بستن فایل از تابع ()close استفاده می شود: close();
تابع ()close نه پارامتری می گیرد و نه مقداری بر می گرداند.
با استفاده از عملگرهای >>، << و نام stream مورد نظر می توان برای دریافت و نمایش اطلاعات استفاده کرد.


file
mode می تواند از حالات زیر باشد.
می توان چند تا از مدها را با هم OR (|) کرد.
ios::in
از آن فایل به عنوان ورودی استفاده می شود.
ios::out
از آن فایل به عنوان خروجی استفاده می شود.
ios::binary
فایل به صورت باینری باز می شود.(پیش فرض مد متن است)
ios::ate
درزمان بازشدن یک فایل، نشانگر آن فایل به انتهای آن برده می شوداما درهر نقطه ای ازفایل عملیات ورودی خروجی ممکن است.
ios::app
همه داده های خروجی که به آن فایل فرستاده می شوند به انتهای آن افزوده می شود. تنها برای ارتباط با فایل هایی استفاده می شود که توانایی دریافت خروجی را داشته باشد.
ios::trunc
محتویات قبلی فایل پاک می شود و فایل تازه ای به طول صفر شناخته می شود.
ios::nocreate
اگر فایل موردنظر وجود نداشته باشد، فایل تازه ای ایجاد نمی کند و با شکست مواجه می شود.
ios::noreplace
اگر فایل موردنظر وجود داشته باشد، فایل تازه ای ایجاد نمی کند و با شکست مواجه می شود.

file

```
#include "stdafx.h"
#include <iostream>
#include<fstream>
using namespace std;
int main()
{
    ofstream fileout1("test");
    if (!fileout1){
        cout<<"can not open output file\n";
        return 1; }
    fileout1<<"Hello!!!!!\n";
    fileout1<<100;
    fileout1.close();
    ifstream filein1("test");
    if (!filein1){
        cout<<"can not open input file\n";
        return 1;}
    char str[80];
    int i;
    filein1>>str>>i;
    cout<< str<<i;
    filein1.close();
    system("pause");
    return 0;
}
```

ایجاد فایل متنی

file

⇦ مد متن و مد باینری

☜ به طور پیش فرض همه فایل ها در مد text (متن) باز می شوند. در این مد ممکن است تبدیلات کاراکتری گوناگونی رخ دهد از جمله کاراکترهای carriage return و line feed هر دو به newline تبدیل می شوند. در مد binary (دودویی) چنین تبدیلاتی رخ نمی دهد.

☜ هر فایلی می تواند به هر دو صورت باز شود.

☜ eof() تابعی است که اگر به انتهای فایل برسیم مقدار true (غیرصفر) برمی گرداند در غیراینصورت مقدار false (صفر) برمی گرداند.

⇦ مد باینری

☜ از تابع put برای نوشتن یک بایت (کاراکتر) در فایل خروجی استفاده می شود.

```
ostream &put(char ch);
```

☜ از تابع get برای خواندن یک بایت (کاراکتر) از فایل ورودی استفاده می شود.

```
istream &get(char &ch);
```

file

ایجاد فایل، باینری

```
#include "stdafx.h"
#include <iostream>
#include<fstream>
using namespace std;
int main()
{
    ofstream fileout1("test1");
    if (!fileout1){
        cout<<"can not open output file\n";
        return 1; }
    fileout1<<"Hello!!!! \n This is a test ";
    fileout1<<100;
    fileout1.close();
    ifstream filein1("test1",ios::binary);
    char ch;
    if (!filein1){
        cout<<"can not open input file\n";
        return 1; }
    while(!filein1.eof()){
        filein1.get(ch);
        cout<<ch;
    }
    filein1.close();
    system("pause");
    return 0;
}
```

file

برنامه ای بنویسید که اطلاعات نام خانوادگی و سن ۵ کارمند را دریافت کرده در یک فایل ذخیره نمایید. سپس اطلاعات را از فایل خوانده و نمایش دهد.

```
#include<fstream>
using namespace std;
struct karmand{
char family[20];
int age;
}kar[5];
int main(){
    int i;
    char familyk[20];
    int agek;
    ofstream fileout1;
    fileout1.open("D:\\karmand");
    if (!fileout1){
        cout<<"can not open\n";
        return 1; }
    for(i=0; i<5; i++){
        cin>>kar[i].family>>kar[i].age;
        fileout1<<kar[i].family<<"\n";
        fileout1<<kar[i].age<<"\n";   }
    fileout1.close();
    ifstream filein1("D:\\karmand",ios::binary);
    if (!filein1){
        cout<<"can not open \n";
        return 1;}
    while(!filein1.eof())
        { filein1>>familyk>>agek;
          cout<<familyk<<agek;  }
    filein1.close();
    system("pause");
    return 0; }
```

توابع کلی (Generic)

یک تابع generic مجموعه کلی از عملیات را تعریف می‌کند که بر روی انواع گوناگون داده ها اعمال خواهند شد. تابع ژنریک دارای همان نوع داده ای است که به صورت پارامتر به آن ارسال شده و روی آن کار خواهد کرد. با این روش می‌توان یک روتین کلی را روی دامنه گسترده‌ای از داده ها اعمال کرد.

بسیاری از الگوریتم ها بدون توجه به نوع داده ای که در آنها به کار گرفته می‌شود از لحاظ منطقی یکی هستند. مثلا الگوریتم مرتب سازی سریع (Quicksort) را می توان در مورد آرایه ای از مقادیر صحیح یا در مورد آرایه ای از مقادیر اعشاری استفاده نمود. در این دو تنها چیزی که فرق می‌کند نوع داده هایی است که قرار است مرتب شوند.

با ساختن یک تابع *کلی* می‌توان رفتار یک الگوریتم را مستقل از هر نوع داده ای تعریف کرد. هنگامی که آن تابع را اجرا می‌کنید کامپایلر خود برای آن نوع داده ای که در هنگام فراخوانی مورد استفاده قرار گرفته است، کدهای متناسب با نوع داده مورد نیاز را به طور اتوماتیک تولید می‌کند.

توابع کلی (Generic)

با ساختن تابع ژنریک، تابعی ایجاد می شود که به طور اتوماتیک توسط کامپایلر سربارگذاری می شود.

شکل کلی تعریف یک تابع به صورت template :

```
template <class Type> re-type func-name(parameter-list)
{
   // body
}
```

type عنوان موقت محلی برای نوع داده ای است که توسط این تابع مورد استفاده قرار خواهد گرفت. از این نام می توان داخل بدنه تابع استفاده کرد. اما این نام فقط یک جا نگهدار است و کامپایلر وقتی نسخه به خصوص از تابع ایجاد کند آن را به طور اتوماتیک با یک نوع داده استفاده شده در فراخوانی جایگزین خواهد نمود.

توابع کلی (Generic)

برنامه ای بنویسید که یک تابع generic ایجاد کند. این تابع مقادیر دو متغیری که به عنوان آرگومان ارسال می‌شوند با هم تعویض می کند.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
template <class X> void swapping(X &a, X &b)
{
    X temp;
    temp = a;
    a = b;
    b = temp;
}
int main()
{
    int i = 10, j = 20;
    float x = 5.1, y = 2.5;
    cout << i << ' ' << j << "\n";
    cout << x << ' ' << y << "\n";
    swapping(i, j);
    swapping(x, y);
    cout << i << ' ' << j << "\n";
    cout << x << ' ' << y << "\n";
    system("pause");
    return 0;
}
```

output

```
10 20
5.1 2.5
20 10
2.5 5.1
Press any key to continue . . .
```

توابع کلی (Generic)

- ✓ زمانی که کامپایلر نسخه به خصوصی از تابع template می سازد، یک تابع تولید شده (generated function) ساخته شده است.
- ✓ به عمل تولید یک تابع (instantiating) گفته می شود. یک تابع تولید شده مورد خاصی از یک تابع کلی است.
- ✓ با به کار بردن لیستی از پارامتر ها که با کاما از هم جدا شده باشند در یک دستور template می توان بیش از یک نوع داده کلی را مشخص ساخت.

توابع کلی (Generic)

برنامه زیر یک تابع generic ایجاد می‌کند که دارای دو نوع داده کلی است.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

template <class type1, class type2>
void myfunc(type1 x, type2 y)
{
    cout << x << ' ' << y << "\n";
}
int main()
{
    myfunc(10, "hi");
    myfunc(0.53, 't');
    system("pause");
    return 0;
}
```

output
10 hi
0.53 t
Press any key to continue . . .

کلاس های Generic

علاوه بر توابع generic می‌توان کلاس‌های generic تعریف کرد. با تعریف کلاس generic در واقع کلاسی می سازیم که همه الگوریتم های مورد استفاده اش را تعریف می‌کند، اما انواع واقعی داده هایی که قرار است با آنها سر و کار داشته باشد در زمان ساخته شدن اشیایی از نوع اطلاعات به صورت پارامتر مشخص خواهند شد.

کلاس‌های generic زمانی سودمند هستند که آن کلاس شامل منطقی قابل تعمیم باشد. به عنوان مثال همان الگوریتمی که زنجیره‌ای از مقادیر صحیح را اداره می‌کند، با زنجیره‌ای از کاراکترها نیز کار خواهد کرد. وقتی یک شیء از این نوع کلاس ها ساخته می شود کامپایلر با توجه به نوع داده که ما در آن موقع مشخص ساخته ایم به طور اتوماتیک نوع درستی از آن شیء تولید خواهد کرد. شکل کلی معرفی یک کلاس generic به صورت زیر است:

template <class Type> class class-name{
}

type عنوان جا نگهدارنده است که در زمان نمونه‌سازی از آن کلاس مشخص می‌شود.

کلاس های Generic

در یک کلاس generic می توان با استفاده از یک لیست بیش از یک نوع داده کلی تعریف کرد. توابع عضو کلاس generic خودشان به طور اتوماتیک generic خواهند بود و نیازی نیست که صراحتاً با استفاده از template مشخص شده باشند. بلافاصله پس از آنکه یک کلاس generic ساخته می شود می توان با استفاده از شکل کلی زیر یک مورد خاص از آن کلاس ایجاد کرد. در اینجا type نام نوع آن داده است که این کلاس روی آن عمل خواهد کرد:

class-name <type> ob;

کلاس generic با تابع getmax

```
template <class T>
class mypair {
    T a, b;
public:
    mypair(T first, T second)
    {
        a = first; b = second;
    }
    T getmax();
};

template <class T>
T mypair<T>::getmax()
{
    T retval;
    retval = a>b ? a : b;
    return retval;
}

int main() {
    mypair <int> myobject(100, 75);
    cout << myobject.getmax();
    system("pause");
    return 0;
}
```

Exception handling

با استفاده از روش رسیدگی به استثنا (Exception handling) راحت‌تر می‌توان به خطاهای زمان اجرا پاسخ داده و آنها را مدیریت نمود.

