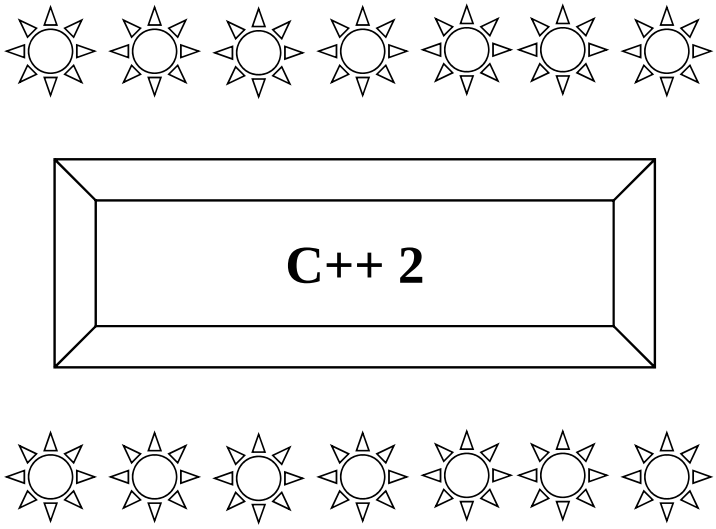


C++ 2

2014 *M. Damrudi*

struct

هر ساختار از دو يا چند عضو كه به همراه هم يك واحد منطقي را مي‌سازند، تشكيل مي‌شود.

ساختارها شبيه آرايه‌ها هستند، بدين صورت كه يك نوع داده گروهي(جمعی) است كه فضاي پيوسته از حافظه اصلي را اشغال مي‌نمايد. اما عناصر ساختار الزاماً از يك نوع نمي‌باشند بلكه اعضاي يك ساختار مي‌توانند از نوع‌هاي مختلف مانند: `char`، `int`، `float`و ... باشند.

اعضای ساختار ربط منطقی دارند. اعضای ساختار `field` یا `element` گویند.

M. Damrudi

struct

حالت کلی ساختار به صورت زیر است. وجود نام ساختار یا متغیرها اختیاری است ، اما حداقل یکی باید ذکر شود.

```
struct name{
type member1;
type member2;
.
.
.
.
.
}variables;
```

M. Damrudi

مثال

نام ساختار

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

M. Damrudi

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;
```

M. Damrudi

struct

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

روش دوم

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

M. Damrudi

struct

مقدار اولیه برای ساختارها:

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

M. Damrudi

struct

بمنظور دسترسي به عناصر يك ساختار از عملگر(dot) . استفاده مي‌گردد .

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

M. Damrudi

struct

عضو يك ساختار خود مي‌تواند يك ساختار ديگر 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;
```

M. Damrudi

struct

عضو يك ساختار خود مي‌تواند يك ساختار ديگر باشد. (روش ديگر)

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

M. Damrudi

struct

آرایه ای از ساختارها:

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

M. Damrudi

مثال

نام خانوادگی دانشجویان 18 سال

```
#include <iostream.h>
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;
cin>>st3[i].age;}
for(i=0;i<100;i++){
if(st3[i].age==18)
cout<<st3[i].family<<"\n";
}
return 0;}
```

M. Damrudi

enum

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

مي‌توان نوع داده مجتمعي ساخت كه از ليستي از ثابت‌هاي صحيح داراي اسم تشكيل 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 <iostream.h>
enum language{C,CPP,Pascal,BASIC,Ada};
int main(){
language p;
for(p=C;p<=Ada;p++){
   switch(p){
        case C: cout<<“C language”;
        break;
        case BASIC: cout<<“Basic language”;
        break; }
}
return 0;
}
```

M. Damrudi

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;
```

int type یا unsigned است.

اگر نوع علامت دار باشد، بیت با بالاترین مرتبه به عنوان علامت در نظر گرفته می شود.

M. Damrudi

bit fields

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];
```

M. Damrudi

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;
};
```

M. Damrudi

union

- union از نظر ساختاري شبيه struct مي‌باشد . با اين تفاوت كه عضوهائي كه تشكيل unionميدهد همگي از حافظه مشتركي در كامپيوتر استفاده مي‌نمايند. بنابراين استفاده از union باعث صرفه‌جوئي در حافظه مي‌گردد.
- union از یک مکان ثابت حافظه که بین یک یا چند متغیر 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.

