

### جملات فیوناچی 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 تغییری 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 تغییری 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...

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

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

const

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

```
typedef OldDataType NewName;
```

typedef

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;
```

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;
}
```

## تمرین

- برنامه‌ای بنویسید که
- ❖ ۵۰ اسم را دریافت کرده، تعداد اسامی که با B شروع می‌شود را نمایش دهد.
  - ❖ عدد را دریافت کرده، تعداد اعدادی را که به ۴ ختم می‌شوند، نمایش دهد.
  - ❖ دو رشته را از ورودی دریافت کرده، به هم پیوند دهد و در خروجی نمایش دهد.
  - ❖ مقادیر n و k 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.
- $$y = \begin{cases} x^3 + 4x + 5 & x \geq 0 \\ 1 & x < 0 \end{cases}$$
- ❖ عناصر آرایه را گرفته مکعب آنها را محاسبه و در همان عنصر ذخیره نمایید.
  - ❖ 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.
  - ❖ 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.
  - ❖ یک عدد بین ۱ تا ۱۲ دریافت کرده معادل ماه مربوطه را نمایش دهد.
  - ❖ 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.

## 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 می‌گیرد تا به طول آرایه تعریف شده برسد. ممکن است کلمه مورد نظر کاربر ۳ حرفی باشد برای حل این مشکل از

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

روش زیر استفاده کنیم:

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

### مثال

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

<code>#include "stdafx.h"</code>	EOF به معنی End of
<code>#include &lt;iostream&gt;</code>	File می باشد که در
<code>using namespace std;</code>	تعریف <code>iostream.h</code>
<code>int main(){</code>	شده و مقدار آن برابر با
<code>char x;</code>	۱- می باشد. مقدار آن در
<code>while((x = cin.get( )) !=EOF)</code>	سیستم عامل DOS
<code>cout &lt;&lt; x ;</code>	عبارتست از <code>ctrl z</code>
<code>system("pause");</code>	
<code>return 0;</code>	
<code>}</code>	

### #define

از `define` جهت تعریف یک شناسه و یک دنباله کاراکتری استفاده می شود طوری که در `source` برنامه هر جا با آن شناسه برخورد شود آن دنباله کاراکتری جانشین آن گردد.

```
#define macro-name character-sequence

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

#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;
}
```

## 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 آرایه‌ها هستند، بدین صورت که یک نوع داده گروهی (جمعی) است که فضای پیوسته از حافظه اصلی را اشغال می‌نماید. اما عناصر ساختار الزاماً از یک نوع نمی‌باشند بلکه اعضای یک ساختار می‌توانند از نوع‌های مختلف مانند float، int و ... باشند. اعضای ساختار ربط منطقی دارند. اعضای ساختار را field یا element گویند.

## 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 :

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

روش اول

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

روش دوم

struct

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

مقدار اولیه برای ساختارها:

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

بمنظور دسترسی به عناصر یک ساختار از عملگر (dot). استفاده می گردد.



**struct**

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

```
struct date {
int month;
int day;
int year;
};
struct Info{
    Info x;
int Id;
char name[80];
date entrance; };
x.entrance.month=10;
x.entrance.day=22;
x.entrance.year=1378;
```

**struct**

عضو یک ساختار خود می تواند 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

```

#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;
}

