Unit IV Object Oriented Programming with C++

CHAPTER 14

Classes and objects

Learining Objectives

After learning this chapter, the students will be able to

- Understand the purpose of classes, objects Constructors and Destructors
- able to construct C++ programs using classes with Constructors and Destructors
- Execute and debug class programs with Constructors and Destructors

14.1 Introduction to Classes

The most important feature of C++ is the “Class”. It’s significance is highlighted by the fact that Bjarne Stroustrup initially gave the name 'C with classes'. C++ offers classes, which provide the four features commonly present in OOP languages: **Abstraction, Encapsulation**, **Inheritance** and **Polymorphism**.

14.1.1 Need for Class

Class is a way to bind the data and its associated functions together. Classes are needed to represent real world entities that not only have data type properties but also have associated operations. It is used to create user defined data type

14.1.2 Declaration of a class

A class is defined in C++ using the keyword **class** followed by the name of the class. The body of the class is defined inside the curly brackets and terminated either by a semicolon or a list of declarations at the end.

Note

The only difference between structure and class is the members of structure are by default **public** where as it is **private in class**.

The General Form of a class definition

```
class class_name
{
private:
        variable declaration;
        function declaration;
protected:
        variable declaration;
        function declaration;
public:
        variable declaration;
        function declaration;
};
```

- The class body contains the declaration of its members (Data member and Member functions).

- The class body has three access specifiers (visibility labels) viz., private , public and protected.

14.1.3 Class Access Specifiers

Data hiding is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type. The access restriction to the class members is specified by public, private and protected sections within the class body. The keywords public, private and protected are called access specifiers. The default access specifier for members is private.

The Public Members

A public member is accessible from anywhere outside the class but within a program.You can set and get the value of public data members even without using any member function.

The Private Members

A private member cannot be accessed from outside the class. Only the class member functions can access private members.By default all the members of a class would be private.

The Protected Members

A protected member is very similar to a private member but it provides one additional benefit that they can be accessed in child classes which are called derived classes (inherited classes).

Example

Keyword class intimates the compiler that it is a class definition

Class name or tag name acts as a user defined data type. Using this, object of the same class type will be created.

```
class student
{
private:
        char name [10];
        int rollno, mark1, mark2, total;
protected:
        void accept();
        void compute();
public:
        void display();
};
```

These are private access specifier members
That means these members cannot be accessed from outside

These are protected access specifier members
These members also cannot be accessed from outside

Members under this specifier can be accessed from outside

Note

If all members of the class are defined as private ,then the object of the class can not access anything from the class.

Activity 1

State the reason for the invalidity of the following code fragment

(i)	(ii)

class count
{
  int first;
  int second;
  public:
   int first;
};
	class item
	{
	int prd;
	  };
	item int prdno;

14.1.4 Definition of class members

Class comprises of members. Members are classified as Data Members and Member functions. Data members are the data variables that represent the features or properties of a class. Member functions are the functions that perform specific tasks in a class. Member functions are called as methods, and data members are also called as attributes.

Example

```
Class result                          Data members
{
Private:
        char name [10];             }
        int rollno,mark1, mark2, total; }

Public:
        void accept();              }
        void display();             }
};

                          Member functions
```

Note

Classes also contain some special member functions called as constructors and destructors.

14.1.5 Defining methods of a class

Without defining the methods (functions), class definition will become incomplete. The member functions of a class can be defined in two ways.

(1) Inside the class definition

(2) Outside the class definition

(1) Inside the class definition

When a member function is defined inside a class, it behaves like inline functions. These are called Inline member functions.

Note

If a function is inline, the compiler places a copy of the code at each point where the function is called at compile time.

