UNIT 2 CLASSES AND OBJECTS

Structure **Page Nos.**

2.0 INTRODUCTION

In the previous Unit, we discussed different programming paradigms as well as the syntax of various C++ statements. We also learnt how to make our own library. In this Unit, we will discuss Class, as important Data Structure of C++. A Class is the backbone of Object-Oriented Computing. It is an abstract data type. We can declare and define data as well as functions in a class. An object is a replica of the class to the exception that it has its own name. A class is a data type and an object is a variable of that type.

2.1 OBJECTIVES

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

- define the basic concept of a class;
- explain private and public clauses;
- create objects; and
- write programs with functions taking objects as arguments.

2.2 DEFINITION AND DECLARATION OF A CLASS

A class is a user-defined type. The definition of a class includes declaring a data object of that type, specifying the data items as well as functions which operate on these data items, indicating if the data can be accessed by functions which are out of scope of class by indicating “public or private” and other details.

We shall consider the class “Queue” as an example:

```
class Queue
{
        int front, rear;
        int queue_array [10];
public:
        int isqueueempty ( );
        int isqueuefull ( );
        void insert(int);
        void delete ( );
        void print ( );
```

};

In the above Class front, rear, queue/array are data members. Whenever we define and declare a class, all the data as well as member functions are "private" by default.

"Private" means that they can be accessed only by the functions within the class. These functions are known as **Member functions.**

"Public" means that they can be accessed by the functions which are outside the class also. Since everything in a class is Private by default, we can specifically indicate that something is public as we have done in the case of above member functions.

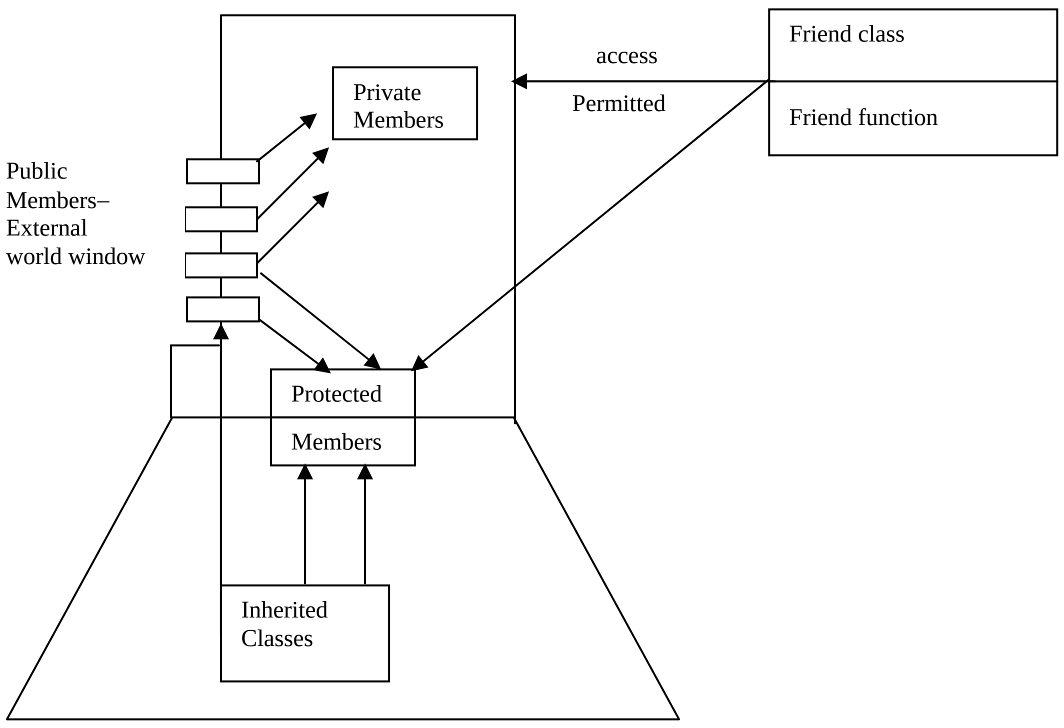

Figure 1: Public and Private Members of a class

So, in the above example, the Private members front, rear, queue_array can be accessed only by the five member functions. (Anyway we can override this. We will discuss about it later).