M. Damrudi

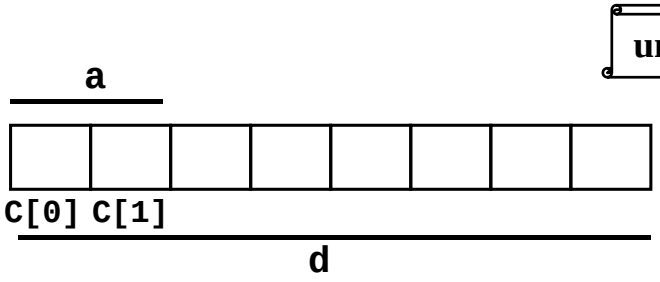
union

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

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

M. Damrudi

union




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

M. Damrudi

union ،decoder ،encoder

```
#include <iostream.h>
int encode(int i);
int main()
{
int i;
i=encode(10);
cout<<"10 encoded is:"<<i;
cout<<"\n";
i=encode(i);
cout<<" i decoded is:"<<i;
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;
 }
```

M. Damrudi

اشاره‌گر

- ✓اشاره‌گر متغیری است که آدرس حافظه یك شي 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.

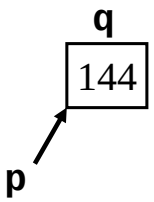
```
type *var_name;
```

M. Damrudi

اشاره‌گر

رجوع به مقدار یک متغیر با اشاره گر رجوع غیرمستقیم indirection گفته می شود.

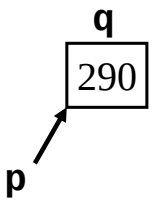
```
#include <iostream.h>
int main()
{
int *p, q;
q=144;
p=&q;
cout<<*p;
cout<<p;
return 0;}
```



M. Damrudi

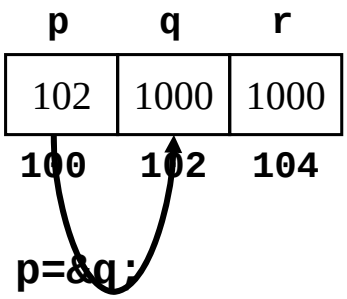
اشاره‌گر

```
#include <iostream.h>
int main()
{
int *p,q;
p=&q;
*p=290;
cout<<q;
return 0;
}
```



M. Damrudi

اشاره‌گر




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

M. Damrudi

اشاره‌گر

- از آنجایی که کوچکترین واحدی از حافظه که میتواند 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.

M. Damrudi

اشاره‌گر NULL

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.(null با هر header file تعریف می شود.)

```
#include <iostream.h>
int main(){
int *p=NULL;
if (!p) cout<<“P is null”;
return 0;
}
```

M. Damrudi

اشاره‌گر void

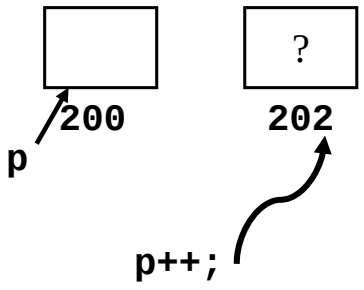
اشاره گر void با استفاده از *void تعریف می شود. این اشاره گر به هر نوع شیء می تواند اشاره کند. در صورتی می توان یک اشاره گر را به اشاره گر دیگری نسبت داد که نوع داده آنها با هم سازگار باشند. اما بدون نیاز به استفاده از قالب بندی نوع هر اشاره گری را می تو.ان به یک اشاره گر void نسبت داد.

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

M. Damrudi

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.



M. Damrudi

```
#include <iostream.h>
#include <conio.h>
int main()
{
int *p;
int q;
clrscr();
p=&q;
cout<<p<<"\n";
p++;
cout<<p;
return 0;}
```

output
0x8fdffff4
0x8fdffff6

M. Damrudi

```
#include <iostream.h>
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";
return 0;
}
```

output
0x8f95ffec 0x8f95ffe6
0x8f95ffee 0x8f95ffea

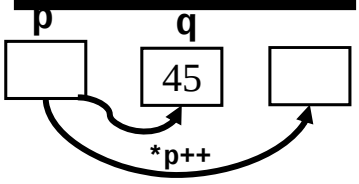
M. Damrudi

```
#include <iostream.h>
#include <conio.h>
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;
getch();
return 0;}
```

نکته

output

```
p is:0x8fd8fff4
*p is:45
p is:0x8fd8fff6
*p is:0
q is:45
```



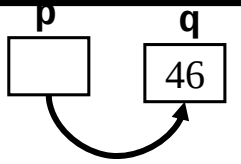
M. Damrudi

```
#include <iostream.h>
#include <conio.h>
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;
getch();
return 0;}
```

نکته

output

```
p is:0x8fd8fff4
*p is:45
p is:0x8fd8fff4
*p is:46
q is:46
```



M. Damrudi

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.