(2) Outside the class definition

When Member function defined outside the class just like normal function definition (Function definitions you are familiar with) then it is be called as **outline member function or non-inline member function. Scope resolution operator (::) is used for this purpose.** The syntax for defining the outline member function is

Syntax

```
return_type class_name :: function_name (parameter list)
{
        function definition
}
```

For example:

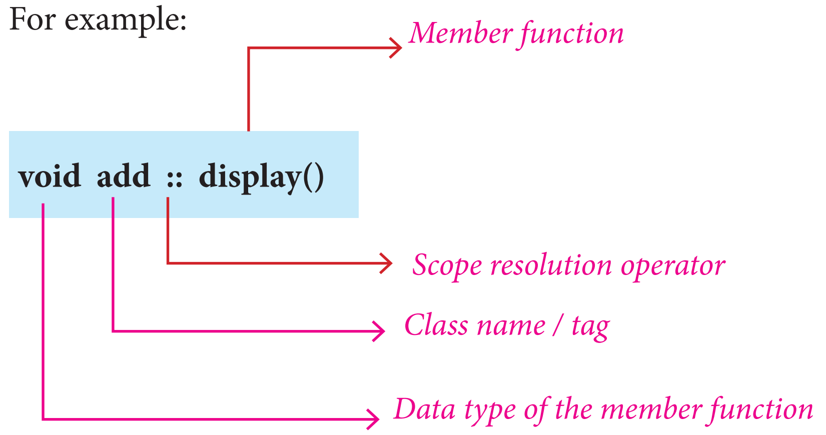


Illustration 14.1 Inline and Outline member function

Absence of access specifier means that members are private by default..

```
# include <iostream>
using namespace std;
class Box
{
        double width;        // no access specifier mentioned
public:
        double length;
        void printWidth( )          //inline member function definition
        {
                cout<<"\n The width of the box is..."<<width;
        }
        void  setWidth( double w );          //prototype of the function
};
void Box :: setWidth(double w)      // outline member function definition
{
        width=w;
}
int main( )
{
Box b;          // object for class Box
b.setWidth(10.0);       // Use member function to set the width.
b.printWidth( );                //Use member function to print the width.
return 0;
}
```

Output:

```
The width of the box is... 10
```

Note

Declaring a member function having looping construct, switch or goto statement as inline is not advisable.

14.2 Creating Objects

A class specification just defines the properties of a class. To make use of a class, the variables of that class type have to be declared. The class variables are called *object*. Objects are also called as *instance* of class.

For example

student s;

In the above statement **s** is an instance of the class **student**.

Objects can be created in two methods,

(1) Global object

(2) Local object

(1) Global Object

If an object is declared outside all the function bodies or by placing their names immediately after the closing brace of the class declaration then it is called as *Global object*. These objects can be used by any function in the program

(2) Local Object

If an object is declared with in a function then it is called *local object*. It cannot be accessed from outside the function.

Illustration 14.2 The use of local and global object

```
# include <iostream>
# include <conio.h>
using namespace std;
class add
{
  int a,b;
  public:
    int sum;
  void getdata()
  {
    a=5;
    b=10;
    sum = a+b;
  }
} a1;                        //global object
add a2;                      //global object
int main()
{
add a3;           // Local object
a1.getdata();
a2.getdata();
a3.getdata();
cout<<a1.sum;
cout<<a2.sum;
cout<<a3.sum;
return 0;
}
```

Output:

```
151515
```

ACTIVITY 2

Identify the error in the following code fragment

```
class A
{
        float x;
        void init()
        {
        A a1;
        X1.5=1;
        }};
void main()
{    A1.init(); }
```

14.3 Memory allocation of objects

The member functions are created and placed in the memory space only when they are defined as a part of the class specification. Since all the objects belonging to that class use the same member function, **no separate space is allocated for member functions when the objects are created. Memory space required for the member variables are only allocated separately for each object** because the member variables will hold different data values for different objects

Illustration 14.3 Memory allocation for objects

```
# include <iostream>
using namespace std;

class product
{
        int code, quantity;
        float price;
        public:
        void assignData();
        void Print();
};
int main()
{
        product p1, p2;
        cout<<"\n Memory allocation for object p1 " <<sizeof(p1);
        cout<<"\n Memory allocation for object p2 " <<sizeof(p2);
        return 0;
}
```