The ability to declare the members of a class as private gives rise to concept of "data hiding".

Encapsulation is the mechanism that binds together code and the data. It manipulates and keeps both safe from outside interference and misuse. The concept of class in C++ supports encapsulation by enabling both data as well as functions to be declared and defined in the same class and declaration of all members as "private" by default.

We can declare arrays within a class just like any other data item. Queue array in the class "Queue" is an example.

2.3 SCOPE RESOLUTION OPERATION

The scope resolution operator is denoted by "::".

A class is identified by the external world through the public member functions, constructor and destructors, which is also referred to as the interface of a class. The interface of class in general needs to specify the names of member functions and the parameters to be passed to values to be returned. Therefore, in general the interface of the class is kept separately from the implementation of its member functions. One of the key reasons for that is a user of a class need not know the implementation details, it just need to know how a class can be used.

Since implementation/definition may be defined separately, therefore, we need a mechanism to identify with what class a function implementation is associated. The scope resolution operator specify the scope of a function that is to what class it is associated.

For example, any function of the Queue class in Section 2.2 may be implemented as:

```
Queue:: isqueueempty ( )
         {
         ...
         ...
         }
```

The scope resolution operator can also be used for defining the global variables.

Please note here that the functions can also be defined inline, that is, during the class definition itself.

Consider the following example:

```
#include < iostream.h >
int x;
void main ( )
{
        f ( );
}
void f ( )
{
        int x; // hides the Global x
        x = 1; // assigns to local x
        // if we want to make any assignments to global x in f ( ), it is not
        // possible
}
```

But with the help of scope resolution operator "::", we can do that as follows:

```
void f ( )
{
        int x;
        x = 1; // assigns to local x
        :: x = 2; // assigns to global x
}
```

So, whenever a variable is prefixed by :: it refers to Global variable.

So, with the help of Scope Resolution operator, we can use a hidden global name.

2.4 PRIVATE AND PUBLIC MEMBER FUNCTIONS

All the member functions, which are declared in a class, are private by default unless specifically indicated as public (refer to Queue class where all the member functions are declared public).

Other member functions of the same class and friends of it can only call all private functions. Private functions, by themselves cannot be invoked. (Refer to *Figure 1*). For example,

```
class one
{
        int x, y;
        one ( )
        //one ( ) is a private function by default.
        {
        x = 1;
        y = 2;
}
void print ( )
{
        cout << x<<y;
}
};

one x1; // wrong
        // the procedure one ( ) cannot be executed as it is private.
```

If we change it to public, then it will be executed.

Consider the following:

```
class one
{
int x,y;
void print ( )
{
        cout <<x<<y;
}
public:
one ( )
{
x = 1;
y = 2;
}
};
one x1;
void main ( )
{....}
```

Now, one ( ) will be executed. But, any call such as x1.print ( ) is invalid as it is a private member function. So, we can invoke it from some other member function of the same class follows:

```
class one
{
int x,y;
void print ( )
{
cout <<x<<y;
}
public:
one ( )
{
x =1;
y=2;
print ( ); // valid call to print within one ( )
}
};

one x1;
void main ( )
{
}
```

Also, a private member function can be called from another private member function. Anyway, the first member function, which starts the chain **should be** a public member function.

2.5 CREATING OBJECTS

In this Section, let us discuss about various issues relating to creation of objects.

Object initialisation

A class can have many member functions. One of these member functions allows proper initialization of object at the time of creation. This function is called a constructor. A Constructor has the same name of a class and initialises the data members of an object of that class. It has no return type.