M. Damrudi

اشاره گر و آرایه

```
#include <iostream.h>
#include <conio.h>
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];
getch();
return 0;
}
```

outpt
1 2 3
1 2 3

M. Damrudi

کپی کردن محتویات یک رشته به ترتیب عکس ورودی با اشاره‌گر

```
#include <iostream.h>
#include <string.h>
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 str1
while(p1>= str1){
*p2=*p1;
p1--;   p2++;}
*p2='\0';        // null terminate str2
cout<<"string1:"<<str1<<"string2:"<<str2;
return 0;}
```

M. Damrudi

کپی کردن محتویات یک رشته به ترتیب عکس ورودی با اشاره‌گر

output

```
string 1:Our class is advanced
string 2:decnavda si ssalc ruO
```

- 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.

M. Damrudi

اشاره گر و آرایه

❁ برای استفاده از یک اشاره گر چهت 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

M. Damrudi

آرایه ای از اشاره گرها

```
#include<iostream.h>
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);
return 0;}
void error(int num){
cout<< p[num];}
```

برای ساختن جداول رشته ای هم به کار می رود.

M. Damrudi

اشاره گر به جای ایندکس کردن آرایه ها

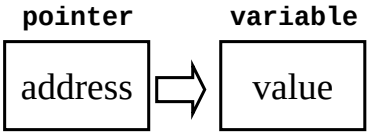
```
#include<iostream.h>
#include<ctype.h>
int main(){
int i;
char s[]="test in";
for(i=0; s[i];i++){
  s[i]=toupper(s[i]);
  cout<<s[i];}
return 0;}
```

```
#include<iostream.h>
#include<ctype.h>
int main(){
char s[]="test in";
char *p;
p=s;
while(*p){
  *p=toupper(*p);
   cout<<*p;
   p++;}
return 0;}
```

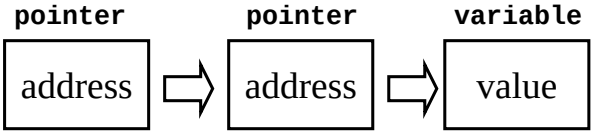
M. Damrudi

رجوع غیر مستقیم چندگانه multiple indirection

اشاره گری که به اشاره گر دیگری اشاره کند.



Single Indirection



Multiple Indirection

M. Damrudi

multiple indirection

```
#include <iostream.h>
#include <conio.h>
int main()
{
char **mp,*p,ch;
clrscr();
p=&ch;
mp=&p;
**mp='A';
cout<<"ch:"<<ch<<"\n*p:“;
cout<<*p<<"\n**mp:"<<**mp;
getch();
return 0;
}
```

M. Damrudi

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 <iostream.h>
char *findblank(char blank, char *s);
int main(){
char *p;
p=findblank(‘ ‘,”This is a Test”);
return 0;}
char *findblank(char blank, char *s){
while(blank !=*s && *s) s++;
return s;}
```

M. Damrudi

ارسال اشاره گر به تابع

1- پارامتر تابع از نوع اشاره گر باشد.

2- درآرگومان تابع به جای یک مقدار از آدرس استفاده می شود.

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

M. Damrudi

اشاره گر به ساختار

برای دسترسي به عناصر يك ساختار با اشاره‌گر از عملگر (arrow operator) -> استفاده مي‌گردد .