روش رسیدگی به استثنا ها در C++ بر اساس سه کلیدواژه بنا نهاده شده است: try، catch و throw. آن دستورات از برنامه که می خواهید برای احتمال بروز استثناهایی در آنها مورد بازبینی قرار بگیرند در یک بلوک try قرار داده می شود. اگر داخل بلوک try یک استثنا (یک خطا) روی دهد با استفاده از دستورات throw آن استثنا اعلام و بیرون کشیده می‌شود. با به کار بردن catch، استثنای رخ داده به دام افتاده و مورد پردازش قرار می‌گیرد.

هر دستوری که یک استثنا را به بیرون پرتاب می‌کند یا باید از داخل یک بلوک try به اجرا درآمده باشد یا توسط توابعی که از داخل یک بلوک try فراخوانی شده باشند. در غیراین صورت استثنای رخ داده باعث توقف روند اجرای برنامه می شود. هر استثنایی باید از طریق یک دستور catch به دام بیافتد. این دستور بلافاصله پس از دستور try آورده می‌شود که استثنا را به بیرون پرتاب کرده است.

Exception handling

شکل کلی دستورات try و catch به صورت زیر است:

```
try{
// try block
}
catch(type1 arg){
// catch block
}
catch(type2 arg){
// catch block
}
.
.
.
catch(typeN arg){
// catch block
}
```

Exception handling

بلوک try شامل آن بخشی از برنامه است که باید جهت کنترل احتمال بروز خطا هایی در آن بخش مورد مراقبت قرار گیرد. این بلوک می‌تواند مختصر باشد که تعدادی دستور در داخل یک تابع را شامل شود یا فراگیر باشد که تمامی کد main() را در بر گیرد.

زمانی که یک استثنا به بیرون پرتاب می‌شود توسط دستور catch متناظر با آن به دام خواهد افتاد و توسط آن استثنا پردازش خواهد شد. در ارتباط با یک بلوک try ممکن است بیش از یک دستور catch وجود داشته باشد. از بین چند catch، دستورات آن catch به اجرا در خواهد آمد که نوع داده catch با نوع داده آن استثنا مطابقت داشته باشد.

وقتی یک استثنا به دام بیافتد، کد اطلاعات خطا با پارامتر arg مقدار دهی خواهد شد. هر نوع داده ای ممکن است به دام بیفتد از جمله انواع کلاسی که خود شما می سازید.

Exception handling

شکل کلی دستور throw به صورت زیر است:

throw exception;

exception مقداری است که به بیرون پرت می شود تا یکی از بلوک های catch متناظر با آن بلوک try به اجرا در آید.

اگر استثنایی به بیرون پرت شود که برای آن دستور catch قابل اعمالی وجود نداشته باشد در آن صورت ممکن است برنامه به طور نابهنجاری خاتمه پیدا کند.

پرتاب کردن استثنایی که به آن رسیدگی نخواهد شد باعث می‌شود تابع terminate() احضار شود. به طور پیش‌فرض تابع terminate() برای آنکه برنامه را خاتمه دهد به نوبه خود تابع abort() را فراخوانی می کند. می توان به جای آن، اداره کننده ای جهت خاتمه برنامه مشخص ساخت.

Exception handling

مثال:

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
    cout << "start\n";
    try {
        cout << "Inside try block\n";
        throw 10;
        cout << "This will not execute";
    }
    catch (int i) {
        cout << "caught one! Number is:";
        cout << i << "\n";
    }
    cout << "end";
    system("pause");
    return 0;

}
```

output

start
Inside try block
caught one! Number is:10
endPress any key to continue . . .

Exception handling

بلوک try را می توان نسبت به یک تابع، محلی نمود. در این حالت هر بار که تابع اجرا می شود، مکانیسم رسیدگی به استثناها نسبت به آن تابع reset می شود. مثال:

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void Xhandler(int test)
{
    try {
        if (test) throw test;
    }
    catch (int i) {
        cout << "Caught one! Ex. #:" << i << "\n";
    }
};
int main()
{
    cout << "Start\n";
    Xhandler(1);
    Xhandler(2);
    Xhandler(0);
    Xhandler(3);
    cout << "end";
    system("pause");
    return 0;
}
```

output

Start
Caught one! Ex. #:1
Caught one! Ex. #:2
Caught one! Ex. #:3
endPress any key to continue . . .

Exception handling

❖ مراقبت کردن از همه عبارت ها

ممکن است نیاز باشد اداره کننده استثنا هر نوع استثنایی را به دام اندازد به جای آن که تنها مترصد یک نوع به خصوص باشد. در این حالت از شکل catch زیر استفاده می شود.

```
catch(…){
// process all exceptons
}
```

در اینجا (...) با هر نوع داده انطباق خواهد داشت. یک کاربرد بسیار خوب برای دستور catch(…) آن است که به عنوان آخرین دستور catch در دسته ای از دستورات catch استفاده شود. با به دام انداختن هر نوع استثنا مانع از آن خواهید شد که یک استثناء رسیدگی نشده باعث خاتمه یافتن برنامه به صورت نابهنجار شود.

Exception handling

❖ محدود کردن عبارت های throw

زمانی که از داخل یک بلوک try تابعی فراخوانی می شود، می توان نوع استثنایی که آن تابع می‌تواند بیرون اندازد را محدود کرد. حتی می‌توان مانع از آن شد که تابع استثنایی به بیرون پرتاب کند. برای اعمال چنین محدودیت هایی باید به تعریف آن تابع یک عبارت throw اضافه کرد. شکلی کلی در زیر نشان داده شده است:

```
ret-type func-name (arg-list) throw (type-list)
{
//…
}
```

در اینجا این تابع تنها انواع داده‌هایی را می‌تواند به بیرون پرتاب کند که در لیست type-list مشخص شده اند. پرتاب کردن هر نوع داده دیگری برای استثنا، باعث خاتمه یافتن نابهنجار برنامه خواهد شد.

Exception handling

❖ محدود کردن عبارت های throw

- ✓ اگر می خواهید یک تابع بخصوص نتواند هیچ نوع استثنایی پرتاب کند، در آن صورت باید type-list را خالی بگذارید.
- ✓ تلاش در بیرون انداختن استثنایی که از سوی هیچ تابعی پشتیبانی نمی شود، باعث خواهد شد تا تابع unexpected() فراخوانی گردد. این تابع به طور پیش فرض باعث فراخوانی تابع abort() می‌شود که باعث خاتمه یافتن ناهنجار برنامه می گردد.
- ✓ همانطور که قبلا گفته شد، می توان به جای abort() تابع دیگری جهت خاتمه بخشیدن به برنامه مشخص کرد.

Exception handling

❖ محدود کردن عبارت های throw

چگونه می توان نوع استثنایی که یک تابع می‌تواند به بیرون پرتاب کند را محدود نمود؟

```
void Xhandler(int test ) throw (int, char, double)
{
  if (test==0) throw test;
  if (test==1) throw 'a';
  if (test==2) throw 12.32;
}
```

یک تابع را فقط نسبت به نوع استثناهایی که به بلوک try فراخواننده خود برمی گرداند می توان محدود کرد. یعنی بلوک try که داخل آن تابع است هنوز هم می‌تواند یک نوع استثنایی را به بیرون پرتاب کند البته تا زمانی که به آن استثنا ها در داخل خود همان تابع رسیدگی شود. این محدودیت فقط در زمان پرتاب کردن یک استثنا به بیرون از آن تابع اعمال می گردد.

Exception handling

❖ راه اندازی مجدد یک استثنا

می توان یک استثنا را از داخل یک اداره کننده استثنا مجدداً به بیرون پرتاب کرد. با فراخوانی throw به تنهایی این کار را می توان انجام داد. این عمل باعث می شود تا استثناء فعلی به دستورات catch/try فوقانی ارسال شود. بدین ترتیب می‌توان به چند اداره کننده استثنا اجازه داد تا به آن استثنا دسترسی پیدا کنند.

ممکن است یک اداره کننده تنها به یکی از جنبه های آن استثنا بپردازد و اداره کننده بالاتر به جنبه دیگری از آن رسیدگی نماید. یک استثنا را فقط از داخل یک بلوک catch یا از داخل هر تابعی که از داخل آن بلوک فراخوانده شده باشد می‌توان راه اندازی مجدد کرد.

زمانی که یک استثنا راه اندازی مجدد می شود، مجدداً توسط همان دستور catch به دام نخواهد افتاد بلکه به سمت دستورات catch بعدی ارسال خواهد شد.

Exception handling

❖ نحوه راه اندازی مجدد یک استثنا

```
#include "stdafx.h"
#include <iostream>
using namespace std;
void Xhandler()
{
    try {
         throw "hello";
    }
    catch (char *) {
        cout << "Caught char * inside xhandler\n";
        throw;
    }
};
int main()
{
    cout << "Start\n";
    try {
    Xhandler();
    }
    catch (char *) {
        cout << "Caught char * inside main\n";
    }
    cout << "end";
    system("pause");
    return 0;
}
```

output

Caught char * inside xhandler
Caught char * inside main
endPress any key to continue . . .

assert()

assert() برای آزمایش برنامه از نظر منطقی استفاده می شوند. برای استفاده از این دستور باید فایل کتابخانه ای cassert یا assert.h به header fileها اضافه شوند. حالت کلی این دستور به صورت زیر است:

void assert(int expression);

اگر expression مقدار صفر (false) داشته باشد، expression، نام فایل Source code و شماره خط به خطای استاندارد ارسال می شود و سپس تابع abort ( ) صدا زده می شود و برنامه به طور غیرعادی خاتمه می یابد.