```

نام خانوادگی دانشجویان بزرگتر از ۱۸ سال

تمرین

برنامه‌ای بنویسید که

- ❖ با استفاده از ساختار، طول و عرض یک مستطیل را دریافت کرده، مساحت چهار مستطیل را محاسبه کرده و نمایش دهد.
- ❖ ۵ عدد را دریافت و در آرایه قرار دهد، اختلاف مجموع اعداد زوج و فرد را نمایش دهد.
- ❖ 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.
- ❖ 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;
```

می توان 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 word یا byte 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 عموماً 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

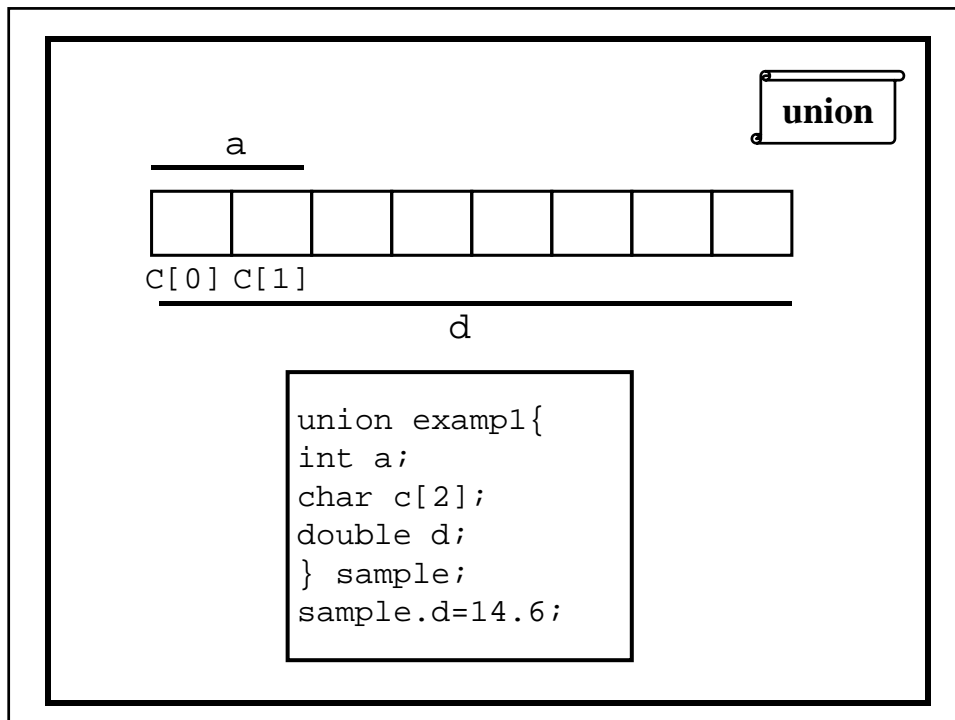
☼ union از نظر ساختاری شبیه struct می باشد. با این تفاوت که عضوهائی که تشکیل union می دهند همگی از حافظه مشترکی در کامپیوتر استفاده می نمایند. بنابراین استفاده از union باعث صرفه جوئی در حافظه می گردد.

☼ union از یک مکان ثابت حافظه که بین یک یا چند متغیر به اشتراک گذاشته شده باشد، تشکیل می گردد.

☼ 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 می توان 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 & decoder .encoder

```

#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;
}

```

### 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 بایستی از تابع rand() استفاده نمود.

```
#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 برنامه قبل را اجرا نمائیم جواب یکسانی را می‌گیریم. برای تصادفی کردن اعداد می‌بایستی 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 . . .

نتیجہ پر تاب 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 ...



### کلاس های حافظه

✓ متغیرها بوسیله نوع (type) و کلاس حافظه متمایز می‌شوند. نوع متغیر می‌تواند int، float، double و ... باشد ولی کلاس حافظه یک متغیر در مورد طول عمر و وسعت و دامنه 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 می‌رود بعبارت دیگر وسعت و دامنه 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 برنامه‌ها دیده می‌شود.

### کلاس های حافظه

```
#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";
}
```

static

Output

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

Press any key to continue ...

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

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.

**static**

**کلاس های حافظه**

```

#include "stdafx.h"
#include <iostream>
using namespace std;
void func(void);
static int count = 10; /* Global variable */
int main()
{
    while(count-->0)
    {
        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...

**static**

**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 می شود.

### external

### کلاس های حافظه

```

#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 ;
}

```

متغیرهای از نوع external متغیرهائی هستند که بیرون از توابع اعلان می شوند و وسعت و دامنه فعالیت آنها کلیه توابعی می باشد 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; ✓

**Output**  
 4  
 3  
 0  
 7  
 7  
 11  
 1  
 3  
 Press any key to continue ...

### register

### کلاس های حافظه

✓ وقتی متغیری از نوع register اعلان می شود از کامپیوتر عملاً درخواست می شود که به جای حافظه از یکی از رجیسترهای موجود استفاده نماید.  
 ✓ معمولاً از نوع رجیستر برای شاخص های دستور تکرار و یا اندیسهای آرایه ها استفاده می شود. متغیرهای از نوع رجیستر قابل استفاده در دستور cin نمی باشند.  
 ✓ متغیرهای از نوع register عمدتاً متغیرهایی هستند که تناوباً 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 بده 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 . . .

**اشاره گر**

**type \*var\_name;**

✓ اشاره گر 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.

