UNIT 3 OBJECT AND CLASSES

Structure **Page Nos.**

3.0 INTRODUCTION

Objects and Classes are key to understand the Object-Oriented Programming Language (OOPL)/Object-Oriented Technique. Object-Oriented Programming Languages are based on the concept of abstraction modeled by classes and objects. OOPL is based on the concept of Object-Oriented technique. Object-Oriented technique (approach) for software development has become defacto standard for software development in software industry. Object-Oriented technique is a natural way of thinking or visualizing the real world problem in terms of the objects involved in the system. In Unit 1 of Block 1, we have learnt the basic concepts and characteristics of Object-Oriented technique.

In this Unit, we can begin with the concept of classification which is foundation in Object-Oriented programming. This is followed by what and how C++ supports classes, objects and how objects are used in problem solving. The classes are the crucial components for developing applications in C++. The accessibility rule that controls the visibility of class members (data and methods) is discussed. Moreover, member functions and friend functions are also discussed. This unit also covers a special kind of member function known as static function. Illustrative examples that facilitate you in understanding of the concept are presented with adequate emphasis.

3.1 OBJECTIVES

After going through this unit, you will be able to:

- understand and define own classes;
- understand objects and use classes to create objects;
- use created objects;
- access members of a class;
- explain the need of friend function;
- describe the need of use static member; and
- write simple C++ program.

3.2 CLASSIFICATION

Classification is one of the important concepts of Object-Oriented philosophy including identity, abstraction, encapsulation, inheritance, polymorphism, and persistence. Let us take an example to understand the concept of classification. In the real world environment in which we operate, we need and identify hundreds or thousands of objects to solve the problem. It is very difficult to manage this large number of objects. Could you tell how to handle them easily, that too in such large numbers? Well, Object-Oriented uses classification to group objects that have attributes and behaviours in common and classify them into bigger entities. The bigger entity is called a 'class'. Thus a class defines a group of objects with similar attributes, common operations, a common relationship to the other objects in the class, and common semantics.

Suppose you have several objects like dog, cat, cow, elephant, car, airplane, fighter-plane, rocket, cup, mountain bike, racing bike, tandem bike. There are four animal objects grouped together in the animal class, three plane objects are grouped together into a plane class, and three bike objects group in bike class. The single car is in separate class as is a cup.

Can you tell how did we classify the objects? Yes, of course we can classify, based on the similar properties, common operations, a common relationship to the other objects in the class, and common semantics. But how will we classify the objects based on the concept of common semantics? The term common semantics in classifying objects is best described by an example. Consider two classes, Bus and House given as follows:

Name of the class	: Bus
Attributes	: Cost
	Colour
	Model
Services	: Maintenance

Name of the class	: House
Attributes	: Cost
	Colour
	Model
Services	: Maintenance

If you look at the above classes, you can observe that both have same set of attributes and service functions. Naturally, one can raise the questions. Whether both of these objects be considered of the same class or should they belong to different classes? These questions can be answered by looking at the semantics of two classes. If the underlying semantics is based on object usage, then two classes should not be combined even if their attributes and services are the same but if the underlying semantics is based on the object's asset, then both Bus and House belong to a common class Asset.

From the above discussion, we have understood the concept of classification and how it is used to reduce hundreds or thousands of objects into smaller groups/class, based on their characteristics.

3.3 CLASS

In this section, we will discuss what is class? What is importance of class in C++? We will define a class and also discuss encapsulation which is one of the striking features of a class.

A class is set of objects that shares a common definition described by data and methods.

According to the Webster's New World dictionary, a class is defined as " *A number of people or things grouped together because of certain likeness; kind; sort".* In other words, we can say that *class is an object template.* Every object under that class has the same data format, definition and responds in the same manner to an operation.

A class is a user defined data type like any other built-in data type for example int, char, float, .. etc. It is most important feature of C++. It makes C++ an Object-Oriented language. Can you tell the difference between structure and class ? Yes, of course, the only difference between a structure and a class in C++ is that *by default, the members of a class are private, while by default the members of a structure are public.* In the next section, we will see as to how we will define and use a class in the program.

3.3.1 Defining a Class

A class has a class name, a set of attributes (data members/characteristics) and a set of actions or services(function members/methods). Now, let us see how a class is defined in C++. The common syntax of a class declaration/specification/definition is given as follows:

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

It can be easily seen that a class is defined by the keyword class. The class specifies the type and scope of variables and functions declared inside the class. The variables(data) declared inside the class are known as data members and the functions(methods) are known as member functions. Being the part of the class, data and function are called members of the class. The body of the class is enclosed within braces and terminated by the colon. The keywords private, public and protected are known as visibility labels, which defines the visibility of members. We will discuss the visibility of members in the next section. It is common practice to declare data members as private and member functions as public. Let us see an example to understand how the class is defined.