Assertion failed: <expression>, file <filename>, line <line number>

assert()

با assert()

```
#include <cassert>
using namespace std;
int Arr[10];
int GetArrayValue(int n)
{
    assert(n >= 0 && n <= 9);
    return Arr[n];
}
int main()
{
    GetArrayValue(10);  // error
    system("pause");
    return 0;
}
```

مثال:

output

```
Assertion failed: n >= 0 && n <= 9, file
c:\users\mdasus\documents\visual studio 2015\
projects\test.cpp, line 12
```

بدون assert()

```
int Arr[10];
int GetArrayValue(int n)
{
    if (n < 0 || n > 9)
        return -1;
    return Arr[n];
}
int main()
{
    GetArrayValue(10);
    system("pause");
    return 0;
}
```

assert()

در production code نباید با دستور assert مواجه شد زیرا تمام آزمایش ها باید تا آن زمان انجام شده باشد. زمانی که آماده بودید که برنامه را به سفارش دهنده تحویل بدهید، استفاده از ماکروی زیر می تواند موجب شود کامپایلر در زمان کامپایل تمام دستورات assert را نادیده بگیرد. این ماکرو باید قبل از هدر فایل assert.h یا cassert تعریف شده باشد. مثال:

```
#include <cassert>
using namespace std;

int main()
{
    int x = 7;
    x = 9;
    assert(x == 7);
    system("pause");
    return 0;
}
```

⇨

Assertion failed: x == 7, file c:\users\mdasus\documents\visual studio 2015\projects\test.cpp, line 14

```
# define NDEBUG
#include <cassert>
using namespace std;
int main()
{
    int x = 7;
    assert(x == 5);
    system("pause");
    return 0;
}
```

⇨

برنامه به درستی کامپایل می شود.

assert()

از assert() در زمان اجرا، استفاده می شود. static_assert() برای اجرا در زمان کامپایل طراحی شده است. اگر شرط ورودی false باشد، پیغام مورد نظر نمایش داده می شود. حالت کلی این دستور به صورت زیر است:

static_assert(constant_expression, string_literal);

مثال زیر از static_assert() برای اطمینان از اندازه نوع داده ها استفاده کرده است.

```
int main()
{
  static_assert(sizeof(long) == 8, "long must be 8 bytes");
  static_assert(sizeof(int) == 4, "int must be 4 bytes");
  system("pause");
  return 0;
}
```

output

static assertion failed with "long must be 8 bytes" Projects c:\Users\MDAsus\Documents\Visual Studio 2015\Projects \test.cpp 14

Standard Template Library (STL)

STL مجموعه ای از کلاس های قالب C++ است تا ساختمان داده و توابع معمول برنامه نویسی مانند لیست ها، پشته ها، آرایه ها و غیره را ارائه دهد. این یک کتابخانه از کلاس های ظروف (container)، الگوریتم ها (algorithms)، Functions و تکرارها (iterators) است.

نکته: در تمام ساختمان داده‌های STL که تخصیص حافظه به صورت پویا وجود دارد، عملیات مدیریت حافظه در پس‌زمینه صورت گرفته و برنامه‌نویس درگیر این موضوع نمی‌شود.

این یک کتابخانه عمومی است. بنابراین، اجزای آن پارامتر می شوند. دانش کار در کلاس های template پیش شرط کار با STL است. STL شامل چهار جزء است:

❖ Algorithms

Algorithms مجموعه ای از توابع را تعریف می کند که بخصوص در طیف وسیعی از عناصر استفاده می شوند. آنها روی containers عمل می کنند و ابزاری برای عملیات مختلف برای محتویات containers فراهم می کنند.

Standard Template Library (STL)

❖ Containers

کلاس هایی هستند که داده و اشیاء را ذخیره می کنند. در کل هفت کلاس استاندارد "first-class" container و سه کلاس container adaptor و تنها هفت فایل سرآمد (header files) وجود دارد که دسترسی به این containerها یا container adaptors را فراهم می کند. آرایه ساده‌ترین container است که در عموم زبان‌های برنامه‌نویسی وجود دارد.

❖ Functions

STL شامل کلاسهایی است که عملیات فراخوانی توابع را سربارگزاری می‌کند. نمونه‌هایی از این کلاس‌ها اشیا تابع یا functors نامیده می‌شوند. functors اجازه می‌دهند تا توابع مرتبطی که با کمک ارسال پارامتر به آنها سفارشی شده اند کار کنند.

❖ Iterators

از Iterators برای کار بر روی دنباله ای از مقادیر استفاده می شود. با دسترسی مستقیم به حافظه، پیچیدگی و زمان اجرا برنامه را کاهش می دهند.

Standard Template Library (STL)

```
#include <stack>
using namespace std;
int main()
{
    // stack<type> name;
    stack<int> myStack;
    myStack.push(1);
    myStack.push(3);
    myStack.push(4);
    myStack.push(6);
    myStack.push(5);
    while (!myStack.empty()) {
        cout << myStack.top() << "\t";
        myStack.pop();
    }
    // output:  5  6  4  3  1
    system("pause");
    return 0;
}
```

Standard Template Library (STL)

```
#include <queue>
using namespace std;
int main()
{
    // queue<type> name;
    queue<int> myQueue;
    myQueue.push(1);
    myQueue.push(3);
    myQueue.push(4);
    myQueue.push(6);
    myQueue.push(5);
    while (!myQueue.empty()) {
        cout << myQueue.front() << "\t";
        myQueue.pop();
    }
    // output:  1  3  4  6  5
    system("pause");
    return 0;
}
```

Standard Template Library (STL)

✓ کلاس vector از Container

در مثال زیر vector به اسم test شامل ۱۰ خانه از نوع int که مقدار اولیه همه خانه ها با صفر مقداردهی شده است:

vector <int> test(10,0);

❖ دسترسی به عناصر یک vector

۱-روش عادی مانند آرایه: test[5];

۲-با استفاده از تابع عضو at: test.at(5);

استفاده از at بهتر است چون اگر اندیس خارج از سایز را وارد کنیم، برنامه به طور هوشمند خطا می دهد و مشکلی برای داده های دیگر حافظه رخ نمی دهد. ولی در حالت یک چنین اتفاقی نمی افتد.

Standard Template Library (STL)

✓ کلاس vector از Container

❖ اندازه یک شئ vector

اندازه (size): اندازه همان تعداد خانه هایی است که در آن ها مقدار داریم. در وکتوری که ابتدا ایجاد کردیم size ۱۰ است (که در ابتدای تعریف همه با صفر پر شده اند). برای دریافت مقدار size تابع عضو (متد) size را داریم که استفاده از آن به این شکل است: test.size();

❖ ظرفیت یک شئ vector

ظرفیت (capacity): اگر بخواهیم یک مقدار جدید اضافه کنیم و این کار را چند بار انجام دهیم. کمی وقت می گیرد تا از حافظه بخشی را برای آن جدا کند به همین خاطر در vector تعدادی خانه را آماده نگه می دارد برای وقتی که خواستیم به طول آن اضافه کنیم برای پی بردن به ظرفیت از تابع عضو capacity() استفاده می کنیم: test.capacity();

اگر عنصری را به آرایه اضافه کنیم در صورتی که آرایه پر شده باشد یکی به اندازه و یکی بیشتر از دفعه قبل به ظرفیت اضافه می شود.

Standard Template Library (STL)

✓ کلاس vector از Container

❖ بیشترین طول ممکن برای یک شئ vector

max_size () تابع عضو بیشترین طول ممکن برای ایجاد یک vector را نشان می دهدکه اندازه آن به مشخصات سیستم مربوط می شود:

test.max_size();

❖ تغییر اندازه یک vector

تابع عضو resize () یکی از توابع بسیار کاربردی است. فرض کنید در vector، test با اندازه ۱۰ ما ۴ خانه آخر را نخواهیم یا ۴ خانه کم داشته باشیم و بخواهیم اضافه کنیم :

test.resize(6);
test.resize(14);

در دستور اول size را به ۶ کاهش دادیم و در دستور بعدی size به ۱۴ تا افزایش داده شده است.

Standard Template Library (STL)

✓ کلاس vector از Container

❖ اضافه کردن به انتهای vector

تابع عضو push_back: test.push_back(8);

یک خانه به آخر vector با مقدار ۸ اضافه می‌شود و به اندازه هم یکی اضافه می‌شود.

❖ برداشتن آخرین عنصر داخل vector

تابع عضو pop_back: test.pop_back();

❖ دیدن مقدار اولین و آخرین عنصر داخل vector

تابع های عضو back و front هم خانه های آخر و اول vector را تعیین می کنند:

test.front(); test.at(0);

دو دستور بالا با هم برابرند و مقدار عنصر ابتدا vector را نشان می دهد.

❖ پاک کردن تمام مقادیر داخل vector

برای خالی کردن یک vector از تابع عضو clear استفاده می کنیم: test.clear();

Standard Template Library (STL)

✓ کلاس vector از Container

یک vector را می‌توان آرایه‌ای در نظر گرفت که امکان تغییر اندازه‌ی آن به صورت پویا و حین اجرای برنامه وجود دارد. به این ترتیب می‌توان هر تعداد عنصر دلخواه (تا جایی که حافظه یاری کند) به آن push_back نمود و به اندیس دلخواه با کارایی مناسب دسترسی داشت.

```
#include <vector>
using namespace std;
int main()
{
    // vector<type> name(size = 0);
    vector<int> myVector1 = { 1, 2, 3, 4 }, myVector2(5);
    myVector2[2] = 3;
    cout << myVector1.size() << "-" << myVector1.capacity() << endl;
    // output:  4 - 4
    myVector1.push_back(5);
    cout << myVector1.size() << "-" << myVector1.capacity() << endl;
    // output:  5 - 6
    system("pause");
    return 0;
}
```

Standard Template Library (STL)

✓ کلاس vector از Container

با تعریف myVector1 در قطعه کد فوق، آرایه‌ای به طول ۴ برای آن در نظر گرفته می‌شود. زمانی که عدد ۵ با استفاده از متد push_back به انتهای آن درج می‌شود، تعداد عناصر بیش از ظرفیت (capacity) آرایه شده و امکان درج وجود ندارد. به همین دلیل پس از اجرای عمل درج، ظرفیت vector به ۶ افزایش یافته و اندازه نیز به ۵ تغییر می‌کند.

با استفاده از متد resize امکان تغییر اندازه vector وجود دارد. اگر این تغییر اندازه باعث افزایش طول باشد، عناصر جدید با مقدار پیش‌فرض نوع داده (مقدار صفر در مثال بالا) پر می‌شود و اگر اندازه کوچکتر شود، برخی از عناصر از انتهای آرایه حذف می‌شوند. این تغییرات تاثیری در ظرفیت آرایه نمی‌دهند؛ مگر آنکه افزایش طول بیش از ظرفیت vector باشد.

تست و اشکال زدایی برنامه

خطاهای برنامه شامل خطاهای دستور زبان (syntax errors) و خطاهای معنایی (semantic errors) که به آنها خطاهای منطقی (logic errors) نیز گفته می شود، می باشند.

syntax error زمانی اتفاق می افتد که یک statement از نظر قوانین نگارشی زبان C++ صحیح نباشد. خطاهای syntax errors می تواند شامل مواردی مانند فراموش کردن سمی کولون، متغیرهای اعلام نشده، پرانتزها و آکولادهایی که بسته نشده اند، رشته هایی که به درستی پایان نیافته اند و ... باشد. برای مثال برنامه زیر دارای چندین خطای syntax error می باشد:

```
int main()
{
  cout < "Hi there"; << x; // invalid operator (<), extraneous semicolon, undeclared variable (x)
  return 0 // missing semicolon at end of statement
}
```

تست و اشکال زدایی برنامه

کامپایلر معمولاً syntax errors را کشف می کند و خطایی تولید می کندکه به سادگی می توانید این خطاها را پیدا کرده و برطرف سازید.

semantic error هنگامی اتفاق می افتد که برنامه از نظر syntax مشکلی ندارد و به درستی کامپایل و اجرا می شود، اما خروجی که مد نظر برنامه نویس می باشد، نشان نمی دهد.

