تكملة موضوع

Classes and Objects

4.15 CONSTRUCTORS AND DESTRUCTORS

INTRODCTION

C++ provides a special member function called the *constructor* which enables an object to initialize itself when it is created. This is known as *automatic initialization* of objects. It also provides another member function called the *destructor* that destroys the objects when they are no longer required.

4.15.1 Constructors

A constructor is a 'special' member function whose task is to initialize the objects of its class. **A constructor is a member function that is executed automatically whenever an object is created.** It is special because its name is the same as the class name. The constructor is invoked whenever an object of its associated class is created. It is called constructor because it construct the values of data members of the class.

A constructor is declared and defined as follows:

```
// class with a constructor
  class integer
 {
        int   m , n;
    public:
       integer ( void );      // constructor declared
       …….
       ……
 };
  integer :: integer (void ) // constructor defined
 {
      m=0; n=0;
 }
```

when a class contains a constructor like the one defined above , it is guaranteed that an object created by the class will be initialized automatically. For example , the declaration

```
integer int1;        //object int1 created
```

not only creates the object **int1** of type integer but also **initializes** its data members **m** and **n** to zero.

A constructor that accepts no parameters is called the ***default constructor***. The default constructor for class **A** is **A::A( ).** If no such constructor is defined , then the compiler supplies a default constructor. Therefore a statement such as (**A a;**)

Invokes the default constructor of the compiler to create the object **a**. The constructor functions **have some special characteristics:**

- They should be declared in the public section.
- They are invoked automatically when the objects are created .
- They do not have return types, not even void and therefore, they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ functions, they can have default arguments.
- We can be defined as **inline** function

- دالة البناء:-
- عند تشغيل البرنامج من غير أي استدعاء هي عبارة عن دالة تنفذ تلقائي
- **أو** هي عبارة عن دالة يتم استدعاؤها مباشرة عند اشتقاق كائن من صنف معين

4.15.2 Parameterized Constructors

the constructor **integer ( ),** defined above, initializes the data members of all the objects to zero. However, in practice it may be necessary the various data elements of different objects with different values when they are created. C++ permits us to achieve this objective by passing arguments to the constructor function when the objects are created.

```
class integer
{
        int   m , n;
   public:
      integer ( int x , int y );      // parameterized constructor
      .......
      ......
};
 integer :: integer (int x , int y ) // constructor defined
{
     m = x;  n = y;
}
```

in above program we must pass the initial values as arguments to the constructor function when an object is declared. This can be done in two ways:

- By calling the constructor **explicitly**.
- By calling the constructor **implicitly**.

The following declaration illustrates the first method:

```
integer int1= integer(0,150);    //explicit call
```

This statement creates an **integer** object **int1** and passes the values 0 and 150 to it.
The second is implemented as follows:

integer int1(0,150); //implicit call (*shorthand*)

```
////////////////////////// Class with Constructors//////////////////////////
#include <iostream>
using namespace std;
class integer
{          int   m , n;
    public:
        integer ( int , int );              //constructor declared
        void display (void);
    };
integer :: integer (int x , int y )        // constructor defined
                  {       m = x;  n = y;       }

void  integer :: display (void)
        {      cout << " m = "  << m  << "\n";
               cout << " n = "  << n  << "\n";
         }
int main ()
{
          integer int1(0,100);                 //implicit call
          integer int2= integer(25,75);    //explicit call
          cout << "\n OBJECT1"  << "\n";
          int1.display( );
          cout << "\n OBJECT2"  << "\n";
          int2.display( );
return 0;
}
```

The output of above program is:

```
OBJECT1
m = 0
n= 100
OBJECT2
m = 25
n= 75
```

