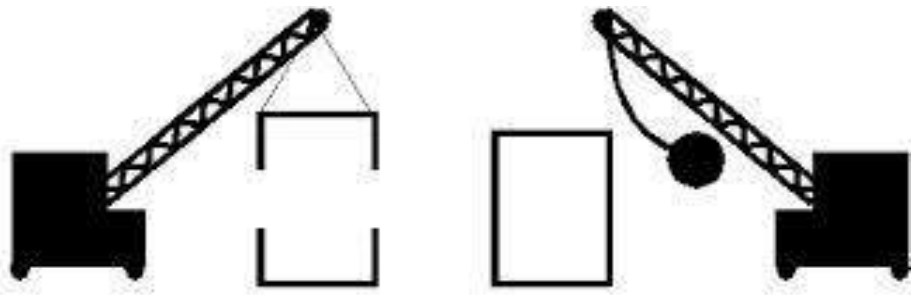
Chapter 9
CONSTRUCTORS AND DESTRUCTORS

Objectives:

- Need for use of constructors
- To identify different types of constructors
- To highlight implicit and explicit declaration of constructors
- To understand the need for constructor overloading
- To understand the need for destructors




```
MyClass *MyObjPtr = new MyClass();
```

```
delete MyObjPtr;
```

9.1 Introduction:

In the previous chapters, we have learnt to use classes, member data and member functions of a class. The use of polymorphism through function overloading has simplified object oriented programming further. In real life applications, sometimes the objects may have to be initialized before using them. The initialization is normally done by a member function which initializes data members to pre-defined values. In this chapter we discuss about special member functions that are used to initialize and destroy the objects of a class automatically so that memory utilization can be optimized.

It is sometimes convenient if an object can initialize itself when it is first created, without the need to make a separate call to a member function. This is possible with special member functions. Automatic initialization is carried out using such a special member function called **constructor**. Generally, we can say, a constructor constructs the data members of an object. i.e., initialized a es a value to the data members.

"A constructor is a special member function that is used in classes to initialize the objects of a class automatically".

A constructor is a member function of a class with the same name as that of the class. A constructor is defined like other member functions of a class. It can be defined either inside the class definition or outside the class definition.

It is called constructor because it constructs the values of data members of the class. The following rules are used for writing a constructor function:

1. A Constructor always has name that is same as the class name of which they are the members. This will help the compiler to identify that they are the constructors.
2. There is no return type for constructors (not even void). Since, the constructor is called automatically by the system, there is no program for it to return anything to; a return value would not make sense.
3. A constructor should be declared in public section.
4. A constructor is invoked automatically when objects are created. Constructors can have default arguments.
5. It is not possible to refer to the address of the constructors.
6. The constructors make implicit calls to the operators **new** and **delete** when memory allocation is required.

9.2 Declaration of constructor inside the class definition

Example 9.1:
```
class test
{
        int m, x, y;
    public:

        test()          //constructor inside the class definition
        {
            x = 0; y = 0;
        }
};
```

Program 9.1: Use of constructor.

```
#include <iostream.h>
#include <iomanip.h>
class counter
{
    private:
        int a;
    public:
        counter()
        {
            a = 0;
        }
        void inc_count()
        {
            a++;
        }
        int get_count()
        {
            return a;
        }
};
int main()
{
    counter c1, c2;
    cout<<"c1 = "<<c1.get_count();
    cout<<"c2 = "<<c2.get_count();
    c1.inc_count();
    c2.inc_count();
    cout<<"c1 = "<<c1.get_count()<<endl;
    cout<<"c2 = "<<c2.get_count()<<endl;
}
```

The above code segment contains a constructor with the name counter(), which is same as the class name. This constructor initializes the class variable a to 0. When, in main(), an object for the counter class is defined will invoke or call the constructor counter() and initialize the variable a to 0. i.e., in main() we are defining two instances for the class counter(), c1 and c2. Now, the constructor counter will be called twice automatically, once for object c1 and once for object c2.

Note that when a constructor is declared for a class, initialization of class objects is done automatically.

9.3 Types of constructors:

There are three types of constructors, namely:

1. Default constructor
2. Parameterized constructor
3. Copy constructor

9.3.1 Default Constructor

A constructor which does not accept any arguments is called a zero argument constructor. It is also known as default constructor. The default constructors are useful when the objects are need to be created without having to type initial values. Default constructor simply allocated memory to data members of objects. Features of default constructor are

- For every object created, this constructor is automatically called.
- All objects of a class are initialized to same set of values by the default constructor.
- If different objects are to be initialized with different values, it cannot be done using default constructor.