گاهی semantic errors باعث می شود تا در هنگام اجرا، برنامه crash کند، مانند خطای تقسیم بر صفر (divide by zero).

```
int main()
{
    int a = 10;
    int b = 0;
    cout << a << " / " << b << " = " << a / b; // division by 0 is undefined
    system("pause");
    return 0;
}
```

تست و اشکال زدایی برنامه

گاهی semantic errors فقط منجر به تولید نتایج اشتباه می شوند.

```
int main()
{
    int x;
    cout << x; // Use of uninitialized variable leads to undefined result
    system("pause");
    return 0;
}

int add(int x, int y) {
    return x - y; // function is supposed to add, but it doesn't
}
int main() {
    cout << add(5, 3); // should produce 8, but produces 2
    system("pause");
    return 0;
}
int main()
{
    return 0; // function returns here
    cout << "Hello, world!"; // so this never executes
    system("pause");
}
```

تست و اشکال زدایی برنامه

✓ Debugger

debugger، برنامه کامپیوتری است که به برنامه نویس اجازه می دهد تا چگونگی اجرای یک برنامه را کنترل و آن چیزی را که در طول اجرای برنامه اتفاق می افتد مشاهده کند. برنامه نویس می تواند با استفاده از ابزار debugger یک برنامه را خط به خط اجرا کند و مقادیر متغیرها را در هر خط مشاهده و بررسی کند. سپس مقادیر ایجاد شده در طول برنامه را با مقادیر واقعی که مد نظر است مقایسه کند. debugger ابزاری فوق العاده برای ردیابی semantic errors می باشد.

✓ Stepping

Stepping یکی از ویژگیهای ابزار debugger می باشد که امکان می دهد برنامه را خط به خط اجرا کنید. سه نوع دستور Stepping وجود دارد:

❖ step into

دستور step into خط بعدی از کد را اجرا می کند. اگر این خط کد مربوط به فراخوانی یک تابع باشد، وارد تابع می شود و کنترل اجرای برنامه را به ابتدای تابع می برد. (F11)

تست و اشکال زدایی برنامه

❖ step over

دستور step over همانند دستور Step into یک خط از کد را اجرا می کند. تفاوت این دو دستور در این است که در دستور Step over هنگامی که اجرای کد به فراخوانی تابع برسد، وارد آن تابع نخواهد شد و تابع کلاً اجرا می گردد و کنترل اجرای برنامه به خط بعدی تابع می رسد.(F10)

❖ step out

دستور step out تنها خط بعدی را اجرا نمی کند. این دستور کلیه کدهای موجود در تابع جاری را اجرا و کنترل اجرای برنامه را به بعد از خاتمه تابع جاری می‌برد.

✓ Breakpoints

Breakpoint یک marker خاص می باشد که به debugger می گوید اجرای برنامه را در خط خاصی که این مارکر را دارد، متوقف کند و وارد حالت دیباگ شود. (F9)

✓ پنجره Watch

هنگام debug کردن برنامه این امکان وجود دارد تا مقادیر متغیرها را مشاهده کرد.

تست و اشکال زدایی برنامه

✓ آزمایش توابع

یکی از روش های متداول برای کشف مشکلات برنامه نوشتن توابع تست برای آزمایش کدی است که نوشته شده است

```
int add(int x, int y)
{
    return x + y;
}

void testadd()
{
    cout << "This function should print: 2 0 0 -2\n";
    cout << add(1, 1) << " ";
    cout << add(-1, 1) << " ";
    cout << add(1, -1) << " ";
    cout << add(-1, -1) << " ";
}

int main()
{
    testadd();
    system("pause");
    return 0;
}
```

تابع testadd () با فراخوانی تابع add () با مقادیر مختلف، عملکرد آن را آزمایش می کند. اگر همه مقادیر با انتظارات مطابقت داشته باشند، می توان اطمینان حاصل کرد که تابع عملکرد درستی دارد. حتی می توان این تابع را حفظ کرد و هر زمان که تابع add () تغییر داده شد آن را اجرا کرده تا اطمینان حاصل شود که به طور تصادفی خراب نشده است.

این یک شکل اولیه از تست واحد (unit test) است، که یک روش تست نرم افزاری است و توسط آن واحد های کوچک کد منبع تست می شوند تا صحت آن تعیین شوند.

مقدمات و معرفی Qt

- امروزه توسعه نرم افزار و به روز رسانی های آن در انواع platform ها از قبیل Linux، Windows، Mac OS X و همچنین platform های موبایل و تبلت از قبیل Andoird، IOS، Blackberry و ... با سرعت بسیار زیادی دنبال می شود.
- اکثر برنامه نویسان تمایل دارند که یک زبان ویژه با تمامی قابلیت ها و مهمتر از همه پشتیبانی از Object Oriented و Performance بالا را همراه با یک IDE همه کاره و جذاب در اختیار داشته باشند. با استفاده از ابزار Qt به همراه C++ می توان همه موارد فوق را همزمان داشت.
- این ابزار بر خلاف محیط های VS سیاست انحصاری بودن را ندارد. به این معنا که فقط ویندوز نیست که از این ابزار پشتیبانی میکند، بلکه سیستم عامل های قدرتمند یونیکسی مانند Linux و Mac OS X از این ابزار پشتیبانی می کنند.

مقدمات و معرفی Qt

- Qt مجموعه‌ای ازکتابخانه‌ها و header های نوشته شده به زبان C++ است که به برنامه‌نویسان امکان توسعه آسان نرم‌افزارهای کاربردی را می‌دهد.
- کتابخانه‌های این framework که صدها کتابخانه کامل به زبان C++ می باشد پایه framework را تشکیل می دهد. این کتابخانه ها شامل مواردی چون کتابخانه های GUI، پایگاه داده، شبکه، OpenGL، XML درونی‌سازی شده، بین‌المللی‌کردن (internationalization) و ... می باشد.
- یک framework برنامه‌نویسی برای ایجاد نرم‌افزارهای گرافیکی (GUI) و خط فرمان (Console) چندسیستم‌عاملی است. Qt در دو نگارش رایگان و تجاری ارائه می‌شود.
- زبان برنامه نویسی در Qt بصورت پیش فرض C++ می باشد. امکان برنامه نویسی با زبانهای دیگر چون Python، Rubby، PHP، PERL، Pascal و حتی #C و JAVA نیز در Qt فراهم می باشد.

مقدمات و معرفی Qt

- widget های رابط کاربری گرافیکی می توانند سیگنال هایی (signals) را ارسال کنند که حاوی اطلاعات مربوط به رویداد هاست که این اطلاعات توسط کنترل های دیگر و به کمک توابعی ویژه که به آن ها شکاف ها (slot) گفته می شود دریافت می شوند.
- از قابلیت های Qt می توان به قابلیت Cross platform بودن برنامه های تولید شده توسط C++/Qt اشاره کرد به راحتی میتوانید خروجی را در سیستم عامل مورد نظر دریافت و کامپایل کنید. با کمک QT می‌توان نرم‌افزارهایی با تغییرات کم یا بدون تغییرات، قابل اجرا روی سیستم‌عامل‌های مختلف را طراحی و برنامه‌نویسی نمود.
- قابلیت طراحی با QML و همچنین پشتیبانی از CSS و HTML یکی دیگر از مزایای Qt می باشد. مثلا فرض کنید یک فرم طراحی می کنید که در حالت عادی خالی از style و افکت های ویژه است برای این کار میتوانید با استفاده از HTML و CSS برنامه را بهینه کنید.

مقدمات و معرفی Qt

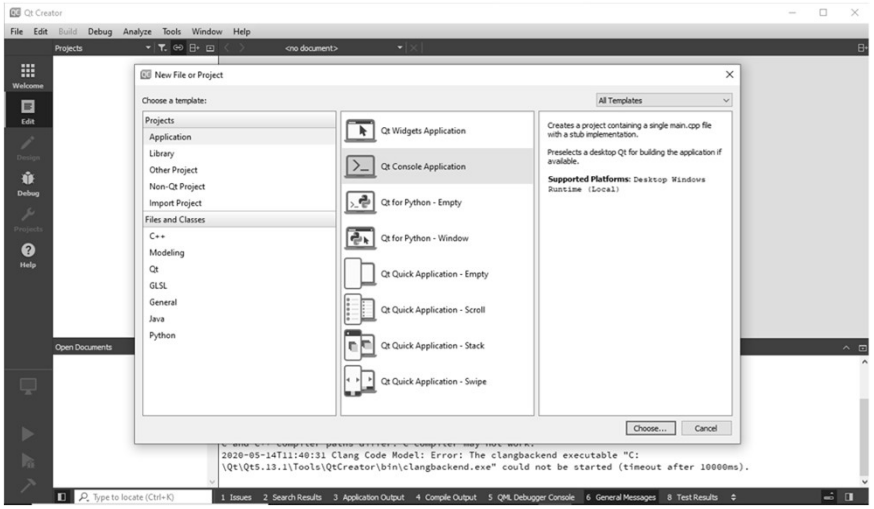
- Qt به خوبی از پردازش‌های موازی پشتیبانی می‌کند و در نتیجه سرعت پاسخگویی به کاربر در سیستم‌هایی که دارای چندین هسته‌ی پردازنده هستند، قابل قبول خواهد بود.
- نرم افزار های بسیاری چون Opera, Google Earth, Skype, Qtopia و ... نیز توسط این ابزار ایجاد گردیده اند. و همچنین شرکت های DreamWorks، Lucasfilm، Panasonic، Philips، Samsung، Siemens، Volvo، Walt Disney، Blizzard Entertainment، Electronic Arts و AMD از این framework استفاده می کنند.