Before defining a class, you should decide about the members of class i.e. who will be the members of a class. Actually, it depends on the problem domain i.e which type of data problems needs to keep in the class and also which type of operations will be needed to manipulate the data.

Based on the above discussion, suppose we want to define a Employee class. We are interested to display the basic information like ID, name, department etc. For this, first we have to decide the data members and their types required to represent the basic information, then we need a member functions to display basic information. We also need the member function to take information from the real world.

Let us see, how Employee class is defined.

```
class Employee
{
int id;
char name[25];
char deptt[25];
public:
void get_data(void)
{
        cout<< "Enter Employee ID:" <<endl;
        cin>>id;
        cout<<"Enter name:"<<endl;
        cin>>name;
        cout<< "Enter department:"<<endl;
        cin>>deptt;
}
void display_information(void)
{
cout<< "Employee ID=" <<id<<endl;
cout<< "Employee Name=" <<name<<endl;
cout<< "Employee Department=" <<deptt<<endl;
}
};
```

We, generally, give a class some meaningful name by reflecting information it holds. In the above declaration of the class, the name of the class is Employee. Now, Employee becomes a new data type. It is used to define instances of class data type.

3.3.2 Encapsulation

In this section, we will discuss about Encapsulation.

Encapsulation is one of the important characteristics of Object Oriented Programming Language. As we have discussed, a class can be described as a collection of data members and member functions. This property of C++ which allow wrapping up of data and functions into a single unit is called encapsulation. Data encapsulation is most striking feature of a class. Could you tell, what are the advantages of encapsulation? Well, the advantages of encapsulation are data hiding, information hiding and implementation independence. Let us see what these terms mean.

If the implementation details are not known to the user, it is called information hiding. Restrictions of external access to features of a class results in data hiding. The user's interface is not affected by changing the implementation mechanism. A change in the implementation is done easily without affecting the interface. This leads to implementation independence.

3.3.3 Accessibility Rules/Labels

You can see, in class declaration, we have used three terms private, public and protected. Can you tell what is the purpose of these terms in the program? Right, private, public and protected are known as visibility labels, used to control the access to members (data members and member functions) of a class.

Why do we need to control the access of members of a class? Well, as we know that purpose of data encapsulation is to prevent accidental modification of information of a class. Data can be protected from external tempering by users and access to specific methods can also be controlled. However, an entire program cannot be hidden. A part of the program needs to be accessed by users. For this and many other reasons, we need to access the members of a class in controlled way. It is achieved by imposing a set of rules-the manner in which a class is to be manipulated and data and functions of the class can be accessed. The set of rules is known as accessibility rules. This accessibility rules are achieved by using three keywords private, public and protected. These keywords are called access-control specifiers (visibility mode). The visibility of class members is summarized in Table 3.1.

Table 3.1: Visibility of class members

Access-control specifiers	Accessible to	
	Own class members	Objects of a class
private	yes	no
protected	yes	no
public	yes	yes

From the above table, you can observe that private and protected members of a class are accessible only from within other members of the same class. They are not accessible by the objects of a class. Further, public members are accessible both from members of the same class and objects of a class. They are accessible from anywhere where the object is visible. Members of a class without any access specifier are private by default. A class which is totally private is hidden from the external world and will not serve any useful purpose. Can you access the private member of a class from outside a class? Yes, we can have a mechanism to access private data using friends, pointers to members etc. from outside the class.

A class can use all of three visibility/accessibility labels as illustrated below:

```
Class A
{
private:
int x;
void fun1()
{
// This function can refer to data members x, y, z and functions fun1(), fun2() and
fun3()
}
protected:
int y;
void fun2()
{
//This function can also refer to data members x, y, z and functions fun1(), fun2()
and fun3()
}
public :
int z;
void fun3()
```

```
{
//This function can also refer to data members x, y, z and functions fun1(), fun2()
and fun3()
}
};
```

Now, consider the statements

A obja; //obja is an object of class A
int b; // b is an integer variable
The above statements define an object obja and an integer variable b. The accessibility of members of the class A is illustrated through the obja as follows:

1. Accessing private members of the class A:

b=obja.x; //Won't Work: object can not access private data member 'x'
obja.fun1(); //Won't Work: object can not access private member function fun1()
Both the statements are illegal because the private members of the class are not accessible.

2. Accessing protected members of the class A:

b=obja.y; // Won't Work: object can not access protected data member 'y'
obja.fun2(); // Won't Work: object can not access member function fun2()
Both the statements are also illegal because the protected members of the class are not accessible.

3. Accessing public members of the class A:

b=obja.c; //OK
obja.fun3(); //OK
Both the statements are valid because the public members of the class are accessible.

3.4 OBJECTS

In this section, we will study about object. What is object? What is the relationship between object and class? How will we create object? We shall make an attempt to all such address this question.