```
#include <iostream.h>
struct students{
  char name[40];
  int  age;
};
int main(){
students *st;
cin>>st->name>>st-> age;
if(st->age==18) cout<<st->name;
return 0;}
```

M. Damrudi

تخصیص حافظه پویا

ذخیره سازی اطلاعات در حافظه به دو صورت است:

1- 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.

2- 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.

M. Damrudi

تخصیص حافظه پویا

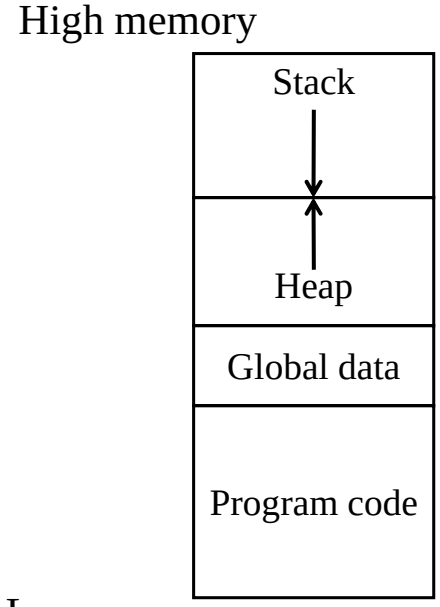
در این روش فضای ذخیره سازی یک داده در صورت نیاز از حافظه آزادی که بین برنامه و پشته (stack) قرار دارد تخصیص داده می شود. به این ناحیه heap گفته می شود.

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.

M. Damrudi

تخصیص حافظه پویا



M. Damrudi

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;
```

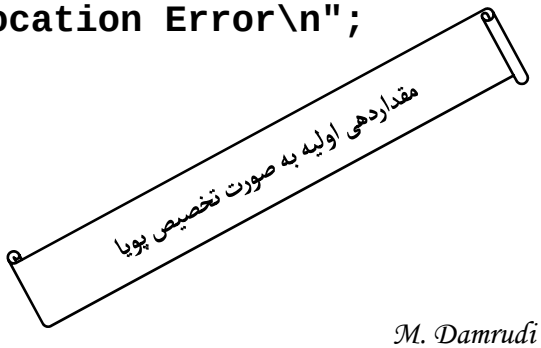
M. Damrudi

مثالی از تخصیص دهی و آزادسازی حافظه

```
#include <iostream.h>
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;
return 0;
}
```

M. Damrudi

```
#include <iostream.h>
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;
return 0;
}
```



M. Damrudi

تخصیص دهی آرایه ها

تخصیص دهی آرایه:

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

Size تعداد عناصر آرایه است.

```
delete[] pointer;
```

آزادسازی حافظه:

M. Damrudi

ذخیره تعدادی داده در آرایه و نمایش میانگین

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

M. Damrudi

#define

از **define** جهت تعريف يك شناسه و يك دنباله كاراكتري استفاده مي‌شود طوري كه در **source** برنامه هرجا با آن شناسه برخورد شود آن دنباله كاراكتري جانشين آن گردد.

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

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

M. Damrudi

مثال

```
#include <iostream.h>
int main()
{
#define myfor for(int k=10;k<15;k++)

myfor
    cout<<k;
return 0;
}
```

M. Damrudi

typedef

typedef OldDataType NewName;

با استفاده از typedef براي يك نوع داده موجود نام تازه‌اي ايجاد كرد.

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

M. Damrudi

شی گرایی

☜ برنامه‌هایی که تا کنون می 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.

M. Damrudi

شی گرایی

☜ 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 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

M. Damrudi

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.

M. Damrudi

شی گرایی

⇦ کپسوله سازی Encapsulation

☜ 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 تغیر مرکب (شیء) است.

M. Damrudi

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 پارامترها متفاوت باشد، سربارگذاری توابع (operator overloading) می گویند.

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

M. Damrudi

شی گرایی

پلی مورفیسم (چندریختی) Polymorphism

unique signature

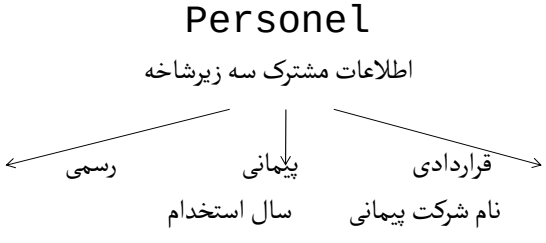
علائمی که باعث می شوند دو تابع توسط کامپایلر متمایز باشند. تعداد ، ترتیب ، نوع و ترکیب پارامترهای متفاوت

M. Damrudi

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.



M. Damrudi

class

کلاس‌ها ارکان برنامه نویسی شی‌گرا 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.

M. Damrudi

class

نمونه ای از یک کلاس:

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

a یک متغیر private 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.

