Chapter 10

Constructors and Destructors

10.1 Introduction

In the examples of classes, we have used member functions like input(), getdata() etc. to initialize the private data members of a class. The function call statements are used with the objects that have already been created. The functions are unable to initialize data members at the time of creation of their objects.

The aim of C++ is that the class behaves like a basic data type. A class type variable (object) should be initialized when it is declared in the same way as basic data type variables.

In this chapter, we will discuss a special member function called constructor which is used to initialize objects when they are created. An another member function destructor that destroys the object when they are no longer required.

10.2 Constructors

A constructor is a special member function that is used to initialize the objects of its class. It is called automatically when the objects of its class are created.

Special Characteristics of constructor functions are:

- Its name is same as class name.
- It must be declared in public section.
- Invoked automatically when objects are created.
- Do not have any return type, not even void and hence, it can't return any value.
- It can't be inherited.
- We can't access their addresses.

For example

```
class point
{
        int x,y;
public:
point(void);   //constructor declared
 ---------
 ---------
};
point :: point(void)   //constructor defined
{
        x=0;
        y=0;
}
```

When we declare the object of the class point. For example
point p;
The constructor in the class is automatically called and initialize the private data members x and y to zero for the object p.
A constructor with no arguments is called default constructor. If no such constructor is defined in a class, then compiler provides a default constructor to create the object of the class.

10.3 Parameterized constructors

The constructors that receive arguments are called parameterized constructors. For example

```
class point
{
        int x, y;
public:
        point(int a, int b);     // parameterized constructor
        {
        --------
        --------
        }
};
point::point(int a, int b)
{
        x=a;
        y=b;
}
```

The parameterized constructors can be called in two ways:

```
    point p= point(10,20);   //explicit call
```

This statement create an object p and passes the values 10 and 20 to it.

```
point p(10,20);                //implicit call
```

This statement works same as above statement.
The constructor functions can also be defined as inline functions. For example

```
class point
{
        int x, y;
public:
        point(int a, int b)
        {
                x=a;
                y=b;
```

```
          }
 --------
 --------
};
```

The arguments of a constructor can be of any type except the class to which it belongs. For example

```
class X
{
--------
--------
public:
X(X);
};
```

is illegal.

But a constructor can accept a reference to its own class as an argument. For example

```
class X
{
--------
--------
public:
X(X&);
};
```

is valid and the constructor is called copy constructor.

Program 10.1: Parameterized constructor

```
#include<iostream>
using namespace std;
class rectangle
{
          int length;
          int breadth;
public:
          rectangle(int a, int b)
          {
                    length=a;
                    breadth=b;
          }
          void area()
          {
                    cout<<"Area="<<length*breadth;
          }
};
int main()
{
        rectangle r(5,10);
        return 0;
}
```

The output of the program 10.1 would be:

```
Area=50
```

10.4 Multiple constructors in a class

A class can have more than one constructors and it is called constructor overloading.

Program 10.2: Overloaded constructors

```
#include<iostream>
using namespace std;
class point
{
         int x,y;
public:
        point()// no argument constructor
        {
                x=0;
                y=0;
        }
        point(int a)    //one argument constructor
        {x=y=a;}
        point(int m, int n)     //two arguments constructor
        {
                x=m;
                y=n;
        }
        void show()
        {
                cout<<"x="<<x<<"\n";
                cout<<"y="<<y<<"\n";
        }
};
int main()
{
        point p1;
        point p2(5);
        point p3(7,11);
        cout<<"Coordinates of p1 are\n";
        p1.show();
        cout<<"Coordinates of p2 are\n";
        p2.show();
        cout<<"Coordinates of p3 are\n";
        p3.show();
        return 0;
}
```

The output of the program 10.2 would be:

```
Coordinates of p1 are
x=0
y=0
Coordinates of p2 are
x=5
y=5
Coordinates of p3 are
x=7
y=11
```

In the above program the point has three constructors. First is no argument constructor and it initializes the object with zero values. Second constructor receives one value as an argument and initialize the object with this value. Third constructor receives two arguments and initialize the object with these two values.

10.5 Constructor with default arguments

The constructor can take default arguments. For example
point(int a, int b=0);
Note that default arguments are given from right to left. The default value of argument b is zero. Then, the statement
point p(5);
assign the value 5 to a and 0 to b(by default). But the statement
point(7,11);
assign the value 7 to a and 11 to b because when actual parameters are given, they overrides the default arguments.
If one argument constructor is also present with this constructor , then the calling statement
point p(5);
is unable to decide which constructor is to be called and an ambiguity is created. The compiler will generate an error message.

10.6 Dynamic initialization of objects

The initial value of an object can be provided at the run time. The advantage of dynamic initialization is that we can give different input formats by using constructor overloading.