Webster's New World Dictionary defines object as *"A thing that can be seen or touched; material thing; a person or thing to which action, thought or feeling is directed".* In Object-Oriented domain an object is one of the many things that together constitute the problem domain. You can look around and find many real world examples of objects like person, customer, student, employee, car, dog, table, bike …etc. An object may stand alone or it may belong to a class of similar objects. Examples of objects that may stand alone are knife, frame … etc. Examples of objects that belong to a class of similar objects are: your car, your table … etc.

An object may have a name, a set of attributes, and a set of actions or services.

Can you tell the difference between object and class? Well, objects are the basic run-time entities in Object-Oriented programming language. They occupy space in memory that keeps its state and is operated on by the defined operations on the object, while a class defines a possible set of objects. What is the relationship between object and class? The relationship between a class and objects of that class is similar to the relationship between a type and elements of that type. *A class represents a set of objects that share a common structure and a common behavior, whereas an object is an instance of a class.* In the next section we will see as to how object is created.

3.4.1 Instantiating Object

The declaration of a class does not define any object but only specify the structure of objects i.e. what they will contain. A class must be instantiated in order to make use of the services provided by it. This process of creating objects (variables) of the class is called class instantiation or instantiating of objects. Thus, an object is an instantiation of a class. The common syntax of instantiating object of a class / declaration of a object is as follows:

class className objectName1, objectName2 …;
or
className objectName1, objectName2 …;

For example, let us see, how will we create the object of the Employee class discussed in class section?
class Employee emp1;
or
Employee emp1;

You can see that the declaration of an object is similar to that of any basic type. You can also notice that keyword class is optional. Employee class creates object emp1. More than one object can also be created with a single statement as follows.
Employee emp1, emp2,emp3;

Objects can also be created by placing their names immediately after the closing brace as we do in the case of structure. Thus, the definition
class Employee
{
.
.
.
}emp1,emp2,emp3;
would create objects emp1, emp2 and emp3 of the class Employee.

Now let us see, how will we assess the members of a class? We can assess the member of a class using the member assess operator, dot(.). The syntax for assessing data member of a class is given as follows:
objectName.dataMember;
The syntax for assessing member functions of a class is given as follows.
objectName.functionName(actualArguments);
Let us consider the snapshot of a program which illustrates use of dot (.) operator:

```
class ABC
{
        int x;
        int y;
        void f1(void);
        public:
        int z;
        void f2(void);
};
.
.
void main()
```

```
{
ABC a;

a.x=10;          //error, x is private data
a.z=10;          // OK, z is public data
.
.
a.f1();          //error, f1 is private function
a.f2();          //OK, f2 is public function
.
.
}
```

If you look at the above snapshot of C++ program, you can observe that private data like x and private member function like f1() cannot be accessed through object while public data like z and public member function like f2() are accessed through object. Furthermore, this program shows that you can access the members of a class through dot (.) operator.

Let us consider the complete program employee.cpp which illustrates the declaration of the class Employee with the operations on its object. Further, it is also seen that how we will access the members of a class. This program reads the basic information of employee like id, name, age etc. from the keyboard and display on the screen.

```
//program:employee.cpp
#include<iostream.h>
#include<string.h>
class Employee
{
int id;
int age;
char name[25];
public:
int salary;
void get_data(void)
{
cout<<"Enter ID :"<<endl;
cin>>id;
cout<<"Enter Name:"<<endl;
cin>>name;
cout<<"Enter Age:"<<endl;
cin>>age;
cout<<"Enter Salary :"<<endl;
cin>>salary;
}
void display_info(void)
{
cout<<"\nID     :" <<id<<endl;
cout<<"Name   :"<<name<<endl;
cout<<"Age    :"<<age<<endl;
cout<<"Salary :"<<salary<<endl;
}
};
```

```
void main()
{

Employee e1;                    // first object/variable of a Employee class
Employee e2;                    // second object/variable of a Employee class

cout<<"\nEnter 1st Employee Basic Information:"<<endl;
e1.get_data();                  // object e1 calls member get_data()
cout<<"\nEnter 2nd  Employee Basic Information:"<<endl;
e2.get_data();                  // object e2 calls member get_data()
cout<<"\n1st Employee Basic Information:"<<endl;
e1.display_info();              // object e1 calls member display_info()
cout<<"\n 2nd Employee Basic Information:"<<endl;

e2.display_info();              // object e2 calls member display_info()
}
```

The output of the above program is given below.

```
Enter 1st Employee Basic Information:
Enter ID:
100
Enter Name:
A.K. Malviya
Enter Age
39
Enter Salary
20000
Enter 2nd Employee Basic Information:
Enter ID:
101
Enter Name:
N. Badal
Enter Age
35
Enter Salary
22000

1st Employee Basic Information:
ID      : 100
Name    : A.K. Malviya
Age     : 39
Salary  : 20000
2nd Employee Basic Information:
ID      : 101
Name    : N. Badal
Age     : 35
Salary  : 22000
```

This program shows how we can access the members of a class with the help of the object. Furthermore, it also illustrates that the various operation on the objects of the class Employee. In main(), the statements Employee e1;
Employee e2;
create two objects called e1 and e2 of the employee class. The statements
e1.get_data();
e2.getdata();
initialize the data members of the objects e1 and e2. The statements
e1.display_info();
e2.display_info();
call their display_info() to display the contents of data members namely id, name, age and salary of the employee objects e1 and e2.

