آموزش سریع C++

مولف: رضا سپاس یار (info@avr.ir)

رئوس مطالب:

- مبانی C++
- دستورات ورودی و خروجی
- تصمیم گیری و تکرار
- توابع
- نوع داده های ساخت یافته شامل:

آرایه ها، رشته ها، ساختارها، یونیون ها، کلاس ها

- اشاره گرها

مبانی C++

- در انتهای هر عبارت زبان C++ یک سمی کالن می آید.
- Brace ابتدا و انتهای یک تابع و همچنین یک بلوک را مشخص می کند.
- از " " برای مشخص کردن ابتدا و انتهای یک رشته ی متنی استفاده می شود.
- از // یا /* ... */ برای نوشتن توضیحات استفاده می شود.
- شناسه ها اسامی متغیرها، ثوابت و یا توابع هستند و نمی تواند از کلمات رزرو شده باشند و همچنین نمی توانند با یک کاراکتر عددی شروع شود و طول آن ها باید کمتر از ۳۱ کاراکتر باشد.
- C یک زبان Case Sensitive است و بین حروف کوچک و بزرگ تفاوت قائل می شود.
- کلمات رزرو شده حتما باید با حرف کوچک استفاده شوند. (مثل if, char, while,...)

متغیر ها و ثوابت

- تعریف متغیر یعنی انتخاب نام مستعار برای مکانی از حافظه
- فرم تعریف متغیر: نوع متغیر نام متغیر ; مثال: char a ;
- فرم تعریف ثابت: نوع ثابت نام ثابت = مقدار ; مثال: float pi=3.14;

انواع داده ها:

نوع	توضیح	اندازه	بازه
char	Character or small integer	1byte	signed: -128 to 127 unsigned: 0 to 255
short int	Short Integer	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int	Long integer	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value	1byte	true or false
float	Floating point number	4bytes	3.4e +/- 38 (7 digits)
double	Double precision floating point	8bytes	1.7e +/- 308 (15 digits)

اعمال ریاضی و محاسباتی

عملگرهای حسابی و بیتی

عملگر	عملکرد	مثال	نتیجه
*	ضرب	2*3	10
/	تقسیم	2/5	0.4
+	جمع	6+3	9
-	تفریق	8-3	5
%	باقیمانده	10%3	1
&	AND	0xF0 & 0x0F	0x00
|	OR	0x00 | 0x03	0x03
^	XOR	0x0F ^ 0xFF	0xF0
~	مکمل یک	~(0xF0)	0x0F
>>	شیفت به راست	0xF0 >> 4	0x0F
<<	شیفت به چپ	0x0F << 4	0xF0

عملگرهای یکانی

عملگر	عملکرد	مثال	نتیجه
-	قرینه	-a	a = a×-1
++	افزایش یک واحد	a++	a = a + 1
--	کاهش یک واحد	a--	a = a - 1

عملگرهای مقایسه ای

عملگر	عملکرد	مثال	نتیجه
>	بزرگتر	2>3	False
<	کوچکتر	'm'>'e'	True
>=	بزرگتر یا مساوی	5>=5	True
<=	کوچکتر یا مساوی	2.5<=4	False
==	تساوی	'A' =='B'	False
!=	نامساوی	2!=3	True

عملگرهای منطقی

عملگر	عملکرد	مثال	نتیجه
&&	AND منطقی	(2>3)&&(1!=3)	False
||	OR منطقی	('a'>3)||(5<2.5)	True
!	نقیض	!(7>5)	False

عملگرهای انتساب

عملگر	عملکرد	مثال	نتیجه
=	انتساب	a=b	a ⇦ b

a=a*b	a*=b	ضرب و انتساب	*=
a=a/b	a/=b	تقسیم و انتساب	/=
a=a+b	a+= b	جمع و انتساب	+=
a=a-b	a-=b	تفریق و انتساب	=
اگر value مقدار صحیح باشد a برابر x و در غیراینصورت برابر y می شود.		انتساب شرطی	a=(value)?x:y

---

دستورات ورودی و خروجی

برای دریافت اطلاعات از ورودی استاندارد که در PC صفحه کلید می باشد دستور **cin** و برای ارسال اطلاعات به خروجی استاندارد که نمایشگر می باشد دستور **cout** موجود می باشد.