Output:

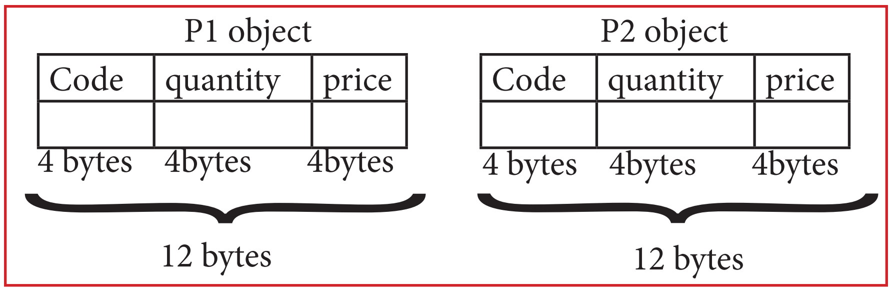
```
 Memory allocation for object p1  12
 Memory allocation for object p2  12
```

Member functions assignData( ) and Print( ) belong to the common pool in the sense both the objects p1 and p2 will have access to the code area of the common pool.

Note

The members will be allocated with memory space only after the creation of the class type object

Memory for Objects for p1 and p2 is illustrated:



ACTIVITY 3

What is the size of the objects s1, s2?

```
class sum
{
        int n1,n2;
        public:
        void add(){int n3=10;n1=n2=10;}
}  s1,s2;
```

14.4 Referencing class members

The members of a class are referenced (accessed) by using the object of the class followed by the dot (membership) operator and the name of the member.

The general syntax for calling the member function is:

Object_name . function_name(actual parameter);

For example consider the following illustration

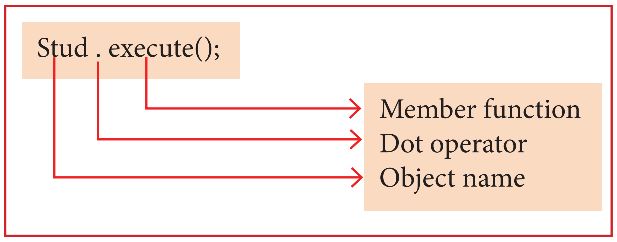


Illustration 14.4 C++ program to illustrate the communication of object:

```
#include<iostream>
using namespace std;
class compute
{
 int n1,n2;      //private by default
 public :
 int n;
 int add (int a, int b)   //inline member  function
 {
 int c=a+b;      //int c ; local variable for this function
 return c;
 }

}c1,c2;
int main()
{
 c1.n =c1.add(12,15); //member function is called
 c2.n =c2.add(8,4);
 cout<<"\n Sum of object-1  "<<c1.n;
 cout<<"\n Sum of object-2  "<<c2.n;
 cout<<"\n Sum of the two objects are  "<<c1.n+c2.n;
          return 0;
}
```

Output:

```
Sum of object-1  27
Sum of object-2  12
Sum of the two objects are  39
```

Note

Even an array of objects can be created for a class. It is declared and defined in the same way as any other type of array.

Example :

student s[10];

Where student is the class name and s[10] is 10 objects created for the student class.

14.5 Introduction to Constructors

The definition of a class only creates a new user defined data type. The instances of the class type should be instantiated (created and initialized) . Instantiating object is done using constructor.

14.5.1 Need for Constructors

An array or a structure in c++ can be initialized during the time of their declaration.

For example

```
struct sum
   {
          int n1,n2;
   };
class add
  {
          int num1,num2;
  };
int  main()
   {
          int arr[]={1,2,3};        //declaration and initialization of array
          sum s1={1,1};            //declaration and initialization of structure object
          add  a1={0,0};          // class object declaration and initialization throws
                                                                       compilation error
    }
```

Member function of a class can access all the members irrespective of their associated access specifier.