```

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

```

The diagram shows a box labeled 'q' containing the number '144'. An arrow labeled 'p' points from below to the box, indicating that 'p' holds the address of 'q'.

اشاره گر

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

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

q
290

p
↖

p	q	r
102	1000	1000
100	102	104

↘
↖

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

اشاره گر

✓ 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;
}
```

**اشاره گر 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 ...

```
#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;
}
```

**output**  
003FF808 003FF7F0 003FF7D4  
003FF80C 003FF7F4 003FF7DC  
Press 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:0017FF08  
 \*p is:45  
 p is:0017FF0C  
 \*p is:-858993460  
 q is:45 Press 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:46 Press 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 اشاره گر

```
#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 اشاره گر

#### *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 جدول 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];
}
    
```

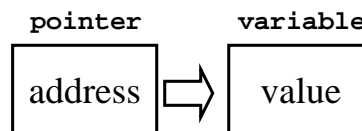
**output**  
 Disk error  
 Out of range  
 Paper out  
 Disk full  
 Press any key to continue ...

**اشاره گر به جای ایندکس کردن آرایه ها**

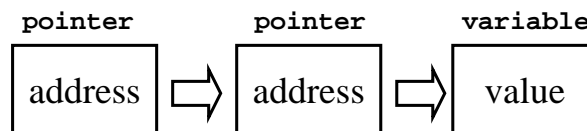
<pre> #include "stdafx.h" #include &lt;iostream&gt; using namespace std;  int main() {     int i;     char s[]="test in";     for(i=0; s[i];i++)     {         s[i]=toupper(s[i]);         cout&lt;&lt;s[i];     }     system("pause");     return 0; }                 </pre>	<pre> #include "stdafx.h" #include &lt;iostream&gt; using namespace std;  int main() {     char s[]="test in";     char *p;     p=s;     while(*p)     {         *p=toupper(*p);         cout&lt;&lt;*p;         p++;     }     system("pause");     return 0; }                 </pre>
--	---

## رجوع غیر مستقیم چندگانه multiple indirection

اشاره گری که به اشاره گر دیگری اشاره کند.



*Single 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;
}
```

## multiple indirection

### output

```
ch:A
*p:A
**mp:A Press 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- در آرگومان 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 اشاره گر به تابع ارسال شود، هر چه را که اشاره گر به آن اشاره می کند، تغییر خواهد 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:

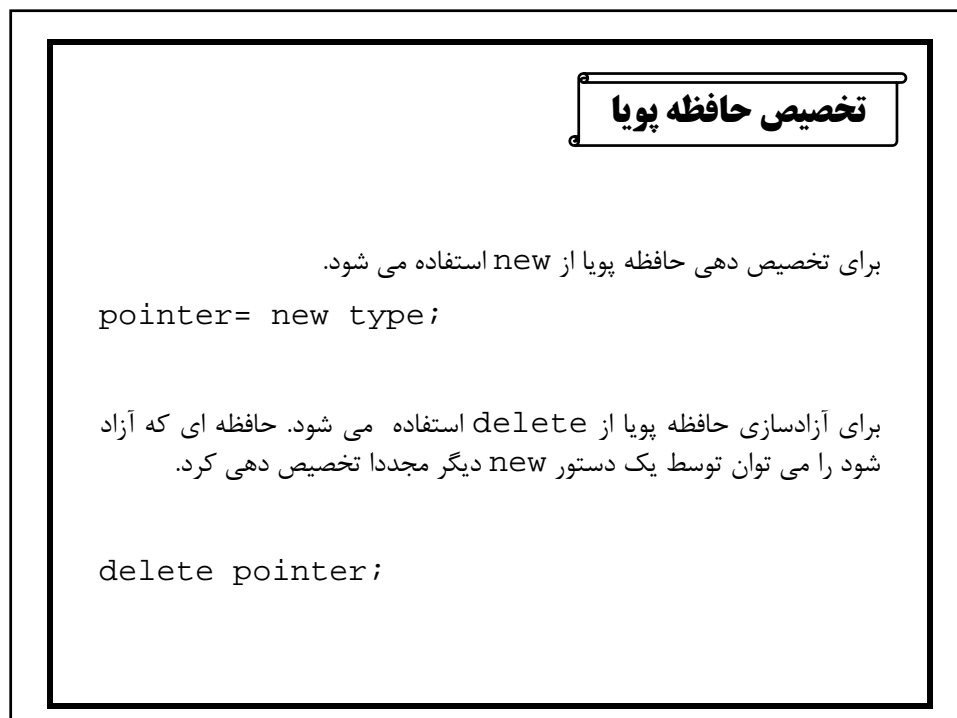
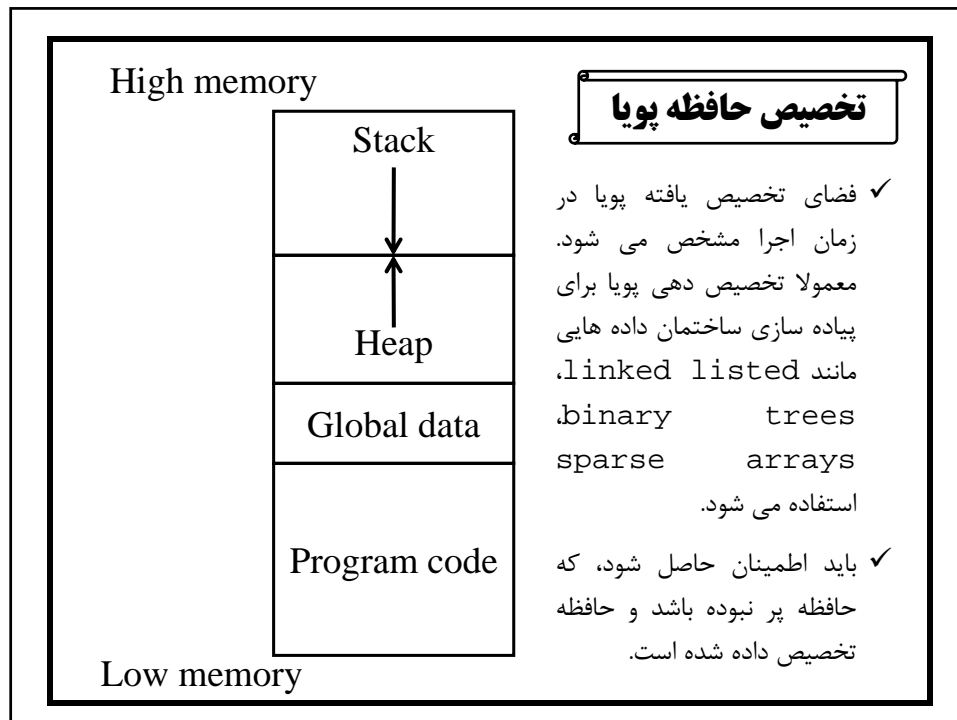
۱- استفاده از متغیرها

فضای ذخیره سازی که برای هر متغیر در نظر گرفته می‌شود، در زمان کامپایل ثابت است و در طول اجرای برنامه تغییر نمی‌کند.

۲- تخصیص حافظه پویا(دینامیک)

در این حالت برنامه می‌تواند 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 می‌شود. به این ناحیه heap گفته می‌شود.



```

#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;
}

```

مثالی از تخصیص دهی و آزادسازی حافظه

```

#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:

```
#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;
}
```

ذخیره تعدادی داده در آرایه و نمایش میانگین

### شی گرای

☞ 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 recursion و 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 امکان 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 شوند که 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 گرای

☞ 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 داده ها 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 گرای

↪ 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. کامپایلر خود 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.

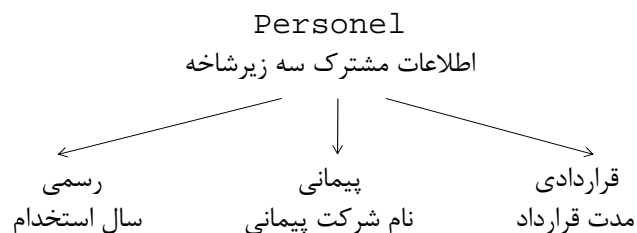
```
int abs(int a);
int abs(int a , int b);
long abs(long a);
float abs(float a);
```

unique signature ↪

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

### شی گرای

↪ 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