دستور خروجی cout

```
cout << "Hello World";
```

دستور فوق عبارت This is a test را بر روی صفحه نمایش چاپ می کند.

```
cout << x;
```

دستور فوق محتویات متغیر x را به صفحه نمایش می فرستد.

علامت << عملگر insertion بوده و ممکن است بیش از یک بار در یک عبارت به کار رود، به عنوان مثال:

```
cout << "Hello" << "World!";
```

عملگر insertion امکان ترکیب رشته ها و متغیرها می دهد.

```
int age=25;
cout << "My age is:" << age;
```

برای انتقال به هر خط جدید از علامت \n یا endl استفاده می کنیم. به عنوان مثال دستورات زیر:

```
cout << "First Line.\n";
cout << "Second Line.\n Third Line.";
```

```
First Line.
Second Line.
Third Line.
```

```
cout << "First Line." << endl;
cout << "Second Line." << endl;
```

```
First Line.
Second Line.
```

دستور ورودی cin

دستور cin همراه علامت << به کار می رود.

```
int age;
cin >> age;
```

توسط یک دستور cin می توان بیش از یک متغیر را مقدار دهی کرد. به عنوان مثال دستورات زیر معادل یکدیگر می باشند:

```
cin >> a >> b;
```

اجرای برنامه ی Hello World در Visual C++ 6.0

۱. پس از اجرای برنامه File\New\Win32 Console Application را انتخاب کنید. (با فرض انتخاب نام و مسیر پروژه)

۲. از کادر حاصل An Empty Project را انتخاب کنید.

۳. مسیر File\New\C++ Source File را طی کرده و مطمئن شوید که Add to Project انتخاب شده است.

٤. در فایل با پسوند cpp کد زیر را وارد کنید:

```
#include <iostream.h>

int main()
{
      cout << "Hello World!"<< endl;
      return 0;
}
```

ابتدا F7 و سپس Ctrl+F5 را کلیک کنید، نتیجه به صورت زیر خواهد بود:

برنامه ی زیر محیط و مساحت یک دایره را حساب می کند.

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

```
int main()
{
      int r;
      const float pi=3.1459;

      cout <<"Enter Radius: ";
      cin >>r;
      cout <<"If radius="<<r<<" Circumference is:
"<<2*pi*r<<endl;
      cout <<"If radius="<<r<<" Area is: "<<pi*r*r<<endl;

      return 0;
}
```

تصمیم گیری و تکرار

دستور goto

پرش بدون شرط به یک برچسب انجام می شود. به عنوان مثال برنامه ی زیر عدد ۱۰ تا ۰ صفر و در نهایت پیام End! را نمایش می دهد.

```
#include <iostream.h>

int main()
{

        int n=11;

loop1:          n--;
        cout<<n<<endl;
        if(n!=0)
            goto loop1;

        cout << "End!"<<endl;
```

```
        return 0;

}
```

ساختار if - else:

```
if(شرط)
{
      دستورات۱;
}
else
{
      دستورات ۲;
}
```

- ✓ در صورتی که دستورات یک خطی باشند می توان brace را حذف نمود.
- ✓ استفاده از بلاک else اختیاری است.

به عنوان مثال برنامه ی زیر ریشه های معادله ی درجه دوم را حساب می کند:

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

int main()
```

```
{
    float a,b,c,delta;

    cout <<"Enter a, b and c: "<<endl;
    cin >>a>>b>>c;
    delta=b*b-4*a*c;
    if(delta>=0)
    {
        cout <<"X1="<<(-b+sqrt(delta))/(2*a)<<endl;
        cout <<"X2="<<(-b-sqrt(delta))/(2*a)<<endl;
    }
    else
        cout<<"Roots are not real!"<<endl;

    return 0;
}
```

ساختار Switch - Case:

```
switch(مقدار)
{
    case  مقدار۱:
    دستورات۱;
    break;
    case  مقدار۲:
    دستورات۲;
    break;
    .
```