The initialization of class type object at the time of declaration similar to a structure or an array is not possible because the class members have their associated access specifiers (private or protected or public). Therefore Classes include special member functions called as *constructors*. The constructor function initializes the class object.

14.6 Declaration and Definition

When an instance of a class comes into scope, a special function called the *constructor* gets executed. The constructor function name has the same name as the class name. The constructors return nothing. They are not associated with any data type. It can be defined either inside class definition or outside the class definition.

Example 1:

Illustration 14.5 A constructor defined inside the class specification.

```
#include<iostream>
using namespace std;
class Sample
{
        int i,j;
        public :
          int k;
          Sample()
          {
             i=j=k=0;   //constructor defined inside the class
          }
};
```

14.6.1 Functions of constructor

As we know now that the constructor is a special initialization member function of a class that is called automatically whenever an instance of a class is declared or created. The main function of the constructor is

1) To allocate memory space to the object and

2) To initialize the data member of the class object

There is an alternate way to initialize the class objects but in that case we have to explicitly call the member function.

14.7 Types of constructors

There are different types of constructors.

- **Default Constructors**

A constructor that accepts no parameter is called default constructor. For example in the class Data, Data ::Data() is the default constructor . Using this constructor Objects are created similar to the way the variables of other data types are created. If a class does not contain an explicit constructor (user defined constructor) the compiler automatically generate a default constructor.

- **Parameterized Constructors**

A constructor which can take arguments is called parameterized constructor .This type of constructor helps to create objects with different initial values. This is achieved by passing parameters to the function.

Example :

```
Data :: Data(int,int);
```

- **Copy Constructors**

A constructor having a reference to an already existing object of its own class is called copy constructor. It is usually of the form Data (Data&), where Data is the class name.

A copy constructor can be called in many ways:

1) When an object is passed as a parameter to any of the member functions

   Example void Data::putdata(Data x);

2) When a member function returns an object

   Example Data getdata() { }

3) When an object is passed by reference to an instance of its own class

   For example, Data d1, d2 (d1); // d2(d1) calls copy constructor

Illustration 14.6 Types of constructor

```
#include<iostream>
using namespace std;
class Data
 {
        int i, j;
     public:
        int k;
         Data()
          {
                cout<<"Non Parametrerized constructor";
                 i=0;
                  j=0;
            }
          Data(int a,int b)
          {
                cout<<"Parametrerized constructor";
                 i=a;
                  j=b;
            }
         Data(Data &a)
          {
                cout<<"Copy constructor";
                 i=a.i;
            }

            void display()              //member function
             {
                cout<< i<<j;
              }
        };
int main()
{
            Data  d1,d2(10,20),d3(d2);
            d1.display();
            d2.display();
            d3.display();
             return 0;

}
```

14.8 Invocation of constructors

There are two ways to create an object using parameterized constructor

- Implicit call
- Explicit call

14.8.1 Implicit call

In this method ,the parameterized constructor is invoked automatically whenever an object is created. For example simple s1(10,20); in this for creating the object s1 parameterized constructor is automatically invoked.

14.8.2 Explicit call

In this method ,the name of the constructor is explicitly given to invoke the parameterized constructor so that the object can be created and initialized .

For example

```
simple s1=simple(10,20);        //explicit call
```

Explicit call method is the most suitable method as it creates a temporary object ,the chance of data loss will not arise.A temprory object lives in memory as long as it is being used in an expression.After this it get destroyed.

14.9 Dynamic initialization of Objects

When the initial values are provided during runtime then it is called dynamic initialization.