EXAMPLE

```
#include <iostream>
```

```
using namespace std;

class operations
{
float a,b,c; int ch;
public:
operations( );
void result( );
};

operations ::operations( )
{
cout<<"Mathematical Operations \n";
cout<<" 1- Addition \n";
cout<<" 2- Subtraction \n";
cout<<" 3- Multiplication \n";
cout<<" 4- Division \n";
 cout<<" Please Enter your choice : \n";
 cin>>ch;
cout<<" Please Enter two Values a and b \n";
cin>>a>>b;
}
void operations::result( )
{
switch (ch)
{
case 1: c=a+b; cout<<a<<"+"<<b<<"="<<c<<endl;break;
case 2 :c=a-b;cout<<a<<"-"<<b<<"="<<c<<endl; break;
case 3: c=a*b;cout<<a<<"*"<<b<<"="<<c<<endl; break;
case 4: if(b!=0){ c=a/b;cout<<a<<"/"<<b<<"="<<c<<endl;}
       else   cout<<"the result of division is infinite ";
defult :cout<< "error choice";
}   }

int main( )
{     int key;
do
{    operations op;
op.result( );
 cout <<"to terminate program write key= 0 otherwise enter any value to key \n";
 cin>>key ;     }while (key !=0);
return 0; }
```

4.15.3 Destructors

A destructor, as name implies, is used to destroy the objects that have been created by a constructor. Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde (~). For example, the destructor for the class **integer** can be defined as shown below:

~integer() { }

A destructor never takes any argument nor does it return any value. It will be invoked implicitly by the compiler upon exit from the program (or block or function as the case may be) to clean up storage that is no longer accessible.

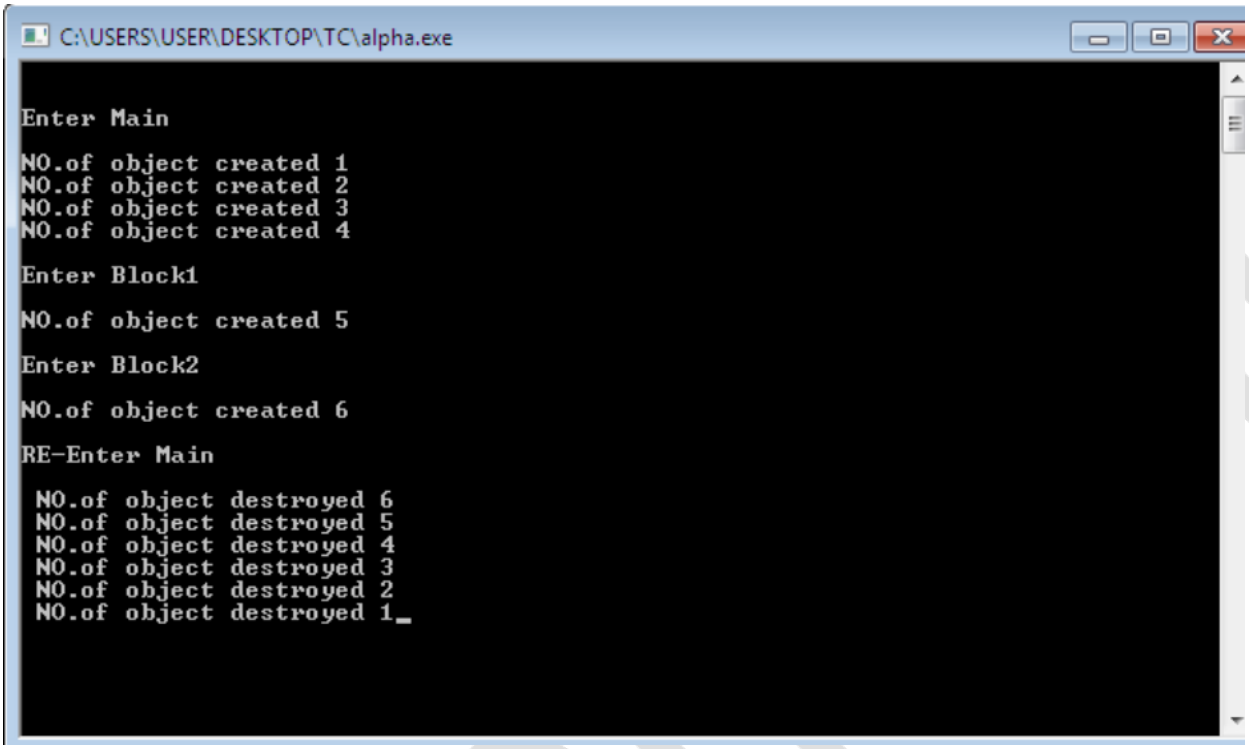
دالة الهدم-:

هي عبارة عن دالة يتم استدعاؤها مباشرة عند اشتقاق كائن من صنف ولكن عند نهاية البرنامج

```
////////////////// implementation of destructors//////////////////
#include <iostream>
using namespace std;
int count=0;
class alpha
{
    public:
        alpha()
        {
            count++;
            cout<< "\nNO.of object created " << count ;
        }
        ~alpha()
        {
            cout<< "\nNO.of object destroyed " << count ;
            count--;
        }
};

int main()
{
  cout<< "\n\nEnter Main\n;
  alpha A1,A2,A3,A4;
  {
        cout << "\n\nEnter Block1\n";
        alpha A5;
  }
```

```
   {
        cout << "\n\nEnter Block2\n";
       alpha A6;
    }
   cout<< "\n\nRE-Enter Main\n;  return 0;
}
```



الحل بطريقة ثانية

```
#include <iostream>
#include <conio.h>
using namespace std;
int count=0;
class alpha
{
    public:
        alpha()
        {
            count++;
            cout<< "\nNO.of object created " << count ;
         }
        ~alpha()
         {
            cout<< "\n NO.of object destroyed " << count ;
```

```
            count--;
        }
};
int main()
{
 { cout<< "\n\nEnter Main\n";
  alpha A1,A2,A3,A4;
        cout << "\n\nEnter Block1\n";
        alpha A5;
       cout << "\n\nEnter Block2\n";
      alpha A6;
       cout<< "\n\nRE-Enter Main\n";
  }
  getch(); return 0;
}
```

The output of a sample run of Program 6.7 is shown below:

```
ENTER MAIN

No.of object created 1
No.of object created 2
No.of object created 3
No.of object created 4

ENTER BLOCK1

No.of object created 5
No.of object destroyed 5

ENTER BLOCK2

No.of object created 5
No.of object destroyed 5

RE-ENTER MAIN

No.of object destroyed 4
No.of object destroyed 3
No.of object destroyed 2
No.of object destroyed 1
```

خواص دوال البناء والهدم:-

1. تحمل نفس اسم الانف ولكن دالة الهدم تسبق بعلامة ~
يتم تعريفهم في مستوى الحماية العام **public**
3. يمكن إنشاء أكثر من دالة بناء
4. يمكن إنشاء دالة هدم واحدة فقط
5. ليس لهما أنواع رجوع

4.15.4 Multiple Constructors In A Class

so far we have used two kinds of constructors. They are:

```
integer();              //No arguments
integer(int , int);     //two arguments
```

in the first case, the constructor itself supplies the data values and no values are passed by the calling program. In second case, the function call passes the appropriate value from **main ()**. C++ permits us to use both these constructors in the same class. For example, we could define a class as follows:

```
class integer
{
      int m,n;
  public:
     integer()  { m=0; n=0; }    //constructor 1
     integer(int a , int b )
     {m = a ; n = b;}              //constructor 2
     integer(integer &i)
     {m =  i . m ; n = i . n;}     //constructor 3
};
```

this declares three constructors for an **integer** object. The first constructor receives no arguments, the second, receivers two integer arguments, and the third receives one **integer** object as an argument. For example, the declaration:

```
integer I1;
```

Would automatically invoke the first constructor and the set both **m** and **n** of **I1** to zero. The statement

```
integer I2 (20 , 40);
```

Would call the second constructor, which will initialize the data members **m** and **n** of **I2** to 20 and 40 respectively. Finally, the statement

```
integer I3 (I2);
```

would invoke the third construct which copies the value of **I2** into **I3**. That is, it sets the value of every data element of **I3** to the value of corresponding data element of **I2.** Such a constructor is called the *copy constructor*.

When more than one constructor function is defined in a class, we say that the constructor is overloaded.