☞Check Your Progress 1

1) What are the differences between structures and classes in C++?

……

……

……

2) What is the concept of classification in Object-Oriented Programming Languages?

……

……

……

3) What are empty classes? Explain purpose of empty classes?

……

……

……

4) Write a C+ + program to find out the sum of n numbers.

……

……

……

3.5 MEMBER FUNCTIONS

In this section, we will discuss what is member function? How will we define a member function?

A member function performs an operation required by the class. It is the actual interface to initiate an action of an object belonging to a class. It may be used to read, manipulate, or display the data member. The data member of a class must be declared within the body of the class, while the member functions of a class can be defined in two places:

(i) inside the class definition
(ii) outside the class definition

The syntax of a member function definition changes depending on whether it is defined inside or outside the class declaration/definition. However, irrespective of the location of their definition, the member function must perform the same operation. Thus, the code inside the function body would be identical in both the cases. The compiler treats these two definitions in a different manner. Let us see, how we can define the member function inside the class definition.

The syntax for specifying a member function declaration is similar to a normal function definition except that is enclosed within the body of a class. Foe example, we could define the class as follows:

```
class Number
{
int x, y, z;
public:
void get_data(void);              //declaration
void maximum(void);               //declaration
void minimum(void)                //definition
{
int min;
min=x;
if (min>y)
min=y;
if (min>z)
min=z;
cout<<"\n Minimum value is ="<<min<<endl;
}
};
```

if you look at the above declaration of class number you can observe that the member function get_data() and maiximun() are declared, but they are not defined. The only member function which is defined in the class body is minimum(). When a function is defined inside a class, it is treated as a inline function. Thus, member function minimum is a inline function. Generally, only small functions are defined inside the class.

Now let us see how we can define the function outside the class body. Member functions that are declared inside a class have to be defined outside the class. Their definition is very much like the normal function. Can you tell how does a compiler know to which class outside defined function belong? Yes, there should be a mechanism of binding the functions to the class to which they belong. This is done by the scope resolution operator (**::**). It acts as an identity-label. This label tells the compiler which class the function belongs to. The common syntax for member function definition outside the class is as follows:

```
return_type class_name :: function_name(argument declaration)
{
functionbody
}
```

The scope resolution :: tells the compiler that the function_name belongs to the class class_name. Let us again consider the class Number.

```
class Number
{
int x, y, z;

public:
void get_data(void);              //declaration
```

```
void maximum(void);              //declaration.
.
.
};
void Number :: get_data(void)
{
cout<< "\n Enter the value of fist number(x):"<<endl;
cin>>x;
cout<< "\n Enter the value of second number(y):"<<endl;
cin>>y;
cout<< "\n Enter the value of third number(z):"<<endl;
cin>>z;
}
void Number :: maximum(void)
{
int max;
max=x;
if (max<y)
max=y;
if (max<z)
max=z;
cout<<"\n Maximun value is ="<<max<<endl;
}
```

if you look at the above declaration of class Number, you can easily see that the member function get_data() and maiximun() are declared in the class. Thus, it is necessary that you have to define this function. You can also observe in the above snapshot of C++ program identity label (::) which are used in void Number :: get_data(void) and void Number ::maximum(void) tell the compiler the functionget_data() and maximum() belong to the class Number.

Now, let us see the complete C++ program to find out the minimum and maximum of three given integer numbers:

```
#include<iostream.h>
class Number
{
int x, y, z;

public:
void get_data(void);             //declaration
void maximum(void);              //declaration
void minimum(void)               //definition
{
int min;
min=x;
if (min>y)
min=y;
if (min>z)
min=z;
cout<<"\n Minimum value is ="<<min<<endl;
}
};
void Number :: get_data(void)
{
```

```
cout<< "\n Enter the value of fist number(x):"<<endl;
cin>>x;
cout<< "\n Enter the value of second number(y):"<<endl;
cin>>y;
cout<< "\n Enter the value of third number(z):"<<endl;
cin>>z;
}
void Number :: maximum(void)
{
int max;
max=x;
if (max<y)
max=y;
if (max<z)
max=z;
cout<<"\n Maximun value is ="<<max<<endl;
}

void main()
{
Number num;

num.get_data();
num.minimum();
num.maximum();
}
```

The output of the above program is given as follows:

Enter the value of the first number (x):
10
Enter the value of the second number (y):
20
Enter the value of the third number (z):
5
Minimum value is=5
Maximum value is=20

3.5.1 Nesting of Member Functions

In this section, we will discuss the nesting of member functions.