Program 10.3: Dynamic initialization of objects

```
#include<iostream>
using namespace std;
class shape
{
        float length, breadth;
        float radius;
        float area;
public:
        shape() {}
        shape(float r)
        {
                radius=r;
                area=3.14*r*r;
        }
        shape(float l, float b)
        {
                length=l;
                breadth=b;
                area=length*breadth;
        }
        void display()
        {
                cout<<"Area="<<area<<"\n";
        }
};
int main()
{
        shape circle, rectangle;
        float r, l, b;
        cout<<"Enter the radius of circle\n";
        cin>>r;
        circle=shape(r);
        cout<<"Enter the length and breadth of rectangle\n";
        cin>>l>>b;
        rectangle=shape(l,b);
        cout<<"Area of circle\n";
        circle.display();
        cout<<"Area of rectangle\n";
        rectangle.display();
        return 0;
}
```

```
The output of the program 10.3 would be:
Enter the radius of circle
5
Enter the length and breadth of rectangle
17      8
Area of circle
78.5
Area of rectangle
136
```

10.7 Copy constructor

A constructor that is used to declare and initialize object from another object of the same class is known as copy constructor. A copy constructor takes a reference to an object of the same class as an argument.

Program 10.4: Copy constructor

```
#include<iostream>
using namespace std;
class product
{
        int code;
public:
        product(){ }   // default constructor
        product(int x)//parameterized constructor
        {
                code=x;
        }
        product(product &y)         //copy constructor
        {
                code=y.code;          //copy the value
        }
        void display(void)
        {
                cout<<code;
        }
};
int main()
{
        product p1(10);
```

```
            product p2(p1);         //copy constructor called
            product p3=p1;          //again copy constructor called
            cout<<"Code of p1:";
            p1.display();
            cout<<"\nCode of p2:";
             p2.display();
            cout<<"\nCode of p3:";
            p3.display();
            return 0;
}
```

The output of the program 10.4 would be:

```
Code of p1:10
Code of p2:10
Code of p3:10
```

Note: When no copy constructor is defined in the program, the compiler supplies its own copy constructor.

10.8 Destructors

A special member function of the class that is used to destroy the objects that have been created by constructor.

Special characteristics of destructors:

- Its name is same as class name but preceded by tilde(~).
- It never takes any argument and does not return any value.
- It is invoked implicitly by the compiler upon exit from the program or block or function.

The following program shows the destructor is invoked implicitly by the compiler.

Program 10.5: Implementation of Destructor

```
#include<iostram>
using namespace std;
class sample
{
   sample()                          // Constructor
   {
      cout<<"Object created\n";
   }
   ~sample()                         // Destructor
   {
       cout<<"Object destroyed";
   }
};
```

```
int main()
{
 sample s;
return 0;
}
```

The output of the program 10.5 would be:

```
Object created
Object destroyed
```

The use of destructors are to free the allocated memory to objects at run time. So, that the freed memory can be reuse for another program or objects. The memory is allocated to an object by using new operator in constructor function and de-allocated by using delete operator in destructor function.

Program 10.6: Memory de-allocation of an object using destructor.

```
#include<iostream>
using namespace std;
class sample
{
        char *t;
public:
        sample(int length)
        {
          t=new char[length];
          cout<<"Character array of length"<<length<<"created";
        }
        ~sample()
        {
         delete t;
         cout<<"\n Memory de-allocated for the character array";
        }
};

int main()
{
 sample s(10);
 return 0;
}
```

The output of the program 10.6 would be:

```
Character array of length 10 created
```

Memory de-allocated for the character array

Important Points

- A constructor is a special member function that is used to initialize the objects of its class.
- The arguments of a constructor can be any type except the class to which it belongs.
- A constructor can accept a reference to its own class as an argument.
- A constructor that is used to declare and initialize object from another object of the same class is known as copy constructor.
- When no copy constructor is defined in the program, the compiler supplies its own copy constructor.
- A special member function of the class that is used to destroy the objects that have been created by constructor is called destructor.

Practice Questions

Objective type questions:

Q.1 Which is true with respect to constructor?
A. Its name is same as class name.
B. It must be declared in public section.
C. Invoked automatically when objects are created.
D. All of these

Q.2 Constructors that take arguments are called
A. Default constructors B. No argument constructors
C. Parameterized constructors D. None of these

Q.3 A constructor that is used to declare and initialize object from another object of the same class is known as
A. Default constructor B. Copy constructor
C. Parameterized constructor D. None of these

Q.4 Which is true with respect to destructors?
A. Its name is same as class name but preceded by tilde(~).
B. It never takes any argument and does not return any value.
C. It is invoked implicitly by the compiler upon exit from the program or block or function.
D. All of these

Very Short Answer Type Questions

Q.1 What are constructors?

Q.2 What are parameterized constructors?

Q.3 What is constructor overloading?

Q.4 What is copy constructor?

Short Answer Type Questions

Q.1 What are the characteristics of constructors?

Q.2 What are destructors? Write its properties.

Q.3 What are the uses of destructors?

Essay Type Questions

Q.1 Explain constructor with default arguments.

Answer Key

1. D	2. C	3. B	4. D