```
    .
    default:
    nدستورات;
}
```

به عنوان مثال برنامه ی زیر یک عدد را از کاربر گرفته و بر اساس تابع انتخاب شده نتیجه را محاسبه می کند:

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

int main()
{
    float num;
    int operation;

    cout <<"Enter a number"<<endl;
    cin >>num;
    cout<<"Enter 1, 2, 3 or 4 to seleting sin(), cos(),
tan() or Log()"<<endl;
    cin>>operation;

    switch(operation)
    {
    case 1:
        cout<<"sin of "<<num<<" is "<<sin(num)<<endl;
        break;
    case 2:
```

```
        cout<<"cos of "<<num<<" is "<<cos(num)<<endl;
        break;
    case 3:
        cout<<"tan of "<<num<<" is "<<tan(num)<<endl;
        break;
    case 4:
        cout<<"log of "<<num<<" is "<<log(num)<<endl;
        break;
    default:
        cout<<"Enter 1, 2, 3 or 4!"<<endl;
    }

    return 0;
}
```

ساختار While:

```
while(شرط)
{
    دستورات;
}
```

برنامه ی زیر برای خروج، از کاربر درخواست کاراکتر e را می کند.

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

```
int main()
{

    char c=0;

    while(c!='e')
    {
        cout<<"Enter e to exit"<<endl;
        cin>>c;
    }

    return 0;
}
```

ساختار Do/While:

در این حالت بر خلاف While شرط در انتهای حلقه آزمایش می شود، بنابراین دستورات داخل حلقه، حداقل یکبار اجرا می شوند.

```
do{
    دستورات;
} while(شرط)
```

حلقه های For:

```
for(مقدار ابتدای حلقه ; شرط پایان ; گام)
    {
    دستورات;
    }
```

برنامه ی زیر یک عدد را از کاربر گرفته و فاکتوریل آن را حساب می کند.

```
#include <iostream.h>

int main()
{

    int i,n;
    double fact=1;

    cout<<"Enter a Number: ";
    cin>>n;

    for(i=1;i<=n;i++)
        fact*=i;
```

```
    cout<<"Factorial of "<<n<<" is "<<fact<<endl;


    return 0;
}
```

توابع

تعریف توابع به صورت زیر می باشد:

نوع داده خروجی نام تابع (آرگومانهای تابع)

{

متغیرهای محلی

دستورات تابع

}

مثال: برنامه ای بنویسید که یک عدد را دریافت کرده و مکعب آن را چاپ کند.

```
#include <iostream.h>

long int cube(int x){
      return x*x*x;
```

```
}

void main(void){
      int a;
      cout<<"Enter a Number: ";
      cin>>a;
      cout<<"Cube of "<<a<<" is: "<<cube(a)<<endl;
}
```

⇦ توابع داخل یکدیگر قابل تعریف نمی باشند و جدا از هم باید تعریف گردند.

مثال: برنامه ای بنویسید که ماکزیمم دو عدد دریافت شده را چاپ کند.

```
#include<iostream.h>

int _max(int a,int b){
      if(a>b)
            return a;
      else
            return b;
}

void main(void){
```

```
      int a,b;
      cout<<" Enter Tow Numbers: "<<endl;
      cin>>a>>b;
      cout<<"Maximum of "<<a<<" & "<<b<<" is
"<<_max(a,b)<<endl;
}
```

می توان تابع را بعد از تابع main() نیز معرفی نمود، با این تفاوت که شناسه ی تابع باید در ابتدای برنامه تعریف شود. به عنوان مثال برنامه ی قبل را به صورت زیر نیز می توان نوشت:

```
#include<iostream.h>

int _max(int a,int b);

void main(void){
      int a,b;
      cout<<" Enter Tow Numbers: "<<endl;
      cin>>a>>b;
      cout<<"Maximum of "<<a<<" & "<<b<<" is
"<<_max(a,b)<<endl;
}

int _max(int a,int b){
      if(a>b)
            return a;
      else
            return b;
```

```
}
```

⇦ با تعریف توابع در یک کتابخانه ی مجزا و اعلان آن در یک فایل Header می توان به انسجام برنامه کمک نمود.