M. Damrudi

class

تکمیل شده کلاس فوق:

```
#include <iostream.h>
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();
return 0;}
```

M. Damrudi

class

```
#include <iostream.h>
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();
return 0;}
```

روش دوم:

عملگر تعیین میدان
scope resolution operator ::

M. Damrudi

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.

M. Damrudi

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.

M. Damrudi

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.

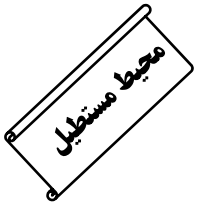
✓ تابع سازنده هم نام با نام کلاس خود است با این تفاوت که پیش از نام خود علامت ~را دارد و نوع مقداربازگشتی هم ندارد.

✓ 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.

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

class

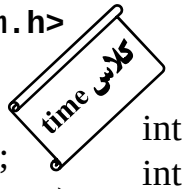



```
int main(){
int m,n;
cin>>m>>n;
rectangle ob(m,n);
cout<<ob.show();
return 0;
}
```

M. Damrudi

```
#include <iostream.h>
class time{
public:
time(int h, int m, int s);
void reset(int h,int m ,int s);
void show();
private:
int hour, min, sec;};
time::time(int h,int m,int s){hour=h;min=m;sec=s;}
void time::reset(int h,int m,int s){
hour=0; min=0; sec=0;}
void time::show(){
cout<<"time is:\n\t"<<hour<<":"<<min<<":"<<sec<<"\n";
}
```

class




```
int main(){
int saat,dagh,san;
cin>>saat>>dagh>>san;
time o(saat,dagh,san);
o.show(); }
```

M. Damrudi

class

```
#include <iostream.h>
#include <stdlib.h>
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);}
```

تخصیص حافظه پویا

```
int main(){
myclass  ob;
int k;
cin>>k;
ob.show(k);
return 0;
}
```

M. Damrudi

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.

M. Damrudi

class

inline

```
#include <iostream.h>
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();
return 0;
}
```

M. Damrudi

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.

M. Damrudi

class

```
#include <iostream.h>
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();
  return 0;
}
```

M. Damrudi

class

اشاره گر به شیء ضرب دو عدد

```
#include <iostream.h>
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  a*b;}
int main(){
int m,n;
cin>>m>>n;
zarb ob(m,n);
zarb *p;
p=&ob;
cout<<ob.show();<<"\n";
cout<<p->show();
return 0;
}
```

M. Damrudi

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.

M. Damrudi

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.

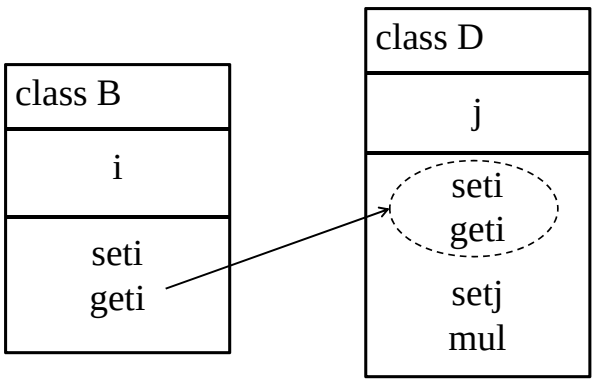
M. Damrudi

class

```
#include <iostream.h>
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();
  return 0;
}
```

M. Damrudi

class



M. Damrudi


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.
M. Damrudi


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.
M. Damrudi

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.

M. Damrudi

class

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

M. Damrudi

class

```
#include <iostream.h>
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<<"\n"<<a<<"\n"<<b<<"\n"<<c;}
};
```

مثال

```
int main(){
derived ob;
ob.setab(2,8);
ob.setc(5);
ob.showabc();
return 0;
}
```

M. Damrudi

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.

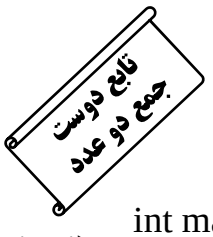
✓ تابع دوست همانند یک تابع معمولی غیر عضو تعریف می شود. پیش از پیش الگوی تابع friend قرار داده می شود.

✓ در تعریف تابع کلمه friend قرار داده نمی شود.

✓ تابع های دوست می توانند با بیش از یک کلاس دوست شوند.