نصب و راه اندازی محیط Qt

- دانلود و نصب محیط از https://www.qt.io/offline-installers
- انتخاب نسخه مورد نظر متناسب با سیستم عامل
- در این مثال ها نسخه Qt 5.14.2 for Windows استفاده شده است.
- پس از نصب و اجرای محیط، New project و Qt console application را انتخاب کنید. (شکل اسلاید بعد)
- نام و آدرس برنامه ای که می نویسید را وارد کنید.
- دو فایل در برنامه وجود دارد:
  - Test1.pro

    کتابخانه های مورد نظر را فرا خوانی می کند. (Test1 اسم برنامه است.)
  - Main.cpp

نصب و راه اندازی محیط Qt



برنامه ساده با Qt

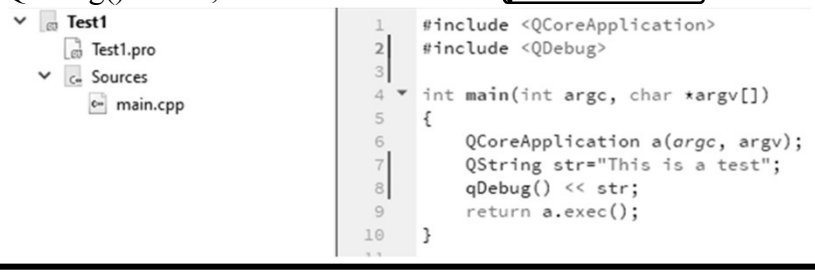
- Qt علاوه بر کتابخانه های C++ کتابخانه های دیگری هم دارد.
- اضافه کردن کتابخانه زیر

```
#include<QDebug>
```

- اضافه کردن دستورات زیر در main برنامه

```
QString str="This is a test";
Qdebug() << str;
```

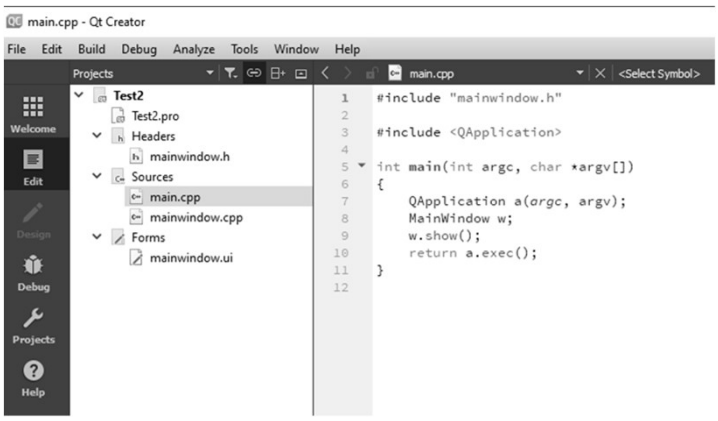
output
This is a test



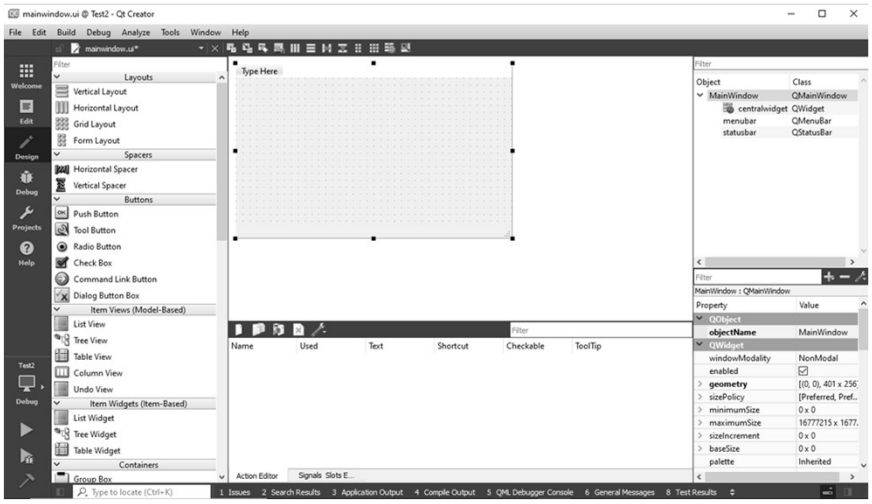
برنامه ساده Qt همراه با فرم

- اجرای محیط، New project و Qt Widgets Application را انتخاب کنید. (شکل اسلاید بعد)
- نام و آدرس برنامه ای که می نویسید را وارد کنید.
- چهار فایل در برنامه وجود دارد:
  - Sources: فایل های با پسوند .cpp
  - Headers: فایل های کتابخانه ای
  - Forms: فایل های با پسوند ui برای محیط گرافیکی
  - فایل Test2.pro نام کلاس

برنامه ساده Qt همراه با فرم



برنامه ساده Qt همراه با فرم



برنامه ساده Qt همراه با فرم

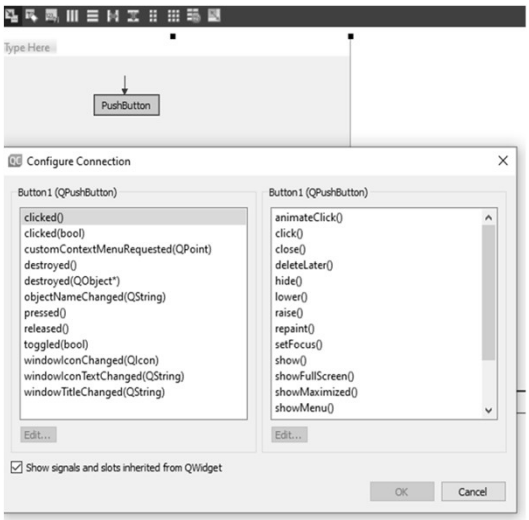
- یک Button به برنامه اضافه کنید.
- اشیا مختلفی مانند Buttonsها و ... را می توانید در بخش گرافیکی اضافه کنید. هر یک از این اشیا خواصی دارند که می توانید تنظیم کنید. به عنوان مثال Button شامل دو نام یک نام در برنامه (objectName) و یک نام که روی آن نمایش داده می شود (text) است.
- برای دسترسی به هرشئ، شئ کلی از پیش تعریف شده ui مورد استفاده قرار می گیرد.

برنامه ساده Qt همراه با فرم

❖ نام شئ را به Button1 و نام نمایش داده شده روی آن را باروش زیر در برنامه بنویسید.

ui->Button1->setText("Exit");

برنامه ساده Qt همراه با فرم



❖ برای اینکه Button عملی را انجام دهد عملیات زیر را انجام دهید:

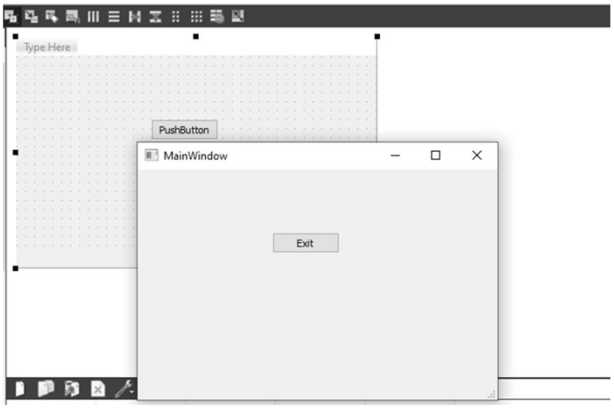
❖ انتخاب گزینه Edit Signals / Slots

❖ بعد انتخاب شئ مورد نظر که می خواهیم عملیات روی آن انجام شود.

❖ از بین رویداد ها clicked() و close() را انتخاب نمایید.

برنامه ساده Qt همراه با فرم

❖ بعد از اجرای برنامه با زدن این دکمه برنامه بسته خواهد شد.



گرافیک

OpenGL یا DirectX؟

✍ OpenGL تقریبا روی تمام سیستم‌عامل‌ها قابل اجرا است و نیازی به تغییر Source برنامه نیست اما به عنوان مثال DirectX روی Linux قابل اجرا نیست.

✍ DirectX نسبت به OpenGL ساده‌تر است اما OpenGL ویژگیها و توانایی‌های بیشتری دارد.

✍ توسعه دهندگان OpenGL برای کار با آن یک کتابخانه به نام OpenGL Utility را گسترش دادند تا به کمک آن می‌توان به راحتی با پنجره‌ها، کی‌بورد، ماوس و موارد دیگر کار کرد. همچنین ویژگیهای مدلسازی بسیاری مانند سطوح مربعی و منحنی را ممکن می‌سازد.

✍ البته GLUT نیز همانند OpenGL مستقل از platform است و در نتیجه امکانات و جزئیات کمتری را می تواند ارائه کند.

گرافیک

✍ OpenGL به شیوه‌ای طراحی شده که دستورات در یک محل صادر و در محل دیگر اجرا می‌شود. به این معماری ساختار مشتری/سرویس‌دهنده (client/server) گویند.

✍ دستورات ترسیمی OpenGL، client نامیده می‌شود و کامپیوتری که این دستورات را می‌گیرد و آن‌ها را اجرا می‌کند server نامیده می‌شود.

✍برای اجرای دستورات OpenGL در ویندوز فایل های کتابخانه ای gl.h و glut.h را در پوشه include و پوشه GL، فایل glut.lib را در پوشه lib و فایل glut.dll را در پوشه bin محیط برنامه نویسی خود قرار دهید.

✍برای مطالعه بیشتر در مواردی که ذکر می شود از مراجع مختلف از جمله کتاب Computer Graphics Programming in OpenGL in C++ نوشته GORDON و CLEVENGER می توانید استفاده کنید.

انواع داده‌ها

Suffix	Data Type	Typical Corresponding	OpenGL Type Definition
b	8bit integer	signed char	GLbyte
s	16bit integer	short	GLshort
i	32bit integer	int or long	GLint, GLsizei
f	32bit floating-point	float	GLfloat, GLclampf
d	64bit floating-point	double	GLdouble, GLclampd
ub	8bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16bit unsigned integer	unsigned short	GLushort
ui	32bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

انواع داده‌ها

✍ به‌عنوان مثال دو دستور یکسان زیر بر روی داده‌های متفاوتی انجام می‌گیرند:

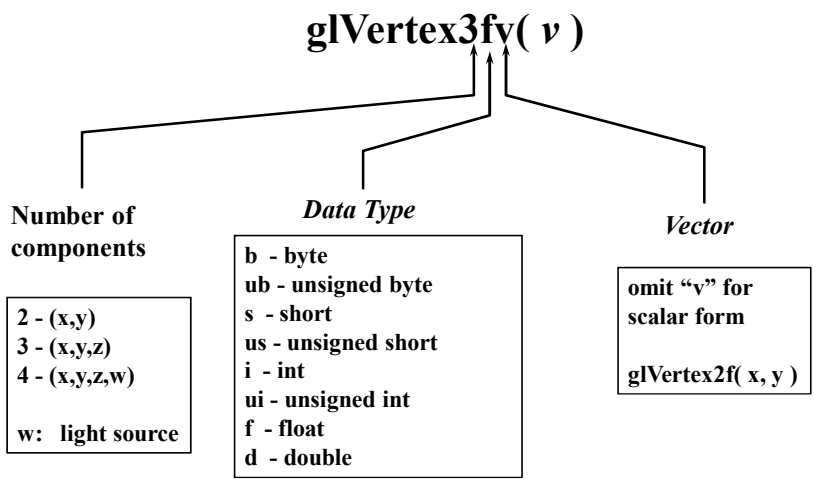
```
glVertex2i(1, 3);
glVertex2f(1.0, 3.0);
```

✍ مثال زیر نشان می‌دهد که چگونه باید فرمت را مشخص کرد:

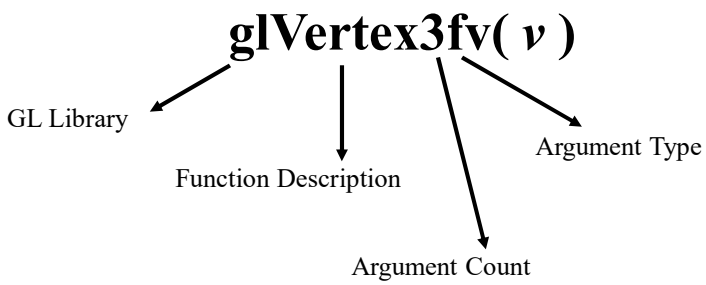
```
glColor3f(1.0, 0.0, 0.0);
GLfloat color_array[] = {1.0, 0.0, 0.0};
glColor3fv(color_array);
```

✍ v در فرمت داده‌ها نشان‌دهنده برداری (vector) و نبود آن نشان‌دهنده اسکالر است.