```

#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;
}

```

تکمیل شده کلاس فوق:

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 مقدار بازگشتی 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;
}

```

محیط مستطیل

```

#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

کلاس time

```

#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

تخصیص حافظه پویا




**class**
**❖ توابع inline**

- ✓ توابعی که داخل بدنه کلاس تعریف می شوند **inline** در نظر 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;
}
```


**inline**

## 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

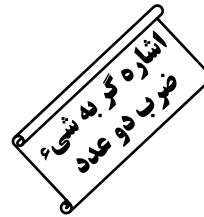
```
#include "stdafx.h"
#include <iostream>
using namespace std;
class test{
    int i, j ;
public:
    test(int n, int m);
    int meanij();
};
test::test(int n, int m){
i=n;
j=m;
}
int test::meanij(){
return (i+j)/2;
}
int main(){
    int a,b;
    cin>>a>>b;
    test O1(a,b);
    test *p;
    p=&O1;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

```

#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;
}

```

class



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 گونه ای از B به ارث می برد که همه اعضای عمومی کلاس مبنا، اعضای عمومی کلاس مشتق شده خواهند بود.

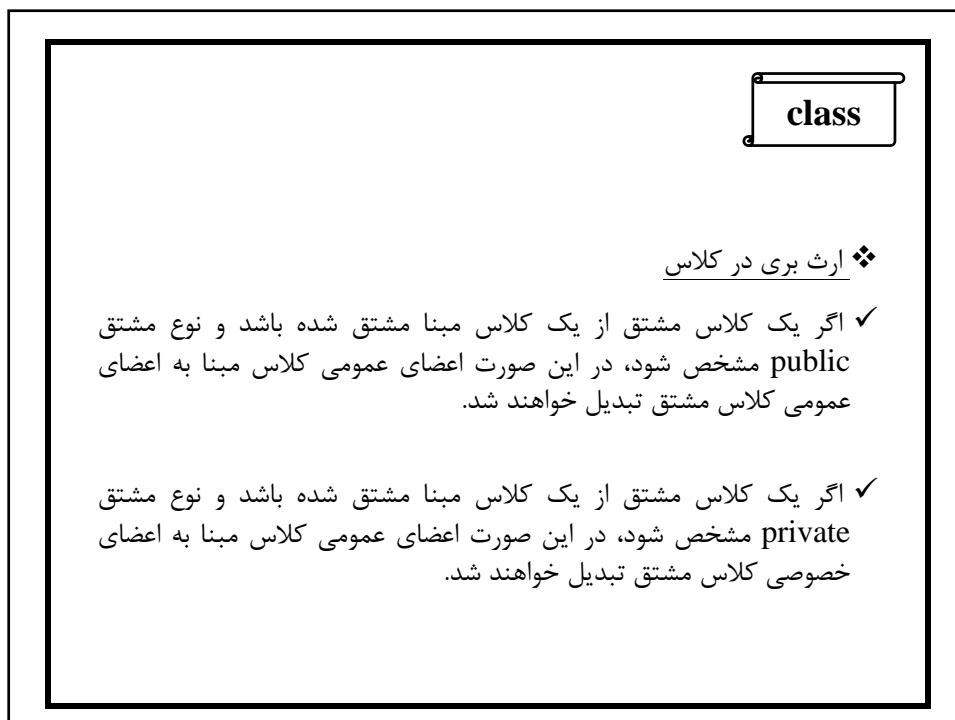
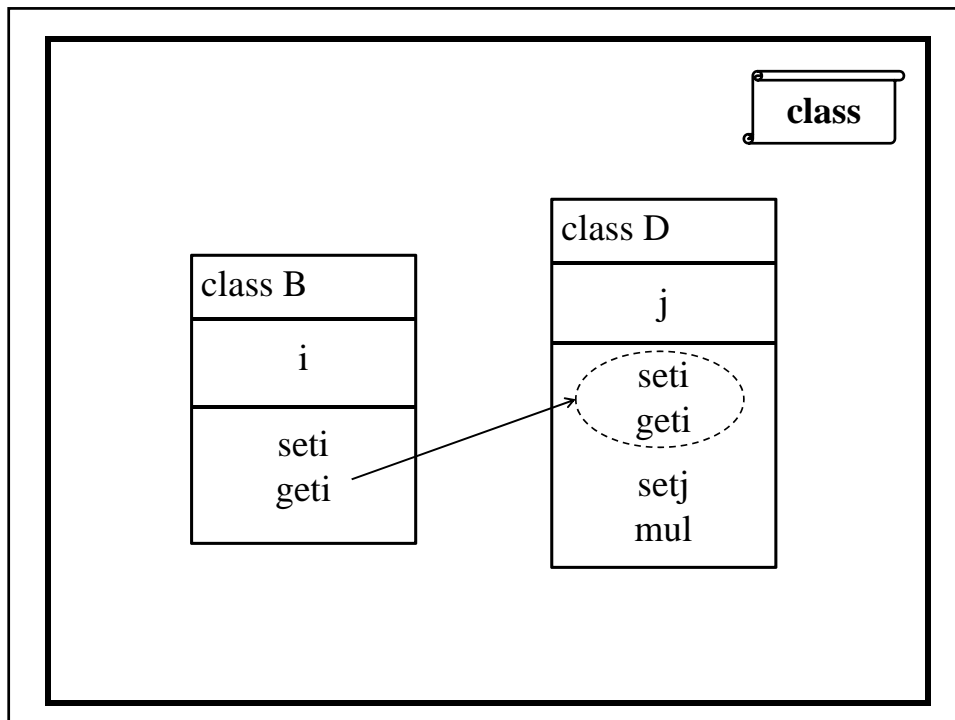
## class

```
#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();}
```