M. Damrudi

class




```
#include <iostream.h>
class myclass {
int  a,b;
public :
friend int sum(myclass  x);
void set_ab(int i, int j);
};
void myclass :: set_ab(int i, int j)
{
        a=i; b=j;
}
int sum(myclass  x){
         return x.a + x.b;
}
```

```
int main()
{
myclass n;
n. set_ab(5,8);
cout << sum(n);
return 0;
}
```

M. Damrudi

class

```
#include <iostream.h>
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";
  else
    cout<<" 6 is n't factor of 18";
 if (isfactor(o2))
     cout<<" 6 is factor of 13";
else
    cout<<" 6 is n't factor of 13";
  return 0;
}
```

M. Damrudi

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.

✓کلاسي كه از كلاس ديگر ارث مي‌برد ، توابع دوست آن كلاس را به ارث نمي‌برند. بعبارت ديگر يك (كلاس مشتق) derived class ، توابع دوست را به ارث نمي‌برد.

M. Damrudi

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.

M. Damrudi

```
#include <iostream.h>
#include <string.h>
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;}
```

class

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

مثال

M. Damrudi

```
#include <iostream.h>
#include <string.h>
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<<ithis->tem<<"\n "<<this->cost;}
```

class

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

مثال با this

M. Damrudi

class

❖ کلاس های دوست friend classes

✓ يك كلاس می تواند دوست كلاس ديگري باشد. در اين حالت تابع دوست به كليه اسامي private تعريف شده در كلاس ديگر دسترسي دارد.

✓ در مثال صفحه بعد از کلاس دوست friendclass در کلاس CPP_Test تعریف شده است.

M. Damrudi

class

```
#include <iostream.h>
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);
return 0;    }
```

M. Damrudi

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:

1- در زمان کامپایل از طریق توابع و سربارگذاری

2- در زمان اجرا از طریق به کار بردن توابع مجازی

✓ تابع مجازي ، تابعي است که در یک کلاس مبنا تعریف شده و

توسط کلاسی که از آن کلاس مبنا مشتق خواهد شد مجددا تعریف می شود(تغیر داده مي شود).

M. Damrudi

class

❖ تابع های مجازی virtual functions

✓ برای معرفی تابع مجازي 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.

M. Damrudi

class

```
#include <iostream.h>
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();
return 0;
}
```

M. Damrudi

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 فایل سرآمد fstream.h استفاده می شود.

☜ این فایل کلاس های ifstream (ورودی)، ofstream (خروجی)، fstream (ورودی و خروجی) را تعریف می کند.

M. Damrudi

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.

می توان چند تا از مدها را با هم OR (|) کرد.

M. Damrudi

file

mode می تواند از حالات زیر باشد

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 خروجیی 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 شاخته 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.**

M. Damrudi

file

بستن فایل

برای بستن فایل از تابع close( ) استفاده می شود:

```
close();
```

تابع close( ) نه پارامتری می‌گیرد و نه مقداری بر می گرداند.

با استفاده از عملگرهای >> ، << و نام stream مورد نظر می توان برای دریافت و نمایش اطلاعات استفاده کرد.

M. Damrudi

file

```
#include <iostream.h>
#include<fstream.h>
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();
return 0;
}
```

M. Damrudi

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.

M. Damrudi

file

- 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);
```

M. Damrudi

file

```
#include <iostream.h>
#include<fstream.h>
int main(){
ofstream fileout1("test");
if (!fileout1){
   cout<<"can not open output file\n";
   return 1;
   }
fileout1<<"Hello!!!! \n This is a test ";
fileout1<<100;
fileout1.close();
ifstream filein1("test",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();
return 0;
}
```

M. Damrudi

file

برنامه ای بنویسید که اطلاعات نام خانوادگی و سن 5 کارمند را دریافت کرده در یک فایل ذخیره نمایید. سپس اطلاعات را از فایل خوانده و نمایش دهد.

M. Damrudi

```
#include <iostream.h>
#include<fstream.h>
struct karmand{
char family[20];
int age;
}kar[5];
int main(){
int i;
char familyk[20];
int agek;
ofstream fileout1;
fileout1.open("c:\\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("c:\\karmand",ios::binary);
if (!filein1){
cout<<"can not open \n";
return 1;
}
while(!filein1.eof())
 {
    filein1>>familyk>>agek;
    cout<<familyk<<agek;
 }
filein1.close();
return 0;
}
```

M. Damrudi