A member function of a class can call any other member function of its own class irrespective of its privilege. This is known as nesting of member functions.

Consider the problem of finding the largest of n given number which illustrates the above concept.

```
#include<iostream.h>
#define MAX_SIZE 100

class Data
{
int num[MAX_SIZE];
int n;
```

```
public:
void get_data(void);              //declaration
int largest(void);                //declaration
void display(void);               //declaration
};
void Data :: get_data(void)
{
cout<< "\n Enter the total numbers(n):"<<endl;
cin>>n;
cout<< "\n Enter the number:"<<endl;
for (int i=0;i<n; i++)
{
cout<< "\n Enter the numer"<<i+1<<": ";
cin>>num[i];
}
}
int Data :: largest(void)
{
int max;
max=num[0];
for(int i=1; i<n; i++)
{
if (max<num[i])
max=num[i];
}
return max;
}
void Data :: display(void)
{
cout<<"The largest number:"<<largest()<<endl;
}
void main()
{
Data num;

num.get_data();
num.display();
}
```

The class Data has the member function display having the statement cout<<"The largest number:"<<largest()<<endl;. It calls the member function largest () to compute the largest of n given numbers.

3.5.2 Passing Objects as Arguments

We can pass objects as arguments to a function like any other data type. This can be done by a pass-by-value and a pass-by-reference. In pass-by-value, a copy of the object is passed to the function and any modifications made to the object inside the function are not reflected in the object used to call the function. While, in pass-by-reference, an address of the object is passed to the function and any changes made to the object inside the function is reflected in the actual object. Furthermore, we can also return object from the function like any other data type.

Consider the following C++ program for addition and multiplication of two square matrices which illustrates the above concepts.

```
#include<iostream.h>
#define MAX_SIZE 10

int n;
class Matrix
{
int item[MAX_SIZE][MAX_SIZE];

public:
void get_matrix(void);
void display_matrix(void);
Matrix add(Matrix m);// Matrix object as argumentand as return: pass by value
void mul(Matrix &mat, Matrix m);// Matrix object as argument: pass by
reference and pass by value
};
void Matrix :: get_matrix(void)
{
cout<< "\n Enter the order of square matrix(nXn):"<<endl;
cin>>n;
cout<< "\n Enter the element of matrix:"<<endl;
for (int i=0;i<n; i++)
for (int j=0;j<n; j++)
cin>>item[i][j];
}
void Matrix :: display_matrix(void)
{
cout<<"\n The element of matrix is :"<<endl;
for (int i=0;i<n; i++)
{
for (int j=0;j<n; j++)
cout<<item[i][j]<<"\t";
cout<<endl;
}
}
Matrix Matrix :: add(Matrix m)
{
Matrix temp;     // object temp of Matrix class

for (int i=0;i<n; i++)
for (int j=0;j<n; j++)
temp.item[i][j]=item[i][j]+m.item[i][j];
return (temp);    // returm matrix object
}
void Matrix :: mul(Matrix &rm, Matrix m)
{
for (int i=0;i<n; i++)
for (int j=0;j<n; j++)
{
rm.item[i][j]=0;
for(int k=0; k<n; k++)
rm.item[i][j]=rm.item[i][j]+item[i][k]*m.item[k][j];
}
}
void main()
{
```

```
Matrix X, Y, Result;
cout<<"Matrix X :"<<endl;
X.get_matrix();
cout<<"Matrix Y :"<<endl;
Y.get_matrix();
cout<<"\n Addition of X & Y :"<<endl;
Result=X.add(Y);
Result.display_matrix();
cout<<"\n Multiplication of X & Y :"<<endl;
X.mul(Result,Y);          //result=X*Y
Result.display_matrix();
}
```

If you look at the above program, you can observe that in main(), the statement
Result=X.add(Y);
in which X.add(Y) invokes the member function add() of the class matrix by the object X with the object Y as arguments. The members of Y can be accessed only by using the dot operator (like m.item[i][j]) within the add() member. Any modification made to the data members of the object Y is not visible to the caller's actual parameter. The data members of X are accessed without the use of the class member access operator. The statement in the function add(),
return(temp);
returns the object temp as a return object. The result has become the return object temp which stores the sum of X and Y. This illustrates that function also return object.
Further, in main(), the statement
X.mul(Result, Y);
transfers the object result by reference and Y by value to the member function mul(). When mul() is invoked with result and Y the objects parameters, the data members of X are accessed without the use of the class member access operator, while the data members of result and Y are accessed by using their names in association with the name of the object to which they belong. Modifications which are carried out on result object in the called function will also be reflected in the calling function.

☞ Check Your Progress 2

1) What is the scope resolution operator?

……

……

……

2) When would we define a member function inside or outside of the class?

……

……

……

3) What is the purpose of the member function?