For example:

```
        class complex {
            int real;
            int imag;
        public:
            complex (int, int);
        }
        complex::complex (int x=0, int y=0)
        {real=x; //x=0 specifies default value
        imag = y;
        };
main ( ) {
        complex a(5,5), b(5), c;}
```

In the example above, three complex objects will be created as:

Object Name	Real part	Imaginary part
a	5	5
b	5	0
c	0	0

If a class has a constructor, then it is called whenever an object of that class is created, otherwise a default constructor for the class is called. If the class has the destructor, then it is called whenever an object of that class is to be destroyed otherwise a default destructor for the class is called.

Object creation: We can create objects in the following ways. We can create an automatic object, which is created each time its declaration is encountered during the execution of the program and is destroyed each time the block in which it occurs is left. For example,

```
class example1
{
        .
        .
        .
        .
};
void main ( )
{
f( )
}

void f ( )
{
example1 ex1;
.
.
.
.
}
```

Now, in the above example, the object ex1 is created whenever the control is transferred to f () and the statement declaring ex1 is executed. ex1 is destroyed, once the execution of f () is complete. Here ex1 is automatic object. Please note that in this example no constructor of the object is defined, therefore, it will use default constructor and destructor generated by the compiler itself.

We can create a static object which is created once during the start of the program and is destroyed once the execution of the program is completed.

Consider the following example:

```
class example
{
        .
        .
        //
        .
        .
};
void main ( )
{
        f ( );
        f ( );
        // last statement
}
void f ( )
{
static example ex1;
.
.
.
}
```

Now, ex1 is created once during the first call to f ( ). It is not destroyed after the completion of execution of f ( ) since it is **static object**. Also, since it is not created once again, the data stored in it which was present at the end of the previous call is not lost when f ( ) is called again. It is destroyed only once and that is done after the completion of execution of the program. In this case, object will still be created by default constructor, however, it will be run only once during the lifetime of the object.

We can create an object on the free store using new operator and destroy it using delete operator. Consider the following example:

```
Class example1
{
.
.
.
.
};
main ( )
{
example1*p=new example1;
delete p;
}
```

P is the pointer to an object of type example1. The object as well as the pointer, which points to it, it destroyed when delete is executed. A user can design his own new and delete operators.

An object can also be created as a member of another class. Consider the following example:

```
Class example1
{
            .
            .
            .
            .
};

Class example2
{
example1 ex1;//ex1 is an object which is member of another class
            .
            .
            .
            .
};
```

Whenever an object of example2 is created and if there were constructors for both or either without taking arguments, then the constructor of object of example1 is executed followed by the execution of constructor of example2.

If there were arguments to one or both the constructors then they have to be explicitly called. The same is the case with destruction. Whenever an object of example2 has to be destroyed, the object of example1 is destroyed followed by the destruction of object of example2.

An object can also be created as an array element. Consider the following example:

```
Class ex
{
.
.
.
};
ex ten [10];
```

If the ex contains constructor, then there should be a default constructor, which need not be called with a parameter. The reason is that there is no way to specify an argument along with an array.

☞ Check Your Progress 1

1) Define a string data type with the following functionality:
   - A constructor having no parameters,
   - Constructors which initialise strings as follows:
     - A constructor that creates a string of specific size
     - Constructor that initialises using a pointer string

- A copy constructor

- Define the destructor for the class
- It has overloaded operators. (This part of question will be taken up in the later units).
- There is operation for finding length of the string.

……………………………………………………………………………………………………
.

……………………………………………………………………………………………………
.

……………………………………………………………………………………………………
.

2.6 ACCESSING CLASS DATA MEMBERS AND MEMBER FUNCTIONS

If a data member is public, then it can be accessed by the following syntax:

objectname.datamember

If the data member is private then it can be accessed by the member functions only.

Any public member function can be a accessed as follows:

object_name.member function_name

If the member function is private, then it cannot be accessed as mentioned for public member functions. A private member function can be accessed only through another member function. (Please refer to *Figure 1*).