حوزه ی تعریف متغیر

قسمتی از برنامه که متغیر در آن تعریف شده را حوزه آن متغیر می گویند. در C++ به قسمتی از برنامه که با یک جفت Brace محصور شده است یک بلاک می گویند. یک متغیر در همان بلاک تعریف شده قابل دسترسی بوده و در در خارج از بلاک قابل استفاده نمی باشد.

به عنوان مثال در برنامه زیر پیغام undeclared identifier را دریافت خواهیم کرد.

```
#include <iostream.h>

void main( )
{
    {
      int x= 1;
    }
    x=x+1;
}
```

متغیری که خارج از همه توابع تعریف می شود برای تمام توابع، شناخته شده و قابل استفاده می باشد. به عنوان مثال در برنامه ی زیر متغیر **X** در چنین وضعیتی می باشد.

```
#include <iostream.h>

int x=5;
int test();

void main(){
   cout<<x<<endl;
   cout<<test()<<endl;
}

int test(){
  return x;
}
```

اگر بخواهیم یک متغیر محلی تابع، مقدار خود را برای فراخوانی های بعدی نیز حفظ کند، C++ کلمه ی کلیدی static را در اختیار ما قرار می دهد.

```
static int x=۵;
```

به عنوان مثال اگر برنامه ی زیر را بدون static اجرا کنیم خروجی همواره عدد ۱۰ خواهد بود.

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

```
int test();

void main()
{
   cout<<test()<<endl;
   cout<<test()<<endl;
}

int test(){
  static int x=10;
  x = x*2;
  return x;
}
```

آرگومانهای پیش فرض توابع

می توان برای توابع آرگومان هایی را به عنوان پیش فرض تعریف نمود. هنگامی که در فراخوانی توابع، آرگومان دارای مقدار پیش فرض حذف شده باشد، مقدار پیش فرض آن آرگومان، به تابع ارسال خواهد شد. به عنوان مثال:

```
#include <iostream.h>
double volume(double r=1, double h=1);
double pi=3.145;

void main(){

  cout<<"The default volume is: "<<volume()<<endl;
```

```
  cout<<"The volume with r=10: "<<volume(10)<<endl;

  cout<<"The volume with r=10 h=5 is:
"<<volume(10,5)<<endl;

}

double volume(double r, double h){
   return pi*r*r*h;
}
```

عملگر تفکیک حوزه (Scope Resolution Operator)

همانطور که می دانیم، می توان متغیر محلی همنام با متغیر عمومی در محدوده ی یک تابع تعریف نمود. با توجه به این مسئله سوالی که مطرح می شود این است که در محدوده ی یک تابع برای دسترسی به متغیر همنام عمومی چه راهکاری وجود دارد؟ به این منظور C++ عملگر یگانی تفکیک دامنه (::) را در اختیار ما قرار داده است. برنامه زیر نحوه کاربرد عملگر (::) را نشان می دهد.

```
#include <iostream.h>

int amount = 123;                    // global variable

void main()
{
```

```
    int amount = 456;             // local variable
    cout << ::amount << endl;     // Print the global
variable (123)
    cout << amount << endl;       // Print the local
variable (456)
}
```

Overload کردن توابع

در C++ این امکان وجود دارد که چند تابع با نام یکسان تعریف نمود که به این عمل Overloading گفته می شود. آنچه توابع را از یکدیگر متمایز می سازد نوع داده ی آرگون های ارسالی به آن ها است.

```
#include <iostream.h>

int abs(char i);
int abs(int d);
int abs(long l);

int main(){

     cout << abs(-10) << endl;
     cout << abs(-11) << endl;
     cout << abs(-12) << endl;
     return 0;
```

```
}

int abs(char i){
      return i<0 ? -i : i;
}

int abs(int d){
      return d<0 ? -d : d;
}

int abs(long l){
      return l<0 ? -l : l;
}
```

فراخوانی تابع با ارجاع

بر خلاف روشی که تاکنون برای ارسال آرگومتن به تابع استفاده کردیم این امکان وجود دارد که مقدار ارسال شده به تابع توسط آن تغییر کند. بع عبارت دیگر به جای آنکه یک کپی از متغیر به تابع ارسال شود آدرس آن متغیر ارسال شده و خود متغیر مورد استفاده ی تابع قرار می گیرد. به عنوان مثال:

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

```
void inc(int &x,int &y);

void main()
{
    int a=1 ,b=2;
    cout<<"a = "<<a<<" and b = "<<b<<endl;
    inc(a,b);
    cout<<"a = "<<a<<" and b = "<<b<<endl;
}

void inc(int &x , int &y)
{
    x++;
    y++;
}
```

⇦ همانطور که در این برنامه ملاحظه می شود برخلاف روش قبل تابع می تواند بیش از یک مقدار بازگشتی داشته باشد.