……

……

……

4) Define a class to represent a bank account. Include the following members:

Data Members:

a. Name of the depositor
b. Account Number
c. Type of Account
d. Balance amount in the account

……

……

……

Member Functions:

a. To assign initial value
b. To deposit an amount
c. To withdraw an amount after checking the balance
d. To display name and balance and account Number

……

……

……

5) Write a interactive program in C++ for the above problem. Assume a bank has maintained two types of account: Saving account and Current account. You should open a saving account with minimum Rs 500 and also keep minimum balance of Rs. 500 and current account with Rs. 5000 and also keep minimum balance of Rs. 5000.

……

……

……

3.6 FRIEND FUNCTION

As we have discussed that the private members cannot be accessed from outside the class. It implies that a non-member function cannot have an access to the private data of a class. Let us suppose, we want a function operate on objects of two different classes. In such situations, C++ provides the friend function which is used to access the private members of a class. Friend function is not a member of any class. So, it is defined without scope resolution operator. The syntax of declaring friend function is given below:

```
class class_name
{
…
…
public:
…
…
friend return type function_name(arguments);
}
```

Let us consider the complete C++ program to find out sum of n given numbers to understand the concept of friend function.

```
#include<iostream.h>
#define MAX_SIZE 100
class Sum
{
int num[MAX_SIZE];
int n;
public:
void get_number(void);
friend int add(void);
};
void Sum :: get_number(void)
{
cout<< "\n Enter the total number(n):"<<endl;
cin>>n;
cout<< "\n Enter the number:"<<endl;
for (int i=0;i<n; i++)
cin>>num[i];
}
int add(void)
{
Sum s;
int temp=0;
s.get_number();
for (int i=0;i<s.n; i++)
temp+=s.num[i];
return temp;
}

void main()
{
int res;
res=add();
cout<<"The sum of n value is="<<res<<endl;
}
```

If you look at the above program, you can easily see that the function add is declared as a friend function of class Sum. The add function accesses the private data, adds the numbers of array and returns value to the main function where it is called upon. Furthermore, you can also see that friend function add() is defined without scope resolution operator(::), because it does not belong to a class.

Now, let us consider a situation in which we want to operate on objects of two different classes. In such a situation, friend functions can be used to bridge the two classes.

```
#include<iostream.h>
class Two;        //forward declaration like function prototype
class One
{
int a;
public:
void get_a(void);
```

```
friend int min(One, Two);
};
class Two
{
int b;
public:
void get_b(void);
friend int min(One, Two);
};
void One :: get_a(void)
{
cout<<"Enter the value of a:"<<endl;
cin>>a;
}
void Two :: get_b(void)
{
cout<<"Enter the value of b:"<<endl;
cin>>b;
}
int min (One o, Two t)
{
if(o.a<t.b)
return o.a;
else
return t.b;
}
void main()
{
One one;
Two two;
int minvalue;

one.get_a();
two.get_b();
minvalue=min(one,two);
cout<<"Minimum="<<minvalue<<endl;
}
```

You can observe that the above program contains two classes named one and two. The function min() is declared in the both the classes with the keyword friend. An object of each class has been passed as an argument to the function min (). Being a friend function, it can access the private members of both classes through these arguments. Now, let us note some special properties possessed by friend function:

(i) A friend function is not in the scope of the class to which it has been declared as friend.
(ii) A friend function cannot be called using the object of that class. It can be invoked like a normal function without the use of any object.
(iii) Unlike member functions, it can not access the members directly. However, it can use the object and dot membership operator with each member name to access both private and public members.
(iv) It can be declared either in the public or the private part of a class without affecting its meaning.
(v) Generally, it has got objects as arguments.

3.7 STATIC MEMBERS

In this section, we will discuss the static members.

The members of a class can be made static (data member and function member both). Can you tell, what is static? Well, static is a storage class specifier/qualifier that provides information about locality and visibility of variables. Let us discuss the static data member. Sometimes, we need to have one or more common data member, which are accessible to all objects of a class. For example, we need to keep the status as to how many objects of a class are created and how many of them are currently active in the program. In such situation, and many others, C++ provides static data member. Static data member is initialized to zero when the first object of its class is created. No other initialization is permitted.

A variable that is part of a class, yet is not the part of an object of that class, is called a static data member. There is exactly one copy of static data member instead of one copy per object, as for ordinary non-static data members. Similarly, a function that needs access to members of a class, yet doesn't need to be invoked for a particular object is called a static member function. The common syntax of defining static data member is given as follows:

```
class class_name
{
…..
….
static data_type data member;
….
….
};
data_type class_name :: data member=initial_value;
```