### output

80Press any key to continue . . . . .

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



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 مینا در کلاس هایی که از آن مشتق می شوند قابل دستیابی خواهند بود. اعضای محافظت شده، خارج از این کلاس مینا 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 مینا به صورت public، توسط یک کلاس مشتق به ارث برده می شود به عضو محافظت شده آن کلاس مشتق تبدیل می گردد. اگر کلاس مینا به صورت private به ارث برده شود در آن صورت اعضای محافظت شده آن به اعضای خصوصی کلاس مشتق تبدیل خواهند شد.

✓ کلاس مینا را یک کلاس مشتق می تواند به صورت protected نیز به ارث ببرد. در چنین مواردی اعضای عمومی و محافظت شده کلاس مینا به اعضای محافظت شده کلاس مشتق شده تبدیل می شوند.

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


class

```

#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 پیش از یک کلاس دوست شوند.

```

#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

جمع دو عدد  
تابع دوست

```

#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

مثال



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.

~~01.isfactor();~~

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

✓ کلاسی که از کلاس دیگر ارث می برد، توابع دوست آن کلاس را به ارث نمی برد. بعبارت دیگر یک (کلاس مشتق) derived class. توابع دوست را به ارث نمی برد.

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.fl();

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

```

#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

```

#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**

مثال با this

class

```

#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;
}

```

**❖ کلاس های دوست friend classes**

✓ یک کلاس می تواند دوست کلاس دیگری باشد. در این حالت تابع 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.

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 مجدداً تعریف می کند. تابع های مجازی فلسفه یک رابط چند کار که همان مفهوم پلی مورفیسیم 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;
}
```



↩ 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 می شود.


↩ این فایل کلاس های

ifstream (ورودی)،

ofstream (خروجی)،

fstream (ورودی و خروجی)

↩ را تعریف می کند.



↩ باز کردن فایل

↩ پس از آنکه stream 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 چگونگی 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.

<div style="float: right; border: 1px solid black; padding: 2px;">file</div> 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	در زمان باز شدن یک فایل، نشانگر آن فایل به انتهای آن برده می شود اما در هر نقطه ای از فایل 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 ساخته می شود.
ios::nocreate	اگر فایل مورد نظر وجود نداشته باشد، فایل تازه ای ایجاد نمی کند و با شکست مواجه می شود.
ios::noreplace	اگر فایل مورد نظر وجود داشته باشد، فایل تازه ای ایجاد نمی کند و با شکست مواجه می شود.

```

#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;
}

```

file

ایجاد فایل، متنی



↩ 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);
```



```
#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;
}
```




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.

file

```
#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; }
```

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;
}
```



**کد ASCII**

☞ هر کاراکتر یک کد ASCII دارد که می توان در برنامه نویسی از آن استفاده نمود.

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

☞ کد اسکی ۲۷ مربوط به کلید Esc است.

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