نوع داده های ساخت یافته

۱. آرایه ها

اعلان آرایه ها

نوع داده نام آرایه [طول] ;

عناصر آرایه را می توان به صورت زیر مقدار دهی نمود. در صورتی که طول آرایه نوشته نشود، کامپایلر آن را به تعداد عناصر تخصیص داده شده در نظر می گیرد.

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

مثال: برنامه ای که ۱۰ عدد را از کاربر گرفته و برای آن ها نمودار ستونی رسم می کند.

```
#include <iostream.h>

void main()
{
  int num[10];
```

```
  int i,j;


  for(i=0;i<10;i++){
        cout<<"Enter Number "<<i+1<<": ";
        cin>>num[i];
  }
  cout<<endl<<"Index"<<"  Number"<<endl;


  for(i=0; i<10;i++){
     cout<<i+1<<"\t "<<num[i]<<"\t";
     for(j=0; j<num[i]; j++)
        cout<<'|';
           cout<<endl;
  }
}
```

رشته ها

در زبان C رشته ها با آرایه ای از کاراکترها پیاده سازی می شوند.

```
char str[]="Test";
```

طول آرایه فوق برابر است با طول کلمه Test بعلاوه یک واحد که مربوط به کاراکتر Null یا \0 است می باشد، بنابراین می توان آن را به صورت زیر نیز تعریف کرد.

```
char str[]={'T','e','s','t',\0'}
```

با دستور cin می توان مقادیر رشته ای را از ورودی خواند.

```
#include <iostream.h>

char str[10];

void main(){
      cout<<"Enter Your Name: ";
      cin >> str;
      cout<<"Hi "<<str<<" !"<<endl;
}
```

به تک تک کاراکترهای یک رشته همانند خانه های یک آرایه می توان دسترسی داشت:

```
#include <iostream.h>

char str[10];
int i;

void main(){
      cout<<"Enter Your Name: ";
      cin >> str;

      for(i=0;str[i]!='\0';i++)
```

```
        cout<<str[i]<<endl;
}
```

۲. ساختارها (رکوردها)

ساختارها دومین نوع داده ی ساخت یافته بوده و عناصر یک ساختار می توانند از نوع های داده ی مختلف باشند. مثال زیر نحوه ی تعریف یک ساختار را نشان می دهد.

```
struct studentRec
{
        char studebtName[16];
        float midTermGrade;
        float homeworkGrade;
        float finalGrade;
        float aveGrade;
};
```

در دستور فوق کلمه struct یک ساختار را با نام studentRec ایجاد می کند ساختار ایجاد شده دارای ۵ عنصر می باشد که یکی از آن ها از نوع رشته و بقیه اعشاری می باشند.

اعلان یک ساختار در واقع تعریف یک نوع داده ی جدید است و بنابراین پس از ایجاد ساختار می توانیم از این نوع داده جدید استفاده کنیم و متغیرهایی را از این نوع تعریف کنیم.

```
studentRec firstStudent;
studentRec Student[5];
```

دسترسی به اعضای ساختار

برای دستیابی به عناصر یک ساختار از عملگر نقطه (.) استفاده می کنیم. عملگر نقطه برای دستیابی به عنصر یک ساختار از طریق نام یک متغیر، مورد استفاده قرار می گیرد، به عنوان مثال:

```
firstStudent.finalGrade = 17;
```