Note:

- When a user-defined class does not contain an explicit constructor, compiler automatically invokes default constructor.
- Declaring a constructor with arguments, hides default constructor.

Syntax: classname::classname //constructor without arguments
{ }

Example 9.2: student() { }

Program 9.2: To show initialization of an object using default constructor.

```
#include<iostream.h>
#include<iomanip.h>
class x
{
        private:
                int     a, b;
        public:
                x() { a=10, b=20; }
                void display()
                {
                        cout<<a<<setw(5)<<b<<endl;
                }
};
void main()
{
        x   obj1,obj2;
        obj1.display();
        obj2.display();
}
```

```
10  20
10  20
```

Program 9.3: To show the default initialization of constructor member function using scope resolution operator.

```
#include<iostream.h>
#include<string.h>
#include <ctype.h>
class student
{
        private:
                int     regno;
                char    name[20];
        public:
                        student();                  //constructor
                void    display();
};
student : :student()
{
        regno = 0;
        strcpy(name, "Book");
}
```

```
void student::display()
{
        cout<<"Reg. No: "<<regno<<endl;
        cout<<"Name: "<<name<<endl;
}
void main()
{
        student s1;
        cout<<"Use of default constructor:"<<endl;
        s1.display();
}
```

```
Use of default constructor:
Reg. No: 0
Name: Book
```

The above program shows the use of default constructor student that initializes the members automatically as soon as they are created.

Disadvantages of default constructor:

- When many objects of the same class are created, all objects are initialized to same set of values by default constructor
- It is not possible to initialize different objects with different initial values using default constructor.

9.3.2 Parameterized Constructors:

A constructor that takes one or more arguments is called parameterized constructor. Using this constructor, it is possible to initialize different objects with different values.

Parameterized constructors are also invoked automatically, whenever objects with arguments are created. The parameters are used to initialize the object.

Features of parameterized constructors:

- The parameterized constructors can be overloaded.
- For an object created with one argument, constructor with only one argument is invoked and executed.
- The parameterized constructor can have default arguments and default values.

Invoking constructors

A constructor is automatically invoked by C++ compiler with an object declaration. The constructors can be invoked through the following methods:

1. Explicit call
2. Implicit call
3. Initialization at the time of declaration with = operator

Explicit call

In explicit call, declaration of an object is followed by assignment operator, constructor name and argument list enclosed in parenthesis.

Program 9.4: To use parameterized constructor through explicit call

```
#include<iostream.h>
class num
{
        private:
                int a,b;
        public:
                num(int p, int q) {a = p,b = q;}
                void display()
                {
                        cout<<"a = "<<a<<" and b = "<<b<<endl;
                }
};
        void main()
        {
                num obj1=num(10,20);
                num obj2=num(40,50);
                cout<<"First construction: ";obj1.display();
                cout<<"Second construction: ";obj2.display();
        }
```

```
First construction: a = 10 and b = 20
Second construction: a = 40 and b = 50
```

In the above program code the objects obj1 and obj2 are called explicitly by using parameterized constructor.

Implicit call

An Implicit call means the declaration of the object is followed by argument list enclosed in parentheses.

Program 9.5: Program to intialise the data members using implicit declaration

```
#include<iostream.h>
class   num
{
        private:
```

```
        int a,b;
    public:
        num(int   m, int  n)
        { a = m, b = n; }
    void display()
    {
        cout<<"a= "<<a<<" b= "<<b<<endl;
    }
};
void main()
{
    num obj1(10,20);
    num obj2(40,50);
    obj1.display();
    obj2.display();
}
```

```
    a=10    b= 20
    a=40     b=50
```

Note that:

1. One parameterized constructor num(int m, int n) is defined inside the class.
2. For each object, different values are passed from function main(). Therefore different objects can be initialized to different values.

Initialization of objects during declaration with assignment operator

This method is used for the constructor with exactly one argument. In this method declaration is followed by assignment operator and value to be initialized.

Program 9.6: To initialize objects using assignment operator

```
#include<iostream.h>
class num
{
    private:
        int a;
    public:
        num(int m) { a = m; }
        void display()
        {
            cout<<a<<endl;
        }
};
void main()
{
```

```
        num obj1=100;
        num obj2=200;
        cout<<"Object1 = ";obj1.display();
        cout<<"Object2 = ";obj2.display();
    }
```

```
Object1 = 100
Object2 = 200
```

In the above example, the constructors obj1 and obj2 are initialized using = operator.