Consider the following example which illustrates the above concepts. The following is a linear search program which will receive a key value as well as a list of integers and gives the position of the key value in the list if it is present.

```
#include < iostream.h >
class data
{
        int list [10];
        int size;
        public:
        data ( ) // constructor of class data
        {
                cout << “size of list is atmost 10” <<“Enter the size of the list;
                cin >> size; // Accessing private data
                for ( int i = 0; i<size;++i)
                {
                        cout<<“Enter the element”;
                        cin>> list [i];
                }
        }// end of constructor
void search (int p)
{
        int check = 1;
        int i = 0;
        while ((check)&&(i<size))
        //Check is not equal to 0 and 1 is less than size
        {
```

```
                        if (p= =list[i])
                        check = 0;
                        else
                        ++ i ;
            }

            if (check = = 0) // key is found as check is assigned to zero value
                        cout << “key matched at the following position” << ++i;
            else
                        cout << “No matching element found”;
            }
};// end of the class data
void main ( )
{           //Calls the constructor defined in the process
            data d1;
            int key;
            cout << “Enter the key value”;
            cin>>key;
            d1.search(key); // Accessing member function. Notice the way member
                                    // function has been
            // Called by the object d1 which is of class data.
}
```

2.7 ARRAYS OF OBJECTS

Consider the following class:

```
class C1
{
.
.
};
```

Now, an array of objects of C1 can be declared as C1 c [10];

So, we have declared 10 objects of type C1. Also, we can create an array of objects C1 on the free store as follows:

```
C1  c = new C1[10];
```

If C1 is having a constructor, then it should have a default argument or no arguments. During the execution of above statement, each constructor belonging to each object is executed in sequence one after another.

The way of accessing data members or member functions of the objects are similar to the method we have outlined in the previous sections except that the prefix of data member as well as member function name will be
Array_name{index}.

So, if you are accessing data item (say, i assuming that i is public) of object c[3],then it is indicated as c [3].i

When we are creating objects on free store, we can delete them as follows:

```
Delete [  ] c; //Since c is an Array
Delete X; //if X is  single object.
```

2.8 FRIENDS

As we discussed previously, any private data of a class can be accessed by only its member functions. But, any other function, which is not a part of class, can also access private data provided it was declared as a friend of the class whose private data is to be accessed.

For example,

```
Class x {
{         int i;
          char j;
          public:
                    int modify ( )
                    {         //
                              .
                              .
                              .
                              .
                              //
                    }
friend void check ( ) ;
};
void check ( )
{         if (x.i < 0)// no error
          {         //
                    .
                    .
                    .
                    .
          }
}
```

Now, the function “check” is able to access the private data of class x since it has been declared as a friend of the class x. Similarly, we can also declare a member function of a class to be a friend of another class.

2.9 OBJECTS AS FUNCTION ARGUMENTS

We can pass objects as arguments to a function. We can send them by reference or by value.

Consider the problems of adding two matrices:

This can be done by sending objects by reference as follows:

```
#include < iostream.h >
#define MAXROW 50
#define MAXCOL 50
int row, col;
class matrix
{
int m [MAXROW][MAXCOL];
public:
          void in_element (void);
          void print (void);
friend matrix add (const matrix&, const matrix&);
// The addition will be carried out by a friend function.
// This function however will not change the value of its arguments,
// hence const been used before each argument
};
void matrix::in_element ( )
//please note the use of scope resolution operator.

{         cout << “the matrix is:”;
          for (int i=0; i< row; + + i)
```

```
                for(int j = 0; j<col; ++j)
                cin>> m[i] [j];
}
void matrix::print ( )
{       cout <<  "the matrix is:";
        for (int i = 0; i<row; ++i)
        {       cout << "\n";
                for (int j = 0; j< col; ++j) //Print the matrix in tabular form.
                        cout << m[i] [j]<<"\t";
                }
}
```