مثال: برنامه ی زیر نحوه ی تعریف و مقدار دهی یک رکوردها را نشان می دهد.

```
#include <iostream.h>

struct studentRec
{
      char studebtName[16];
      float midTermGrade;
      float homeworkGrade;
      float finalGrade;
      float aveGrade;
};

studentRec firstStudent;

void main(){

      cout<<"Enter Student Name: ";
```

```
      cin>>firstStudent.studebtName;

      cout<<endl<<"Enter Mid Term Grade: ";
      cin>>firstStudent.midTermGrade;

      cout<<endl<<"Enter Homework Grade: ";
      cin>>firstStudent.homeworkGrade;

      cout<<endl<<"Enter Final Grade: ";
      cin>>firstStudent.finalGrade;

      firstStudent.aveGrade = (firstStudent.midTermGrade
+firstStudent.homeworkGrade+firstStudent.finalGrade)/3;

      cout<<endl<<"The Average Grade of
"<<firstStudent.studebtName<<" is
"<<firstStudent.aveGrade<<endl;

}
```

می توان با تعریف آرایه ای از نوع ساختار مورد نظر تعداد زیادی رکورد را پردازش نمود:

```
#include <iostream.h>

struct studentRec
```

```
{
      char studebtName[16];
      float midTermGrade;
      float homeworkGrade;
      float finalGrade;
      float aveGrade;
}


studentRec Student[5];


void main(){
      int i;
      for(i=0;i<5;i++){

            cout<<"Enter Student Name "<<i+1<<": ";
            cin>>Student[i].studebtName;

            cout<<endl<<"Enter Mid Term Grade: ";
            cin>>Student[i].midTermGrade;

            cout<<endl<<"Enter Homework Grade: ";
            cin>>Student[i].homeworkGrade;

            cout<<endl<<"Enter Final Grade: ";
            cin>>Student[i].finalGrade;
```

```
            cout<<"-------------------------"<<endl;
      }

      for(i=0;i<5;i++)
            Student[i].aveGrade = (Student[i].midTermGrade
 +

      Student[i].homeworkGrade+Student[i].finalGrade)/3;

      cout<<"Name\t"<<"Grade"<<endl;

      for(i=0;i<5;i++){

      cout<<Student[i].studebtName<<"\t"<<Student[i].aveG
rade<<endl;
      }

      cout<<"-------------------------"<<endl;
}
```

۳. Union ها

سومین نوع داده ی ساخت یافته Union می باشد. این نوع داده شبیه ساختار بوده، با این تفاوت که در زمان تعریف یک شناسه از این نوع، برخلاف ساختارها فضایی برای تمام اعضا ساختار در نظر گرفته نمی شود و فقط در زمان استفاده از آن متغیر فضای مربوطه در حافظه در تعریف می شود. تعریف Union همانند ساختار می باشد.

```
union studentRec
{
      char studebtName[16];
      float midTermGrade;
      float homeworkGrade;
      float finalGrade;
      float aveGrade;
};
```

همانند یک ساختار برای دستیابی به عناصر یک Union از عملگر نقطه (.) استفاده می کنیم. با این تفاوت که پس از مقدار دهی هر عنصر Union، مقدار قبلی از بین می رود.

```
studentRec.homeworkGrade = 15;
studentRec.finalGrade = 18;
studentRec.aveGrade = 17;
```

٤. کلاس ها

تعریف کلاس

در++C بوسیله ی کلاس ها امکان تعریف نوع داده ی مجرد بوجود می آید. کلاسها همانند ساختارها تعریف می شوند با این تفاوت که در کلاسها، توابع نیز می توانند به عنوان عضو کلاس باشند. یک کلاس توسط کلمه ی کلیدی class تعریف می شود.

```
class TimeType{

  public:
    void setTime(int,int,int);
    void Increment();
    void Decrement();

  private:
    int hour;
    int minute;
    int second;
}
```

هر عضو داده ای یا تابعی که بعد از private تعریف می شود تنها برای توابع عضو کلاس قابل دستیابی می باشد و به طور مستقیم از داخل برنامه قابل دسترسی نمی باشند. تابع عضوی که همنام با کلاس وجود دارد تابع Constructor نامیده می شود. تابع سازنده، تابع عضو بخصوصی است که مقادیر اولیه اعضای داده ای را تعیین می کند. هنگامی که برنامه شی ای را از کلاس ایجاد می کند، تابع سازنده به طور خودکار فراخوانی می گردد، پارامترها را مقداردهی اولیه کرده و مقداری را بر نمی گرداند.