////////////////////////////// Overloading Constructor ////////////////////////////////

Exercises

1. Create a class that imitates part of the functionality of the basic data type int. Call the class Int (note different capitalization). The only data in this class is an int variable. Include member functions to initialize an Int to 0, to initialize it to an int value, to display it

(it looks just like an int), and to add two Int values.
Write a program that exercises this class by creating one uninitialized and two initialized Int values, adding the two initialized values and placing the response in the uninitialized value, and then displaying this result.(Instead of having z=x+y, and x,y and z are int , we could have z.add(x,y) and x,y and z are of type Int.)
Solutions to Exercises
1.

```
// ex6_1.cpp
// uses a class to model an integer data type
#include <iostream.h>
class Int //(not the same as int)
{
private:
int i;
public:
Int() //create an Int
{ i = 0; }
Int(int X) //create and initialize an Int
{ i = X; }
void add(Int i2, Int i3) //add two Ints
{ i = i2.i + i3.i; }
void display() //display an Int
{ cout << i; }
};

void  main()
{ Int Int1(7);                    //create and initialize an Int
Int Int2(11);                   //create and initialize an Int
Int Int3;                       //create an Int
Int3.add(Int1, Int2);             //add two Ints
cout << "\nInt3 = "; Int3.display(); cout << endl;  }
```

4.15.5 copy constructor

a copy constructor is used to declare and initialize an object from another object. For example, the statement

integer L2(L1);

would define the object **L2** and at the same time initialize it to the value of **L1**. Another form of this statement is

integer L2=L1;

//////////////////////////////// copy constructor ////////////////////////////

```
#include <iostream.h>
class code
{
      int id;
  public:
       code()  {}                    // constructor
       code (int a) {id = a;}         //constructor again
       code (code & x)                //copy constructor
        {
          id = x . id;
        }
       void display(void)
       {    cout  <<  id;   }
};
main()
{
    code A(100);
    code B(A);
    code C=A;
    code D;
    D=A;
    cout<<"\n id of A: "; A.display();
    cout<<"\n id of B: "; B.display();
    cout<<"\n id of C: "; C.display();
    cout<<"\n id of D: "; D.display();
}
////////////////////program ///////////////////////
```

ecopycon.cpp

```
// initialize objects using default copy constructor
#include <iostream>
using namespace std;
class  Distance    //English Distance class
{
private:
int feet; float inches;
public:
Distance() : feet(0), inches(0.0)      //constructor (no args)
{ }                                    //Note: no one-arg constructor
Distance(int ft, float in) : feet(ft), inches(in)          //constructor (two args)
{ }
void getdist()                         //get length from user
```

```
{
cout << "\nEnter feet: "; cin >> feet;
cout << "Enter inches: "; cin >> inches;
}
void showdist() //display distance
{ cout << feet << "\'-" << inches << '\"'; }
};
int main()
{
Distance dist1(11, 6.25); //two-arg constructor
Distance dist2(dist1); //one-arg constructor
Distance dist3 = dist1; //also one-arg constructor
//display all lengths
cout << "\ndist1 = "; dist1.showdist();
cout << "\ndist2 = "; dist2.showdist();
cout << "\ndist3 = "; dist3.showdist();
cout << endl;
return 0;
}
```

ecopycon.cpp

```
// initialize objects using default copy constructor
#include <iostream>
using namespace std;
class Distance //English Distance class
{
private:
int feet;
float inches;
public:
//constructor (no args)
Distance() : feet(0), inches(0.0)
{ }                                    //Note: no one-arg constructor
Distance(int ft, float in) : feet(ft), inches(in)      //constructor (two args)
{ }
void getdist()                         //get length from user
{
cout << "\nEnter feet: "; cin >> feet;   cout << "Enter inches: "; cin >> inches; }
void showdist()                        //display distance
{ cout << feet << "\'-" << inches << '\"'; }
};
int  main()
```

```
{
Distance dist1(11, 6.25);               //two-arg constructor
Distance dist2(dist1);              //one-arg constructor
Distance dist3 = dist1;                 //also one-arg constructor
//display all lengths
cout << "\ndist1 = "; dist1.showdist();   cout << "\ndist2 = "; dist2.showdist();
cout << "\ndist3 = "; dist3.showdist(); cout << endl;
return 0;
 }
```