Note that

1. This method is applicable only to the constructors that have exactly one parameter.
2. If a class contains at least one parameterized constructor, then necessary arguments have to be passed to the constructor in the declaration itself.
3. If user does not want to pass arguments, then default constructor must be created in the class.

Example 9.1:

```
num obj1(10,20);
num obj2;                    //error
num() {}     //This default constructor must be included in
             //class num to avoid errors
```

The above example shows that it is necessary to use a default constructor num() to avoid error during the declaration of num obj2;

9.3.3 Copy constructor

Copy constructor is a parameterized constructor using which one object can be copied to another object. Copy constructors are used in the following situations.

- To initialize an object with the values of already existing objects.
- When objects must be returned as function values
- To state objects as by value parameters of a function.

Copy constructor can accept a single argument of reference to same class type. The argument must be passed as a constant reference type.

Syntax: classname :: classname(classname &ptr)

Example 9.2: x::x(x &ptr)

Here, x is a class name and ptr is a pointer to a class object x.

If there is more than one argument present in the copy constructor, it must contain default arguments.

Note that:

1. Copy constructor is not invoked explicitly.
2. Copy constructor is invoked automatically when a new object is created and equated to an already existing object in the declaration statement itself.

```
Example 9.3:    x    a1;                  //default constructor
                x    a2 = a1;             //copy constructor
                a1.display();
```

The above example shows the use of copy constructor to create a new object a2 using existing object a1.

3. When a new object is declared and existing object is passed as a parameter to it in the declaration, then also copy constructor is invoked.

```
Example 9.4:    x    a1(100,200);//parameterized constructor
                x    a2(a1);      //copy constructor is invoked for
                                  //object a2 with a1 as parameter
```

4. When an object is passed to a function using pass-by-value, copy constructor is automatically called.

```
Example 9.5:    void test( x )
                {
                        _________
                        _________
                }
                main()
                {
                        x    a;
                        test(a);      //copy constructor is invoked
                }
```

5. Copy constructor is invoked when an object returns a value.

Example 9.6:

```
class measure
{
        int feet;
        float inches;
        measure sum(measure&);
};
```

```
measure measure::sum(measure &m)
{
        feet=feet+m.feet;
        inches=inches+m.inches;
        if(inches>=12)
        {
                feet++;
                inches=inches-12;
        }
         return measure(feet,inches);
}
```

Program 9.7: To find factorial of a number using copy constructor

```
#include<iostream.h>
class copy
{
        int var;
    public:
        copy(int temp)
        {
                var = temp;
        }
        int calculate()
        {
                int fact, i;
                fact = 1;
                for(i = 1; i <= var; i++)
                        fact = fact * i;
                return fact;
        }
};
void main()
{
    int n;
    cout<<"Enter the number: ";
    cin>>n;
    copy obj(n);
    copy cpy = obj;
    cout<<"Before copying: "<<n<<"! = "<<obj.calculate()<<endl;
    cout<<"After copying: "<<n<<"! = "<<cpy.calculate()<<endl;
}
```

```
Enter the number: 5
Before copying: 5! = 120
After  copying: 5! = 120
```

9.4 Constructor Overloading

Constructors are used to initialize the data members of a class. We can use constructors also to specify the input values for the data members. But the default constructor cannot be used for this purpose. So, we have to overload the default constructor. Overloading a constructor means specifying additional operation by passing the required arguments. Depending on the requirement one can define any number of overloaded constructors in a single class. Depending on the type and number of arguments passed, the compiler decides which version of the constructor to invoke during object creation. The following example has overloaded constructor that is used to specify the input values for the data members of the class.

Program 9.8: To find the simple interest

```
#include<iostream.h>
  class simpleinterest
  {
       private:
              float p, r, t, si;
       public:
              simpleinterest()    //default constructor
              { }
              simpleinterest(float x, float y, float z)//parameterized
              {                                        //constructor
                     p = x;
                     r = y;
                     t = z;
              }
              void computesi()
              {
                     cout<<"Simple interest is :"<< (p * r * t)/100.0;
              }
  };
  void main()
  {
       simpleinterest si1, si2(10000.0, 12.0, 2.0);
       si2.computesi();
  }
```

```
Simple interest is 2400.0
```

In the above code segment, there are two constructors, one is simpleinterest() and another one is simpleinterest(float x, float y, float z). The first one is the default constructor and the second one is the three-argument constructor. When we overload a constructor in a class, then it is the job of the programmer to define too the default constructor.