Illustration14.7 to illustrate dynamic initialization

```
#include<iostream>
using namespace std;
class X
{   int n;
    float avg;
    public:
    X(int p,float q)
    {  n=p;
        avg=q;  }
    void disp()
    {     cout<<"\n Roll numbe:- " <<n;
          cout<<"\nAverage :- "<<avg;         } };
int main()
{
int a ; float b;
          cout<<"\nEnter the Roll Number";
          cin>>a;
          cout<<"\nEnter the Average";
          cin>>b;
          X x(a,b);       // dynamic initialization
          x.disp();
          return 0;
}
```

Output:

```
Enter the Roll Number 1201
Enter the Average 98.6
 Roll numbe:- 1201
Average :- 98.6
```

14.10 Characteristics of Constructors

- The name of the constructor must be same as that of the class
- No return type can be specified for constructor
- A constructor can have parameter list
- The constructor function can be overloaded
- They cannot be inherited but a derived class can call the base class constructor
- The compiler generates a constructor, in the absence of a user defined constructor.
- Compiler generated constructor is public member function
- The constructor is executed automatically when the object is created
- A constructor can be used explicitly to create new object of its class type

14.11 Destructors

When a class object goes out of scope, a special function called the *destructor* gets executed. The destructor has the same name as the class tag but prefixed with a ~**(tilde)**. Destructor function also return nothing and it does not associated with anydata type.

14.11.1 Need of Destructors

The purpose of the destructor is to free the resources that the object may have acquired during its lifetime. A destructor function removes the memory of an object which was allocated by the constructor at the time of creating a object.

14.11.2 Declaration and Definition

A *destructor* is a special member function that is called when the lifetime of an object ends and destroys the object constructed by the constructor. Normally declared under public.

Illustration14.8 To illustrate destructor function in a class

```
#include<iostream>
using namespace std;
class simple
{
private:
int a, b;
public:
simple()
{
a= 0 ;
b= 0;
cout<< "\n Constructor of class-simple ";
}
void getdata()
{
cout<<"\n Enter values for a and b ";
cin>>a>>b;
}
void putdata()
{
cout<<"\nThe two integers are .. ";
cout<<a<<'\t'<< b<<endl;
cout<<"\n The sum = "<<a+b;
}
~simple()
{   cout<<"\n Destructor is executed ";}
};
int main()
{
simple s;
s.getdata();
s.putdata();
return 0;
}
```

Output:

```
Constructor of class-simple
Enter values for a and b  6 7
The two integers are .. 6      7
The sum = 13
Destructor is executed
```

14.12 Characteristics of Destructors

- The destructor has the same name as that class prefixed by the tilde character '~'.
- The destructor cannot have arguments
- It has no return type
- Destructors cannot be overloaded
- In the absence of user defined destructor, it is generated by the compiler
- The destructor is executed automatically when the control reaches the end of class scope to destroy the object
- They cannot be inherited

Points to Remember:

- A class binds data and associated functions together.
- A class in C++ makes a user defined data type using which objects of this type can be created.
- While declaring a class data members , member functions ,access specifiers and class tag name are given.
- The member functions of a class can either be defined within the class (inline) definition or outside the class definition.
- The public members of the class can be accessed outside the class directly by using object of this class type.
- A class binds data and associated functions together.
- A class in C++ makes a user defined data type using which objects of this type can be created.
- While declaring a class data members , member functions, access specifiers and class tag name are given.
- The member functions of a class can either be defined within the class (inline) definition or outside the class definition.
- The public members of the class can be accessed outside the class directly by using object of this class type.
- A class supports OOP features ENCAPSULATION by binding data and functionsassociated together.
- A class supports Data hiding by hiding the information from the outside world through private and protected members.
- When a member function is called by another member function of the same class , it is calledas nesting of member functions.
- The scope resolution operator (::), when used with the class name depicts that the members belong to that class as in class_name :: function_name and only used with the variable name as in :: s variable –name , depicts the global variable.(the one with file scope).
- When an instance of a class comes into scope, a special function called the constructor gets executed.
- The constructor function allocates memory and initializes the class object.
- When an instance of a class comes into scope, a special function called the constructor gets executed.
- When a class object goes out of scope, a special function called the destructor gets executed.
- The constructor function name and the destructor have the same name as the classtag.
- A constructor without parameters is called as default constructor.
- A constructor with default argument is equivalent to a default constructor
- Both the constructors and destructor return nothing. They are not associated with any data type.
- Objects can be initialized dynamically .