Static data member must be defined outside the class. Initialization of static data member is optional. Let us consider the following C++ program to understand the concept of static data members.

```
#include<iostream.h>
class Counter
{
static int count;  // static member: count is static
static int n;        // static member: number is static
public:
void get_data(int num)   // initializes object's member
{
n=num;
count++;
}
void show_count(void)
{
cout<<"Number of times calls made to function 'get_data()' through any object:";
cout<<count<<endl;
cout<<"n:"<<n<<endl;
}
};
int Counter :: count=0;   // definition and initialization(optional) of a static data member
```

```
int Counter :: n;              // definition of static data member
void main()
{
Counter x,y,z;
x.show_count();
y.show_count();
z.show_count();

x.get_data(25);
y.get_data(50);
z.get_data(75);
cout<<"After reading data :"<<endl;
x.show_count();
y.show_count();
z.show_count();
}
```

The above program shows the concept of static data member. There are two static variables **count** and **n**. The variables **count** and **n** are initialized zero when their object is created. The static data member **count** is incremented and data **n** is assigned with parameter value. Whenever the get_data() function is called. Since get_data() member function is called three times by the object **x**, **y**, and **z**, the data member count is incremented three times. Furthermore, the variable n is assigned with values 25, 50 and 75 respectively by each function call get_data(). All the statements print the value of **count** as 3 and **n** as a 75 because there is only one copy of **count** and one copy of **n** which are shared by all three objects.

Now, let us discuss about static member function. Like static data member variable, we can also have a static member function. The static function can only access the static member(data or function) declared in the same class. There is one important difference between static member function and member function. A static member function is called using the class name instead of its object. The following program illustrates the concept of the static member function:

```
#include<iostream.h>
class Counter
{
static int count;  // static member: count is static
 int n;    // static member: number is static
public:
void set_data(void)        // initializes object's member
{
count++;
n=count;
}
void show_value_of_n(void)
{
cout<< "The value of n is ="<<n<<endl;
}
void static show_count(void)
{
cout<<"Count :"<<count<<endl;
}
};
int Counter :: count=0;   // definition and initialization(optional) of a data
member
```

```
void main()
{
Counter c1,c2;
Counter::show_count();
c1.set_data();
c2.set_data();
Counter::show_count();
Counter c3;
c3.set_data();
Counter::show_count();
c1.show_value_of_n();
c2.show_value_of_n();
c3.show_value_of_n();
}
```

If you look at the above program, you can easily see that the static function show count() displays the number of objects created till that moment. A count on the number of objects is maintained by static variable count. The function show_value() displays the value of the non-static variable n.

☞ Check Your Progress 3

1) What is friend class? Write a program to illustrate the concept of friend class.

…..

……

……

2) Why is friend function needed?

……

……

……

3) Discuss memory requirements for classes, objects, data members and member functions.

……

……

……

4) Bring out the differences between the memory requirements of static data members and non-static data members.

……

……

……

3.8 SUMMARY

In this unit, we have seen and discussed the concept of class as well as object which are the basic components used in C++ programming. Class declaration and object creation have been discussed and illustrated with examples. The members viz., data members and function members of a class, defining member functions were explained and elaborated. We have seen the way to pass objects as arguments to the functions with call by value and call by reference. Furthermore, static members, Friend function and Friend class are also discussed with examples.

3.9 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

1. Structures and classes in C++ are given same set of features. A class in C++ is identical to structure in C++. In C++, the difference between structures and classes is that, by default, structure members have public accessibility while class members have private access control unless otherwise explicitly stated.

2. Classification is a concept/technique which is used to make group of objects or partition objects by some logic and classify them into bigger entities i.e. class.

3. Though the main reason for using a class is to encapsulate data and code. It is, however, possible to have a class that has neither data nor code. In other words, it is possible to have empty classes. The declaration of empty classes is as follows:

```
    class ABC{};
    class Employee{};
class xyz
{
};
```

During the initial stage of development of a project, some of the class are either not fully identified, or not fully implemented. In such cases, they are implemented as empty classes.

4.

```
#include<iostream.h>
#define MAX_SIZE 25
class Sum
{
int number[MAX_SIZE];
int n;
int total;
public:

void get_data(void)
{
int i;
cout<<"Enter Total Number :"<<endl;
cin>>n;
cout<<"Enter Number One by One:"<<endl;
for(i=0; i<n; i++)
{
cout<<"Enter Number"<<i+1<<":"<<endl;
cin>>number[i];
```

```
}
}
void cal_sum(void)
{
total=0;
int i;
for (i=0; i<n; i++)
total=total+number[i];
}
void display_sum(void)
{
cout<<"\nSum :"<<total<<endl;
}
};
void main()
{
Sum s;
s.get_data();
s.cal_sum();
s.display_sum();
}
```

Check Your Progress 2

1. The scope resolution operator (::) especially defined in C++. They are used in two ways: (1) for resolving the class member function (2) for resolving the global variable.

2. If your function is very small and make this inline, then you can define your function in the class because all defined function within class are, by default, inline functions. If you are using much control statements and looping, you have to define your function outside the class.