9.5 Destructors:

As we have seen, a constructor, the special member function, will be automatically called when an object is defined for a class. In the same way, a destructor will be called automatically when an object is destroyed. Destroying an object means, de-allocating all the resources such as memory that was allocated for the object by the constructor. A destructor is a special member function that will be executed automatically when an object is destroyed. It will have, like constructor, the name same as that of the class but preceded by a tilde (~).

Syntax:
```
class classname
{
        private:
                //data variables
                //method
        public:
                classname();            //constructor
                ~classname();           //destructor
}
```

Example 9.7: When we want to define a destructor for a class, then we can do this as shown below.

```
class counter
{
        private:
                int     counter;
        public:
                counter( )              //Constructor
                {
                        counter = 0;
                }
                ~counter()              //Destructor
                { }
};
```

In the above example the object counter is automatically destroyed destructors

- The destructor name is same as that of class. The first character must be tilde (~).

- Destructors do not have a return value. Destructor can never return a value.
- They take no arguments. Therefore destructors cannot be overloaded.
- The most common use of destructors is to de-allocate memory that was allocated for the object by the constructor. Also, they are declared public.

Examplem 9.8:

```
class account
{
        private:
                float   balance;
                float   rate;
        public:
                account();              //constructor
                ~account();             //destructor
};
```

Program 9.9: To show the use of destructor

```
#include<iostream.h>
#include<conio.h>
#include  <iomanip.h>
class num
{
         private:
                int x;
        public:
                num();
                void display();
                ~num();
};
 num::num()
{
        cout<<“ In constructor: \n”;
        x=100;
}
num::~num()
{
        cout<<“in  destructor”<<endl;
}
void num::display()
{
        cout<<“Value of x = ”<<x<<endl;
 }
int main()
 {
        clrscr();
```

```
        num a;
        a.display();
        getch();
        return 0;
}
```

```
In constructor:
Value of x = 100
In destructor
```

In the above program after executing the program, before the control is transferred out of the function main(), the destructor ~string() is invoked for each object a, so that memory allotted for data member is de allotted. The object y and p are deleted using the destructor.

Points to remember:

- A Constructor is a special member function that is executed automatically whenever an object of a class is created.
- Constructors should have either public or protected access.
- The three types of constructors are default constructor, parameterized constructor and copy constructor.
- A constructor that does not take any arguments is a default constructor.
- A constructor that takes one or more arguments is called parameterized constructor.
- Parameterized constructor can be invoked by explicit or implicit call.
- Copy constructor is a parameterized constructor using which one object can be copied to another object.
- Constructor overloading means passing arguments to the constructor.
- Destructors are member functions that destroy an object automatically.

One mark questions:

1. What is a constructor?
2. Write one reason which defines the need to use a constructor. Simplify
3. What should be the access parameters for constructor declaration?
4. Can a constructor return a value to a calling function?
5. How many types of constructors are there?
6. What is a default constructor?
7. What is the drawback of default constructor?
8. Is it possible to overload a default constructor?

9. What is a parameterized constructor?
10. Write any one feature of parameterized constructor.
11. Name two methods through which constructors can be invoked.
12. What is an explicit call?
13. What is an implicit call with reference to constructors?
14. When is =used with constructors?
15. What is a copy constructor?
16. Write the syntax for declaration of copy constructor.
17. Can a copy constructor be invoked explicitly?
18. What is meant by constructor overloading?
19. What is a destructor?
20. Which operator is used with destructor?

Two marks questions:

1. What is a constructor? Give an example
2. Why are constructors needed in a program? Justify
3. Write the syntax and example for default constructor.
4. Mention the features of parameterized constructors.
5. Which are the different methods through which constructors are invoked?
6. Write an example to show the use of parameterized constructor through explicit call.
7. When is copy constructor used in a program?
8. Write syntax and example for copy constructor.

Three marks questions:

1. Mention three types of constructors
2. What are the features of default constructors?
3. What are the disadvantages of default constructor?
4. Write short notes for constructor overloading.

Five marks questions:

1. Write the rules for writing a constructor function.
2. Explain default constructor with syntax and example.
3. Explain parameterized constructor with syntax and example.
4. Explain with an example to show how a constructors is invoked explicitly.
5. Explain the features of copy constructor.
6. Explain destructors with syntax and example.

*****