ساختار توابع در OpenGL



ساختار توابع در OpenGL



تعاریف

Render

پردازشی که توسط کامپیوتر برای ایجاد تصویر صورت می‌گیرد Render نامیده می‌شود.

Model

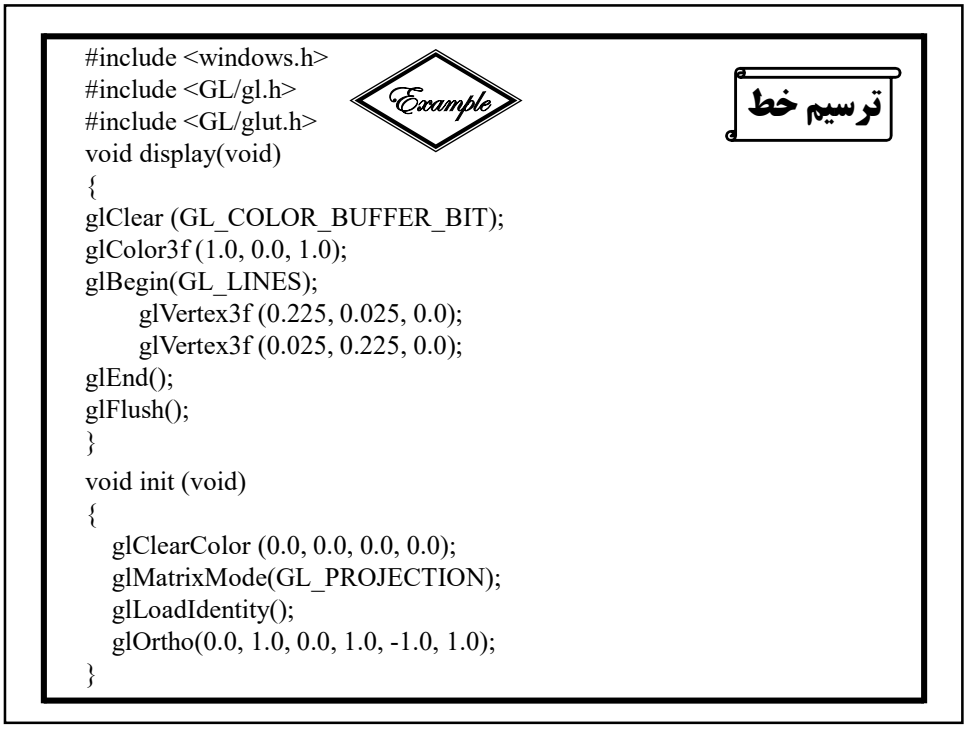
شیء مورد نظر که از نقطه، خط، چندضلعی و اصول هندسی ایجاد می‌شود Model نامیده می‌شود.

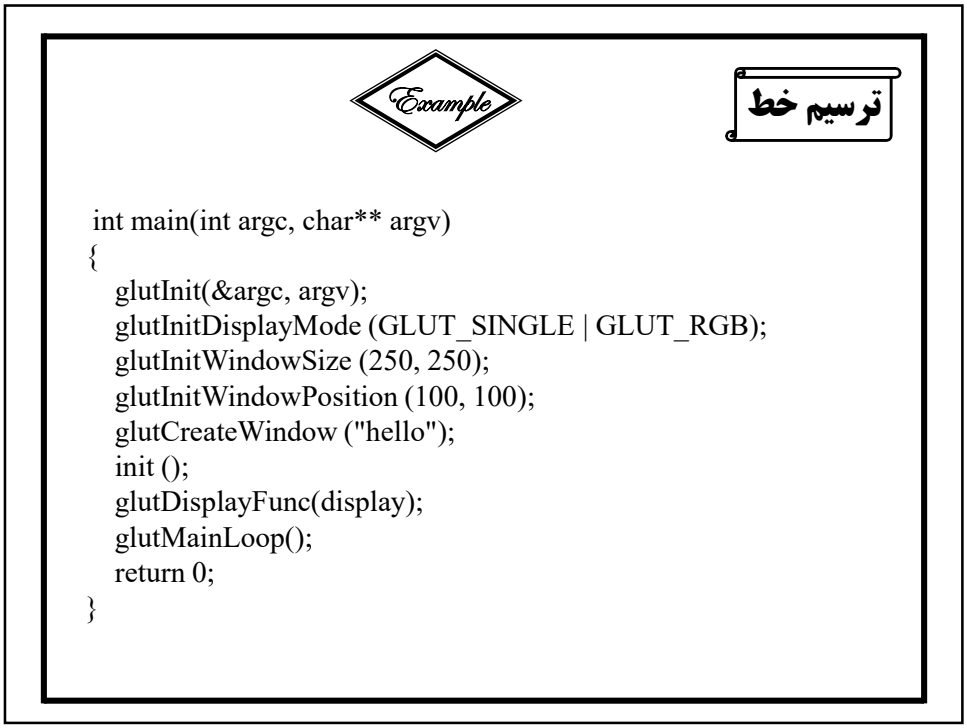
bitplane

قسمتی از حافظه که اطلاعات یک پیکسل را نگه‌داری می‌کند.

framebuffer

حافظه ای که اطلاعات صفحه گرافیکی نمایش داده شده را نگه‌داری می‌کند که از bitplane تمام پیکسل‌ها تشکیل شده است.


Example
ترسیم خط
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>
void display(void)
{
glClear (GL_COLOR_BUFFER_BIT);
glColor3f (1.0, 0.0, 1.0);
glBegin(GL_LINES);
glVertex3f (0.225, 0.025, 0.0);
glVertex3f (0.025, 0.225, 0.0);
glEnd();
glFlush();
}
void init (void)
{
glClearColor (0.0, 0.0, 0.0, 0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}


Example
ترسیم خط
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (250, 250);
glutInitWindowPosition (100, 100);
glutCreateWindow ("hello");
init ();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

شرح برنامه Headers

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>
```

- برای استفاده از OpenGL نیاز به header file ها داریم.
- برای استفاده از دو کتابخانه gl و glut باید windows.h را به کار ببریم، در برنامه‌هایی که بر روی windows اجرا می‌شوند، برای مدیریت پنجره‌ها به آن نیاز داریم. این فایل حتما باید در ابتدا قرار گیرد.
- برای ترسیم اشکال از gl و برای به‌کارگیری پنجره‌ها از glut استفاده می‌شود.

شرح برنامه Display

```
void display(void) {}
```

- تابع display تابعی است که تمام عملیات ترسیم شکل و render در آن قرار می‌گیرد.

```
glClear (GL_COLOR_BUFFER_BIT);
```

- برای پاک کردن صحنه که در بافر است از glClear استفاده می‌شود. در این برنامه بافر رنگ پاک شده است. به این معنی که اطلاعاتی از رنگ صحنه قبلی باقی نماند.
- بافرهایی که وجود دارند، عبارتند از:

Name	Buffer
GL_DEPTH_BUFFER_BIT	Depth buffer
GL_COLOR_BUFFER_BIT	Color buffer
GL_STENCIL_BUFFER_BIT	Stencil buffer
GL_ACCUM_BUFFER_BIT	Accumulation buffer

شرح برنامه Colors

```
glColor3f (1.0, 0.0, 1.0);
```

- برای تعیین رنگ از این تابع استفاده می‌شود که از سه رنگ قرمز، سبزو آبی (RGB) و ترکیب این سه رنگ با نسبت‌های مختلف می‌توان رنگ‌های متعددی را بدست آورد.
- تمام اشیاء از رنگ تعیین شده استفاده می‌کنند تا زمانیکه رنگ دیگری تعیین گردد.

شرح برنامه Objects

```
glBegin(GL_LINES);
    glVertex3f (0.225, 0.025, 0.0);
    glVertex3f (0.025, 0.225, 0.0);
glEnd();
```

- برای ترسیم شکل عملیات با glBegin شروع و با glEnd پایان می‌یابد.
- نوع تصویری که قرار است ترسیم شود در داخل پرانتز glBegin قرار می‌گیرد.
- بر اساس نوع شکل و ویژگیهای آن راسهایی را اضافه می کنیم.
- برای ترسیم خط به دو راس نیاز داریم.

```
glFlush();
```

- این دستور باعث می شود که از شروع و اجرای دستوراتی که در قبل آمده است در زمان محدودی اطمینان حاصل شود.
- در پایان دستورات ترسیمی از این تابع استفاده می شود.

شرح برنامه Objects

❖ انواع شکل‌ها عبارتند از:

Value	مفهوم
GL_POINTS	نقطه
GL_LINES	خط
GL_LINE_STRIP	مجموعه‌ای از خطوط به هم پیوسته
GL_LINE_LOOP	مانند قبلی که ابتدا و انتهای خطوط به هم پیوسته هستند.
GL_TRIANGLES	مثلث
GL_TRIANGLE_STRIP	مثلث‌های به هم پیوسته از قاعده
GL_TRIANGLE_FAN	مثلث‌های به هم پیوسته از راس
GL_QUADS	چهارضلعی
GL_QUAD_STRIP	چهارضلعی‌های به هم پیوسته از قاعده
GL_POLYGON	چندضلعی

شرح برنامه init

❖ glClearColor پیش زمینه را پاک می کند. دارای مقادیر RGBA می باشد که عبارتند از: قرمز، سبز، آبی و شفافیت. مقادیر اعشاری از صفر تا یک را می گیرد. صفر تاریکترین و یک روشن ترین مقدار است.

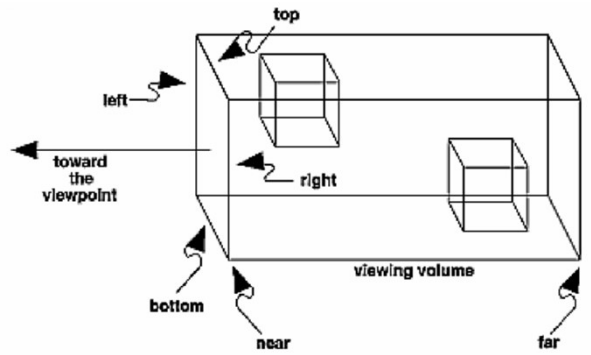
❖ رنگ پیش زمینه در این جا سیاه قرار داده شده است.

❖ برای تغییر مکان شیء و مکان دوربین از ماتریس ها استفاده می شود. glMatrixMode با آرگومان GL_PROJECTION نشان می دهد که ماتریس کنونی تغییرات تصویر را مشخص می کند.

❖ glLoadIdentity(); ماتریس تصویر جاری را به ماتریس واحد تبدیل می کند. (Reset)

❖ دستور glOrtho() دید ارتوگرافیک حجم را بوجود می آورد. در این حالت حجم در حال مشاهده به شکل یک جعبه دیده می شود. در این دید هدف آن است که اندازه ها و زوایای دقیقی از اشیاء را ارائه دهد.

شرح برنامه init



❖ **glOrtho**(GLdouble left, GLdouble right,GLdouble bottom, GLdouble top,GLdouble Near, GLdouble Far)

شرح برنامه main

❖ argc: در زمان اجرای برنامه از خط دستور تعداد پارامترهای خط فرمان که شامل حداقل نام خود برنامه است را تعیین می‌کند. (تعداد آرگومان های ارسالی)

❖ argv: پارامترها در argv قرار دارند. argv به صورت آرایه ای از اشاره گرها به رشته ها می باشد. (مقدار آرگومان های ارسالی)

```
glutInit(&argc, argv);
```

❖ برای آماده سازی محیط جهت استفاده از توابع GLUT استفاده می شود.

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
```

❖ برای موارد زیر به گار می رود:

- تعیین نوع بافر که DOUBLE یا SINGLE است.
- تعیین اینکه کدام یک از بافرهای DEPTH، STENCIL، ACCUMULATION را داشته باشد.
- تعیین محدوده رنگ مورد استفاده که می تواند RGB یا Color-index باشد.

```
glutInitWindowSize (250, 250) ;
```

❖ اندازه پنجره را بر اساس پیکسل مشخص می کند.