In the function prototype, matrix add (const matrix&, const matrix&); the matrix& is a reference to matrix class. Please note this represent call by reference and not call by value. The advantage of call by reference here is that since the matrix is a large object, its instance will not be duplicated as is the case in call by value. Moreover, const ensures that original values of matrix does not get modified even by mistake in programming. Such mistakes will be caught by compiler. Thus, this method of parameter passing may be considered for large objects.

```
matrix first, second;
void main ( )
        {
        cout << "enter the  order of the matrix:";
        cin >> row;
        cin >> col;
        first.in_element ( );
        second.in_element ( );
        matrix result = add (first, second);
        result.print ( );
}
matrix add (const matrix& one, const matrix& two)
{
        matrix temp_result;
        for (int k=0; k<row; ++k)
                for (int 1= 0; 1< col; ++1)
                        temp_result.m[k][1] = one.m[k][1] +two.m[k][1];
return temp_result;
}
```

The same can be done by sending objects by value as follows:

```
//demonstration of call of objects by value.

#include<iostream.h>
#define MAXROW 50
#define MAXCOL 50
int row, col;
class matrix
{
int m[MAXROW][MAXCOL];
public:
        void in_element (void);
        void print (void);
friend matrix add (const matrix&, const matrix&);
};
void matrix::in_element
{       cout <<"the matrix is:";
        for (int i =0; i<row; ++i)
                for (int j=0; j<col; ++j)
                        cin >> m[i][j];
}

void matrix::print ( )
{           cout << “the matrix is:”;
            for (int i =0; i<row; ++i)
            {           cout << “\n”;
                        for (int j=0; j<col; ++j)
                                    cout << m[i][j] << ‘‘\t”;
            }
}

matrix add (matrix, matrix);
matrix first, second;
void main ()
{
            cout << “enter the order of the matrix:”;
            cin >> row;
            cin >>col;
            first.in_element ( ) ;
            second.in_element ( ) ;
            matrix result = add(first, second);
            result.print ( );
}
matrix add (matrix one, matrix two)
{
            matrix temp_result;
            for(int k = 0; k<row; ++k)
                        for (int l=0; l<col; ++l)
                                    temp_result.m[k][l] = one.m[k][l]+two.m[k] [l];
return temp_result;
}
```

☞ **Check Your Progress 2**

1) Write a program for the implementation of stack using classes.

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

2) Write a program for the multiplication of matrix of order m × n with a vector of order m×l classes. Design a function, which accepts matrix and vector as arguments and returns the resultant matrix.

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

2.10 SUMMARY

In this Unit, we have discussed the concept of class, its declaration and definition. It also explained the ways for creating objects, accessing the data members of the class. We have seen the way to pass objects as arguments to the functions with call by value and call by reference.

2.11 SOLUTIONS/ANSWERS

Check Your Progress 1

1)
```
const maxlen = 20;
class string
{private:
        char * str;
        int length;
Public:
        //constructors
        string ( ); //defines a string of length of 0 but of a default size 20.
        string (int i_len); // create a blank string of size i_len
        string (const char *s);
        int strlength (const string &s);
        //:other string functions
        ~string ( );  // destructor
        };

        string:: string ( )
        {
                str = new char [maxlen];
                length = 0;
                str [0]= '\0';
        }

        string:: string (int i_len)
        {
                length = i_len;
                str = new char [i_len];
                int i =0;
                for (i=0; i < i_len; i++) str [0] =' ';
                str [i] = '\0';
        }

        string::string (const char *s)
        //constructor with initialization using a constant string
        {
                length = strlen(s);
                str = new char [length + 1];
                strcpy (str, s);
                //strlen & strcpy are library string functions
        }

        string:: string (const string &s) // copy constructor
        {
                 length = s.length;
                str = new char [length + 1];
                strcpy (str, s.str);
                // create a new instance of a string
        }
        string::length (void) const
        {
                return length;
        }
        string::~string ()
        {
        delete str;
        }
```

Check Your Progress 2

1) **Hint:** Use array as data structure for implementing the stack and for pushing and popping elements on the stack.

2) **Hint:** The following should be provided in your program:

- A Class for Matrix
- A Class for Vector (one dimensional array)
- A function that accepts, Matrix and Vector as arguments and returns Matrix or Vector as per the type of expected result.