همانند ساختارها ، هنگامی که کلاس TypeTime تعریف گردید می توان اشیایی را از نوع کلاس TypeTimeایجاد کرد.

```
TimeType GlobalTime;
TimeType TimeArray[5];
```

مثال:

```
#include <iostream.h>

class Cube
{
public:
      Cube();
      void setSide(double s);
      double getSide();
      double Area();
      double Volume();
      void Properties();
private:
      double Side;
};
```

```
void main()
{
      int side;

      cout<<" Enter side of cube: ";
      cin>>side;
      cout<<endl;

      Cube MyCube;

      MyCube.setSide(side);
      MyCube.Properties();

}

//-----------------------------------

Cube::Cube()
{
      Side=0;
}

void Cube::setSide(double s)
{
```

```
      Side = s <= 0 ? 1 : s;
}


double Cube::getSide()
{
      return Side;
}


double Cube::Area()
{
      return 6 * Side * Side;
}


double Cube::Volume()
{
      return Side * Side * Side;
}


void Cube::Properties()
{
      cout << "Characteristics of this cube";
      cout << "\nSide   = " << getSide();
      cout << "\nArea   = " << Area();
      cout << "\nVolume = " << Volume() <<endl<<endl;
}
```

برای انسجام برنامه و کپسوله کردن یک کلاس می توانیم آن را به صورت مجزا در فایل دیگری تعریف نموده و الگوی کلاس را در فایل Header تعریف نمود.

فایل اصلی با پسوند C .

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

void main()
{
    int side;

    cout<<" Enter side of cube: ";
    cin>>side;
    cout<<endl;

    Cube MyCube;

    MyCube.setSide(side);
    MyCube.Properties();
}
```

تعریف کلاس در فایل cube.c

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

Cube::Cube()
{
    Side=0;
}

void Cube::setSide(double s)
{
    Side = s <= 0 ? 1 : s;
}

double Cube::getSide()
{
    return Side;
}

double Cube::Area()
{
    return 6 * Side * Side;
}
```

```
double Cube::Volume()
{
    return Side * Side * Side;
}


void Cube::Properties()
{


    cout << "Characteristics of this cube";
    cout << "\nSide   = " << getSide();
    cout << "\nArea   = " << Area();
    cout << "\nVolume = " << Volume() <<endl<<endl;
}
```

اعلان کلاس در فایل cube.h

```
class Cube
{
public:
    Cube();
    void setSide(double s);
    double getSide();
    double Area();
    double Volume();
    void Properties();
private:
```

```
    double Side;
};
```

حوزه کلاس و دسترسی به اعضای کلاس

داده های عضو یک کلاس (متغیرهایی که در کلاس تعریف شده اند) و توابع عضو کلاس (توابعی که در کلاس تعریف شده اند) به حوزه این کلاس متعلق می باشند و توابعی که عضو کلاس نمی باشند، دارای حوزه ی فایل می باشند. در حوزه یک کلاس، همه اعضای کلاس توسط توابع عضو کلاس قابل دسترسی می باشند و می توان توسط نامشان مستقیما به آنها مراجعه کرد. در بیرون از حوزه کلاس برای دسترسی به اعضای کلاس از عملگر نقطه (.) استفاده می کنیم. متغیرهایی که در یک تابع عضو کلاس تعریف شده اند، دارای حوزه تابع می باشند، یعنی فقط برای همان تابع شناخته شده اند. اگر در تابع عضوی، متغیری همنام با نام متغیری که دارای حوزه کلاس است تعریف شده باشد، در این صورت متغیری که دارای حوزه کلاس می باشد در داخل تابع به طور مستقیم نمی توان رجوع کرد و برای دسترسی به آن باید عملگر Scope Resolution (::) را قبل از نام متغیر قرار داد.