شرح برنامه main

glutInitWindowPosition (100, 100)

❖ مختصات x و y سمت چپ گوشه بالا را مشخص می کند.

glutCreateWindow ("hello");

❖ یک پنجره با محتوای OpenGL و با نام داده شده ساخته شده است و شناسه پنجره را مشخص می کند.

init ()

❖ فراخوانی تابع init

glutDisplayFunc(display);

❖ تعیین تابعی (display) که عملیات کشیدن را انجام خواهد داد.

glutMainLoop();

❖ تمام پنجره هایی که ساخته شده اند اکنون نمایش داده می شوند.

شرح برنامه

❖ بسیاری از حالات وجود دارند که به طور پیش فرض فعال نیستند. برای فعال کردن آن ها باید از ()glEnable و برای غیرفعال نمودن از ()glDisable استفاده می شود.

❖ برای فعال کردن الگوی خط باید GL_LINE_STIPPLE را فعال نمود.

glEnable(GL_LINE_STIPPLE);

❖ در انتها برای غیرفعال نمودن:

glDisable(GL_LINE_STIPPLE);

❖ عرض خطوط ترسیمی با دستور زیر مشخص می گردد که باید از یک بزرگتر باشد.

glLineWidth(3.0);

❖ GL_LINE_STRIP مجموعه ای از خطوط به هم پیوسته می باشد.

❖ برای ترسیم خط دو راس کافی است.

شرح برنامه

❖ انواع حالات در ترسیم خط عبارتند از:

```
glLineStipple(1,0x0101);        /* dotted*/
glLineStipple(1,0x00FF);        /* dashed*/
glLineStipple(1,0x1C47);        /* dash/dot/dash*/
```

❖ برای ارتباط با برنامه از طریق keyboard از تابع

glutKeyboardFunc(void (*func)(unsigned char key,int x, int y))

❖ کد اسکی کلید فشرده شده را دریافت کرده، براساس عملیات آن مورد نظر انجام می شود.

❖ برای ارتباط برنامه از طریق mouse از تابع

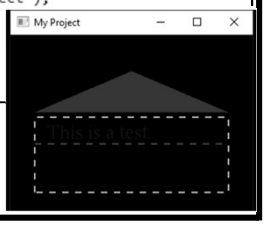
glutMouseFunc(void (*func)(int button, int state,int x, int y));

❖ button های راست (GLUT_RIGHT_BUTTON)، چپ (GLUT_LEFT_BUTTON) و وسط (GLUT_MIDDLE_BUTTON) برای موس وجود دارند. می توان برای هر کدام عملیاتی را درنظر گرفت.

مثال با OpenGL

```
using namespace std;
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glEnable(GL_LINE_STIPPLE);
    glColor3f(1.0, 0.0, 1.0);
    glLineStipple(1,0x00FF);
    glLineWidth(2.2);
    glBegin(GL_LINE_STRIP);
    glVertex3f(0.1, 0.55, 0.0);
    glVertex3f(0.9, 0.55, 0.0);
    glEnd();
    glColor3f(0.75, 0.75, 0.0);
    glBegin(GL_LINE_LOOP);
    glVertex3f(0.1, 0.35, 0.0);
    glVertex3f(0.1, 0.66, 0.0);
    glVertex3f(0.9, 0.66, 0.0);
    glVertex3f(0.9, 0.35, 0.0);
    glEnd();
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0,0);
    glVertex3f(0.1, 0.68,0.0);
    glVertex3f(0.5, 0.85,0.0);
    glVertex3f(0.9,0.68,0.0);
    glEnd();
    glColor3f(0.0, 0.0, 0.97);
    glRasterPos2f(0.15, 0.57);
    char str[] = "This is a test.";
    for (int i = 0; i<strlen(str); i++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, str[i]);
    glFlush();
}
```

```
void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(300, 300);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("My Project");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



نیاز به محاسبات موازی

- پردازش داده ها یا انجام محاسبات بسیار زیاد در مدت زمان قابل قبول
- کامپیوتر موازی (parallel computer)
  - یک کامپیوتر با تعداد زیادی از واحدهای پردازشی یا پردازنده ها
- حل مسأله با استفاده از یک کامپیوتر موازی:
  - تقسیم مسأله به تعدادی زیرمسأله
  - حل همزمان زیرمسائل با حل هر زیرمسأله به وسیله یک پردازشگر جداگانه
  - ترکیب نتایج برای به دست آوردن راه حل مسأله اصلی
- الگوریتم موازی
  - روش حل یک مسأله به منظور اجرا بر روی یک کامپیوتر موازی

مدلهای محاسباتی

- هر کامپیوتر، چه ترتیبی و چه موازی، با اجرای دستورالعمل ها روی داده ها عمل می کند.
  - جریان دستورالعمل ها در هر مرحله به کامپیوتر می گوید چه کاری انجام دهد.
  - جریان داده ها از این دستورالعمل ها تاثیر می گیرند.
- طراحی الگوریتم های موازی نیازمند داشتن درک درستی از مدل های محاسباتی است.
- چهار دسته از معماری کامپیوترها که توسط Flynn در ۱۹۶۶ ارائه شده است:
- یک دستورالعمل، یک داده (SISD)
- چند دستورالعمل، یک داده (MISD)
- یک دستورالعمل، چند داده (SIMD)
- چند دستورالعمل، چند داده (MIMD)

یک دستورالعمل، چند داده (SIMD)

- دستورالعمل می تواند ساده (جمع یا مقایسه دو عدد) یا پیچیده (ادغام دو لیست) باشد.
- داده می تواند ساده (یک داده)، یا پیچیده (چند عدد) باشد.
- در این دسته بندی، یک کامپیوتر موازی شامل N پردازنده یکسان است.
- همه پردازنده ها تحت کنترل یک جریان دستورالعمل ساده که توسط واحد کنترل مرکزی ارائه می شود عمل می‌کنند. N جریان داده برای N پردازنده وجود دارد.
- ارتباط بین پردازنده ها برای انتقال داده ها در کامپیوترهای SIMD به دو روش است:
  - shared memory
  - interconnection network

Parallel programming with **OpenMP**

برنامه نویسی موازی

- تنظیمات در visual studio، C++ و از نوع console application
- در مسیر زیر:
  - Properties-> configuration-> C/C++-> languages-> OpenMP support-> Yes
- اضافه کردن کتابخانه OpenMP
  - #include <omp.h>
- هر کدی که در دستور زیر قرار بگیرد به تعداد core ها به صورت موازی اجرا می شود.

```
#pragma omp parallel
 {
  // Code inside this region runs in parallel
  cout <<"Hello!\n";
 }
```

Parallel programming with **OpenMP**

برنامه نویسی موازی

❖ در کد زیر حلقه تقسیم می شود بین core ها و هر core قسمت مربوط به خود را انجام می دهد و ممکن است ترتیب خروجی ها نامرتب باشد.

```
#pragma omp parallel
 #pragma omp for
 for( i=0; i<100; i++)
 {
  cout<<i<<"\t";
 }
 cout<<"\n";
```

❖ یک نوع خروجی

```
75  76  77  78  79  80  81  82  83  84
85  86  87  88  89  90  91  92  93  94
95  96  97  98  99  25  26  27  28  29
30  31  32  33  34  35  36  37  38  39
40  41  42  43  44  45  46  47  48  49
50  51  52  53  54  55  56  57  58  59
60  61  62  63  64  65  66  67  68  69
70  71  72  73  74  0   1   2   3   4
5   6   7   8   9   10  11  12  13  14
15  16  17  18  19  20  21  22  23  24
```

Parallel programming with **OpenMP**

برنامه نویسی موازی

❖ تاثیر موازی سازی در زمان اجرا

```
double startTime,endTime;
startTime=omp_get_wtime();
 #pragma omp parallel private(i) or shared (i)
 #pragma omp for
 for( i=0; i<100000000; i++)
 {
  cout<<omp_get_num_threads()<<"\t";
  cout<<i<<"\t";
 }
 endTime=omp_get_wtime();
cout<<"\n Time elapsed:"<<endTime-startTime<<"\n";
```

زمان

تعداد threadها

Parallel programming with **MPI**

برنامه نویسی موازی

- تنظیمات در visual studio، C و از نوع console application
- نصب Microsoft Compute Cluster Pack SDK
- در مسیر زیر:

- Properties-> configuration->All configurations
- Properties-> configuration-> C/C++-> General-> Additional Include Directories-> C:\Program Files\Microsoft Compute Cluster Pack\Include
- Properties-> configuration-> C/C++-> Advanced->Compile As-> Compile as C Code (/TC)
- Properties-> configuration->Linker->Additional Library Directories-> C:\Program Files\Microsoft Compute Cluster Pack\Lib\i386
- Properties-> configuration->Linker->Input-> msmpi.lib

- اضافه کردن کتابخانه MPI

- #include "mpi.h"

Parallel programming with **MPI**

برنامه نویسی موازی

```
#include "stdafx.h"
#include "mpi.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
 printf ("Hello\n");
  return 0;
}
```

- برای اجرای این کد باید در command prompt
  - …../mpiexec –n 4 nameOfFile
- خروجی

```
Hello
Hello
Hello
Hello
```

Parallel programming with **MPI**

برنامه نویسی موازی