3. A member function performs an operation required by the class. It is the actual interface to initiate an action of an object belonging to a class. It may be used to read, manipulate, or display the data member. The data member of a class must be declared within the body of the class, while the member functions of a class can be defined in two places:
   (i) inside the class definition
   (ii) outside the class definition

4.

```
#include<iostream.h>
#include<stdlib.h>
#define MAX_SIZE 25

// unit3e3.cpp

class Account
{
char depositor_name[25];
int account_no;
int type_of_account;
int balance;
public:
```

```
void assign_initial_value(void);
void deposit(void);
void withdraw(void);
void account_info(void);
};
void Account :: assign_initial_value(void)
{
cout<< "\n Enter the depositor name:"<<endl;
cin>>depositor_name;
cout<< "\n Enter the account no.:"<<endl;
cin>>account_no;
cout<< "\n Enter the type of account(1-for saving, 2-for current):"<<endl;
cin>>type_of_account;
cout<< "\n Enter the amount to be deposited(For saving min:500 & current min:5000"<<endl;
cin>>balance;
}
void Account :: deposit(void)
{
int da;
cout<< "\n Enter the amount to be deposited:"<<endl;
cin>>da;
balance=balance+da;
}
void Account :: withdraw(void)
{
int aw;
int d;

cout<< "\n Enter the amount to be withdraw:"<<endl;
cin>>aw;
d=balance-aw;
if((type_of_account==1) && (d>=500))
{
balance=balance-aw;
cout<<"\n Withdraw Successful:"<<endl;
}
if((type_of_account==1) && (d<500))
{
cout<<"\n Withdraw UnSuccessful, check your balance:"<<endl;
}
if((type_of_account==2) && (d>=5000))
{
balance=balance-aw;
cout<<"\n Withdraw Successful:"<<endl;
}
if((type_of_account==2) && (d<5000))
{
cout<<"\n Withdraw UnSuccessful, check your balance:"<<endl;
}
}
void Account :: account_info(void)
{

cout<< "\n depositor name:"<<depositor_name<<endl;
cout<< "\n Account no.:"<<account_no<<endl;
cout<< "\n Balance:"<<balance<<endl;
```

```
}
int main()
{
Account a;
int choice;
while(1)
{
cout<< "\n Account operation:..."<<endl;
cout<< "\n 1. Assign initial value";
cout<< "\n2. Deposit Ammount";
cout<< "\n3. Withdraw Amount";
cout<< "\n4. Balance Enquiry";
cout<< "\n5. Quit";
cout<< "\n Enter choice :"<<endl;
cin>>choice;
switch(choice)
{
case 1: a.assign_initial_value();
break;
case 2: a.deposit();
break;
case 3: a.withdraw();
break;
case 4: a.account_info();
break;
case 5: exit(1);
break;
default: cout<<"Bad option selected";
 break;
}
}
}
```

Check Your Progress 3

1. We can define a class as friend of another class, granting that second class access to the protected and private members of the first one. A class declared as a friend of another class, it possesses the rights of access to the private member of this class using objects.

```
#include<iostream.h>
class X
{
int x;
public:
void get_x(void)
{
cout <<" Enter the value of x:"<<endl;
cin>>x;
}
friend class Y;
};
class Y
{
int y;
public:
```

```
void get_y(void)
{
cout <<" Enter the value of y:"<<endl;
cin>>y;
}
void add(void)
{
X objx;
objx.get_x();
get_y();
y=y+objx.x;
}
void display()
{
cout<<"The Sum="<<y;
}
};

void main()
{
Y objy;
objy.add();
objy.display();
}
```

2. There are the following situations in which the friend function needed.
   - Function operating on objects of two different classes.
   - Friend functions can be used to increase the versatility of overloaded operators.
   - Sometimes, a friend allows a more obvious syntax for calling a function rather than what a member function can do.

3. When a class is declared, memory is not allocated to data members but allocated to only member functions. When an object of a particular class is created, memory is allocated only to its data members. Thus, all objects of that class have access to the same area in the memory where the member functions are stored. It is logically also true as the member functions are same for all objects and there is no need to allocate a separate copy for each and every object created. However, storage space for data member is allocated for every object's data members. This is essential because data member will hold different data values for different objects.

4. Whenever a class is instantiated, the memory is allocated for the created objects. Memory space for static data members is allocated only once during class declaration while memory space of on-static data members is allocated when objects of a class are created. Therefore, all objects of the class have access the same memory area allocated to static data members. When one of them modifies the static data member, the effect is visible to all the instance of the class i.e. objects.

3.10 FURTHER READINGS

1) B. Stroustrup, *The C++ Programming Language*, third edition, Pearson/Addison-wesley Publication,1997.

2) K. R. Venu Gopal, Raj Kumar Buyya, T Ravishankar, *Mastering C++*, Tata-McGraw-Hill Publishing Company Limited, New Delhi.

3) E. Balagurusamy, *Object Oriented Programming with C++*, Tata Mc-Graw-Hill Publishing Company Limited, New Delhi.

4) N. Barkakati, *Object Oriented Programming in C++*, Prentice-Hall of India.