کنترل دسترسی به اعضا

معرفهای دسترسی به اعضا یعنی public و private، نحوه دسترسی به اعضای داده و توابع عضو کلاس را کنترل می کنند. حالت پیش فرض برای دسترسی به اعضای داده و توابع عضو در کلاس private می باشد، لذا همه اعضایی که بعد از تعیین نام کلاس و قبل از اولین برچسب معرف آمده اند، خصوصی در نظر گرفته می شوند. پس از هر معرف، حالتی که معرف مربوطه تعیین می کند، به همه اعضا تا برچسب بعدی یا { مربوط به

پایان تعریف کلاس اعمال می شود. اعضای خصوصی یک کلاس فقط از طریق توابع عضو آن کلاس قابل دسترسی می باشند در مقابل اعضای عمومی کلاس توسط هر تابعی در برنامه قابل دسترسی می باشند.

ارث بری کلاسها (Inheritance)

کلاس اصلی را base class و کلاس مشتق شده را derived class می گویند. نحوه ی تعریف یک کلاس ارث بری شده به صورت زیر می باشد:

```
class derived-class-name : access base-class-name
 {
    // body of class
};
```

به عنوان مثال:

```
#include <iostream.h>

class BaseClass {
  int i;
public:
  void set(int n);
  int get();
};
```

```
class DerivedClass : public BaseClass {
  int j;
public:
  void setJ(int n);
  int mul();
};

void BaseClass::set(int n)
{
  i = n;
}

int BaseClass::get()
{
  return i;
}

void DerivedClass::setJ(int n)
{
  j = n;
}

int DerivedClass::mul()
{
  return j * get();
}

int main()
{
  DerivedClass ob;
```

```
  ob.set(10);          // load i in BaseClass
  ob.setJ(4);            // load j in DerivedClass

  cout << ob.mul()<<endl;      // displays 40

  return 0;
}
```

اشاره گرها (Pointers)

تعریف اشاره گر

نوع داده ی اشاره گر، آدرس خانه های حافظه را در خود نگهداری می کند. نام یک متغیر به طور مستقیم به یک مقدار، مراجعه می کند اما یک اشاره گر به طور غیر مستقیم به یک مقدار مراجعه می کند. به عنوان مثال دستور زیر متغیر a را از نوع int و متغیر p را اشاره گری از نوع int تعریف می کند.

```
int a , *p ;
```

عملگر آدرس (&)

این عملگر آدرس خانه حافظه عملوند خود را بر می گرداند. به عنوان مثال:

```
#include<iostream.h>

void main()
{
      int a=10;
      int *p;
```

```
        p=&a;
        cout<<p<<endl;
        cout<<*p<<endl;
}
```

ارسال آرگومان به تابع توسط اشاره گر

اشاره گرها مانند آرگومانهای ارجاع می توانند برای تغییر یک یا چند متغیر ارسال شده از داخل تابع و یا برای ارسال داده های بزرگ به توابع مورد استفاده قرار گیرند. به عنوان مثال:

```
#include<iostream.h>

void swap(int *x, int *y);

int main(void)
{
        int i, j;
        i = 10;
        j = 20;

        swap(&i, &j); /* pass the addresses of i and j */

        cout<<"i is: "<<i<<endl;
        cout<<"j is: "<<j<<endl;
```

```
        return 0;
}

void swap(int *x, int *y)
{
        int temp;
        temp = *x; /* save the value at address x */
        *x = *y; /* put y into x */
        *y = temp; /* put x into y */
}
```

مثال: همانطور که در بخش توابع بررسی شد C++ امکان ارسال آرگومان با ارجاع را بدون استفاده از اشاره گر ها می دهد. بنابراین ازسال آرگومان با ارجاع به دو روش امکان پذیر می باشد. برای مقایسه ی این دو روش به دو برنامه ی زیر توجه کنید:

```
#include <iostream.h>

void neg(int &i); // i now a reference
int main()
{
        int x;
        x = 10;
        cout << x << " negated is ";
        neg(x); // no longer need the & operator
        cout << x << "\n";
```

```
	return 0;
}


void neg(int &i)
{
	i = -i; // i is now a reference, don't need *
}



#include <iostream.h>

void neg(int *i);

int main()
{
	int x;
	x = 10;
	cout << x << " negated is ";
	neg(&x);
	cout << x << "\n";
	return 0;
}

void neg(int *i)
{
	*i = -*i;
}
```