```
#include "stdafx.h"
#include "mpi.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
 int  nTasks, rank ;
 MPI_Init(&argc,&argv);
 MPI_Comm_size(MPI_COMM_WORLD,&nTasks);
 MPI_Comm_rank(MPI_COMM_WORLD,&rank);
 printf ("Number of threads = %d, My rank = %d\n", nTasks, rank);
 MPI_Finalize();
 return 0;
}
```

❖ خروجی

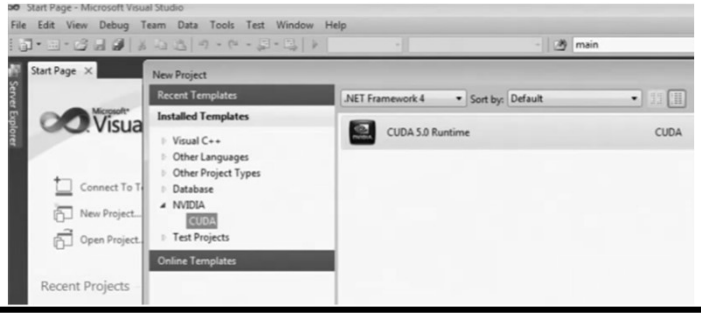
```
Number of threads = 4, My rank = 2
Number of threads = 4, My rank = 0
Number of threads = 4, My rank = 1
Number of threads = 4, My rank = 3
```

Parallel programming with **CUDA**
(Visual studio C++ 2010)

برنامه نویسی موازی

❖ روش مناسبی جهت موازی کردن برنامه ها با استفاده از GPU

- https://developer.nvidia.com/cuda-zone
- https://developer.nvidia.com/cuda-toolkit
- Download and install a version of cuda
- NewProject->NVIDIA->CUDA



برنامه نویسی موازی

```
#include <stdio.h>
#define SIZE 1024
Void VectorAdd(int *a, int *b, int *c, int n)
{
    int i;
    for(i=0;i<n;i++)
        c[i]=a[i]+b[i];
}
int main()
{
    int *a,*b,*c;
    a=(int *) malloc(SIZE*sizeof(int));
    b=(int *) malloc(SIZE*sizeof(int));
    c=(int *) malloc(SIZE*sizeof(int));
    for(int i=0;i<SIZE;i++)
    {
        a[i]=i;
        b[i]=i;
        c[i]=0;
    }
    VectorAdd(a, b, c, SIZE);
    for(int i=0;i<SIZE;i++)
        printf("c[%d]=%d\n",i,c[i]);
    free(a);
    free(b);
    free(c);
    return 0;
}
```

۱) موازی سازی VectorAdd

۲) گرفتن فضا در GPU و ارسال داده ها به GPU

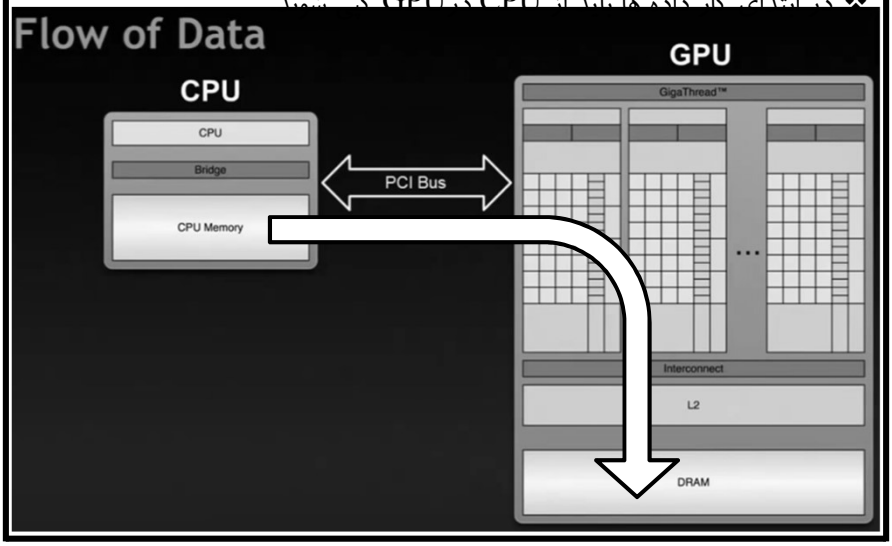
۳) تغییر دادن نحوه صدا زدن VectorAdd برای شروع اجرا در GPU

- ❖ در ابتدای کار داده ها باید از CPU درGPU کپی شوند.
- ❖ عملیات در GPU انجام شود.
- ❖ نتیجه از GPU به CPU کپی شود.

Parallel programming with **CUDA**
(Visual studio C++ 2010)

برنامه نویسی موازی

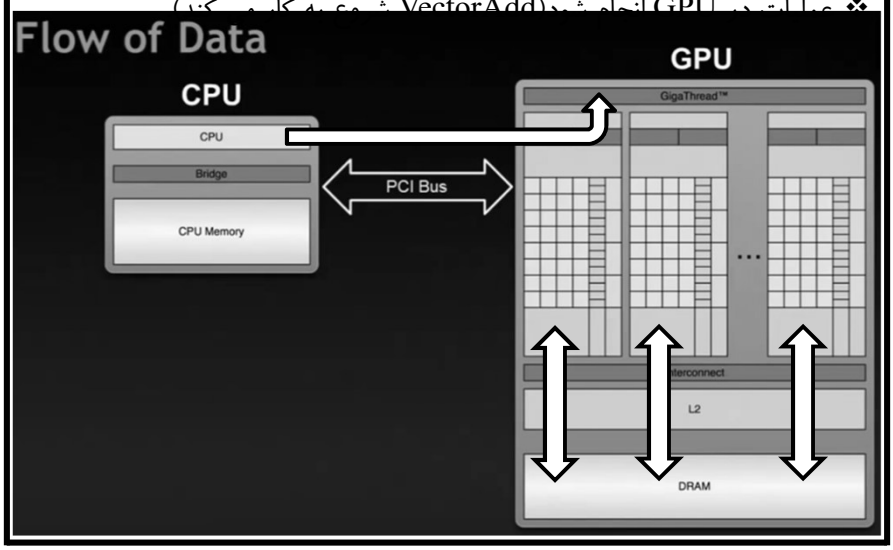
❖ در ابتدای کار داده ها باید از CPU در GPU کپی شوند



Parallel programming with **CUDA**
(Visual studio C++ 2010)

برنامه نویسی موازی

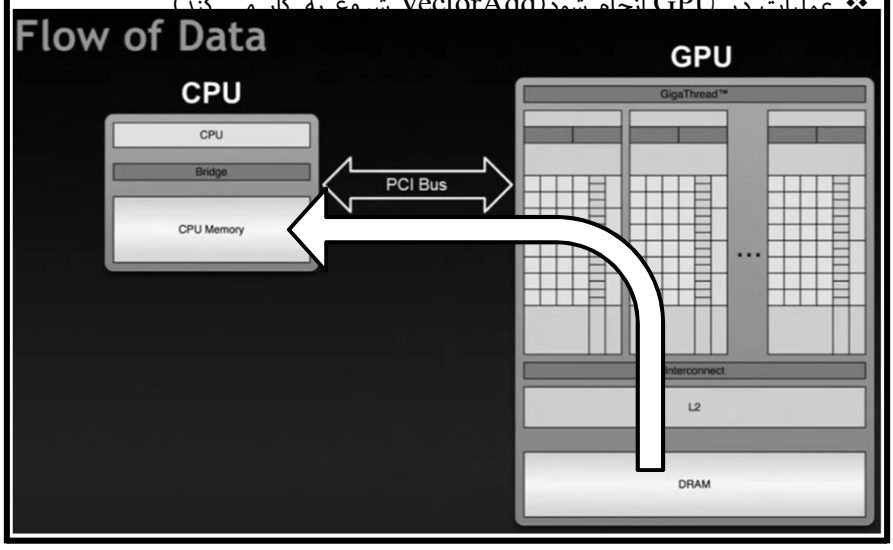
❖ عملیات در GPU انجام شود(VectorAdd شروع به کار می کند)



Parallel programming with **CUDA**
(Visual studio C++ 2010)

برنامه نویسی موازی

❖ عملیات در GPU انجام شود(VectorAdd شروع به کار می کند)



Parallel programming with **CUDA**
(Visual studio C++ 2010)

برنامه نویسی موازی

۱) موازی سازی VectorAdd

❖ روش مناسبی جهت موازی کردن برنامه ها با استفاده از GPU

➢ __global__

❖ به کامپایلر اعلام می کند این کد در GPU اجرا می شود.

➢ i=threadIdx.x

❖ یک thread روی یک عنصر کار می کند.

➢ if(i<n)

❖ نیازی به حلقه for نیست فقط باید چک شود که آیا داده دیگری هست یا خیر.

Parallel programming with **CUDA**
(Visual studio C++ 2010)

برنامه نویسی موازی

۲) گرفتن فضا در GPUو ارسال داده ها به GPU

❖ حافظه گرفتن در GPU

➢ cudaMalloc(&d_a, SIZE*sizeof(int));

❖ کپی کردن داده ها در GPU

➢ cudaMemcpy(d_a,a,SIZE*sizeof(int), cudaMemcpyHostToDevice);

❖ کپی کردن داده ها در CPU بعد از انجام عملیات

➢ cudaMemcpy(c,d_c,SIZE*sizeof(int), cudaMemcpyDeviceToHost);

❖ آزاد کردن فضای حافظه در GPU

➢ cudaFree(d_a);

۳) تغییر دادن نحوه صدا زدن VectorAdd برای شروع اجرا در GPU

➢ VectorAdd<<<1,SIZE>>>(d_a, d_b, d_c, SIZE);

Parallel programming with **CUDA**
(Visual studio C++ 2010)

برنامه نویسی موازی

```
#include <stdio.h>
#define SIZE 1024
__global__ Void VectorAdd(int *a, int *b, int *c, int n)
{
    int i=threadIdx.x;
    if(i<n)
        c[i]=a[i]+b[i];
}
int main()
{
    int *a,*b,*c;
    int *d_a,*d_b,*d_c;

    a=(int *) malloc(SIZE*sizeof(int));
    b=(int *) malloc(SIZE*sizeof(int));
    c=(int *) malloc(SIZE*sizeof(int));
    cudaMalloc(&d_a, SIZE*sizeof(int));
    cudaMalloc(&d_b, SIZE*sizeof(int));
    cudaMalloc(&d_c, SIZE*sizeof(int));

    for(int i=0;i<SIZE;i++)
    {
        a[i]=i;
        b[i]=i;
        c[i]=0;
    }
                                    }
```

```
cudaMemcpy(d_a,a,SIZE*sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_b,b,SIZE*sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_c,c,SIZE*sizeof(int), cudaMemcpyHostToDevice);

VectorAdd<<<1,SIZE>>>(d_a, d_b, d_c, SIZE);

cudaMemcpy(c,d_c,SIZE*sizeof(int), cudaMemcpyDeviceToHost);

for(int i=0;i<SIZE;i++)
    printf("c[%d]=%d\n",i,c[i]);

free(a);
free(b);
free(c);

cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
return 0;
```