CHAPTER 6

CLASSES AND OBJECTS

6.1 Introduction to Classes

The most important feature of C++ is the “Class”. Its significance is highlighted by the fact that Bjarne Stroustrup initially gave the name ‘C with Classes ‘. A class is a new way of creating and implementing a user defined data type. Classes provide a method for packing together data of different types. For Example:

```
class student
{
        char name[30];
        int rollno, marks1, marks2, total_marks;
};
```

The data variables, rollno, marks1, marks2, total_marks define the properties or features of a student ,thus packing together data of different types. The class data type can be further extended by defining its associated functions. These functions are also called as methods, as they define the various operations (in terms of accepting and manipulating data) that can be performed on the data.

In other words

✓ A class is a way to bind the data and its associated functions together

6.2 Specifying a class :

A class specification has two parts :

1) Class declaration
2) Class Function Definitions

```
// Program - 6.1
# include <iostream.h>
# include <conio.h>
class student
{
       private :
         char name[30];
         int rollno, marks1, marks2 ,total_marks;
       protected:
          void accept()
          {
               cout<<"\n Enter data name, roll no, marks 1 and
marks 2.. ";
               cin>>name>>rollno>>marks1>>marks2;
          }
          void compute()
          {
               total_marks = marks1+ marks2;
          }
          void display()
          {
                      cout<<"\n Name "<<name;
                      cout<<"\n Roll no "<<rollno;
                      cout<<"\n Marks 1.. "<<marks1;
                      cout<<"\n Marks 2.. "<<marks2;
                      cout<<"\n Total Marks.. "<< total_marks;
          }
        public:
        student()
        {
               name[0]='\0';
               rollno=marks1=marks2=total_marks= 0;
               cout<<"\n Constructor executed ...  ";
         }
         void execute()
         {
               accept();
               compute();
               display();
         }
};
void main()
{
       clrscr();
       student stud;
       stud.execute();
}
```

The form of class declaration is

General Form

```
class class-name
{
 private:
    variable declaration
    function declaration

 protected:
    variable decl.
    function decl.

 public:
     variable decl.
     function decl.
};
```

With respect to the above example

```
class student
{ private;
   char name [10];
   int roll no, mark1, mark2, total marks;

protected:
   void accept( );
   void compute( );
   void display( );

public:
student( );
void execute( );
};
```

- ✓ The keyword class specifies user defined data type class name
- ✓ The body of a class is enclosed within braces and is terminated by a semicolon
- ✓ The class body contains the declaration of variables and functions
- ✓ The class body has three access specifiers (visibility labels) viz., private , public and protected
- ✓ Specifying private visibility label is optional. By default the members will be treated as private if a visibility label is not mentioned
- ✓ The members that have been declared as private, can be accessed only from within the class
- ✓ The members that have been declared as protected can be accessed from within the class, and the members of the inherited classes.
- ✓ The members that have been declared as public can be accessed from outside the class also

6.3 Data Abstraction

The binding of data and functions together into a single entity is referred to as **encapsulation.**

The members and functions declared under private are not accessible by members outside the class, this is referred to as **data hiding**. Instruments allowing only selected access of components to objects and to members of other classes is called as **Data Abstraction.** Or rather Data abstraction is achieved through data hiding.

Data hiding is the key feature of object oriented programming (OOPS)

private	Accessible by only its own members and certain special functions called as **friend functions**
protected	Accessible by members of inherited classes
public	Access allowed by other members in addition to class member and objects

6.4 Data Members and Member Functions

Class comprises of members. Members are further 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. Now look at the Table 6.1 where information is provided based on Program –6. 1 **class student. Classes** include special member functions called as constructors and destructors. These will be dealt in Chapter – 8 Constructors and Destructors.

student	The data type identifier **student** is also called as class tag
name, rollno, marks1, marks2, total	data members
public accept ( ) compute ( ) display ( ) execute ( ) student ( )	member functions or methods
stud	instance/object/variable of class student

Table 6.1 Class Student of Program - 6.1

6.5 Creating Objects

Look at the following declaration statement **student stud;** This statement may be read as **stud is an instance or object of the class student.**

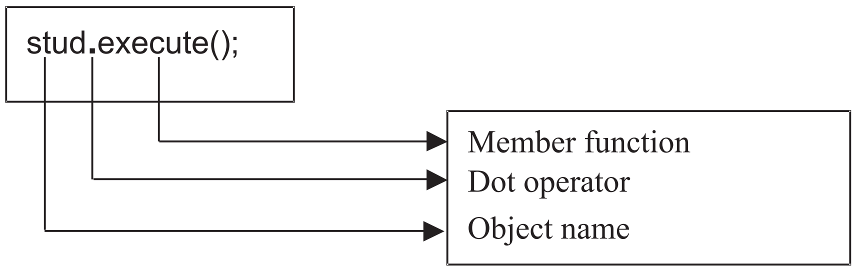
Once a class has been declared, variables of that type can be declared. 'stud' is a variable of type student ,student is a data type of class . In C++ the class variables are known as objects. The declaration of an object is similar to that of a variable of any basic type. Objects can also be created by placing their names immediately after the closing brace of the class declaration.

```
class student
{
        private:


          protected:


          public:

}stud;// stud is an object
```

```
class student
{
        private:
        ........
        protected:
        .......
         public:
};
void main()
{
  student s, s1[5];
}
the variables s and s1 are objects
or instances of the class stduent
```

6.6 Accessing Class Members

The members of a class are accessed using the dot operator. For example, the call statement to the function execute() of the class student may be given as:



The private data of a class can be accessed only through the member functions of its own class and certain special functions called as friend functions.

In the example class student, the data members name, marks1, marks2, rollno, total_marks are accessed only by the member functions accept(), display(), compute(). The objects declared outside the class cannot access members or functions defined under private or protected.

The member functions declared under public can be accessed by the objects of that class. The call statement stud.execute(); is a valid statement, as execute() is a member function defined under public visibility mode and hence can be accessed through the object stud defined outside the class. Where as the statements: stud.accept(), stud.compute(), stud.display(), stud.marks1 etc would force the compiler to throw error messages “not accessible”. Program –6.2 is another example that demonstrates the operation – addition of two numbers. This class wraps three integer variables, and its related member function to accept data, and perform addition. Since the variable sum is defined under public visibility mode, the object is accessing it.

```
//Program - 6.2
# include <iostream.h>
# include <conio.h>

class add
{
       private:
              int a,b;
       public:
              int sum;

              void getdata()
              {
                     a=5;
                     b=10;
                     sum = a+b;
              }
};

void main()
{
       add s;
       s.getdata();
       cout<<s.sum;
}
```

6.7 Defining methods of a class

```
class add
{
        int a,b;
        public:
                add()
                {
                        a='\0';
                        b='\0';                 Method 1
                }
        void display();
};
void add::display()
{
        int sum;
        sum = a+b;                              Method 2
        cout<<sum;
}
```

In Method 1, the member function add() is declared and defined within class add.

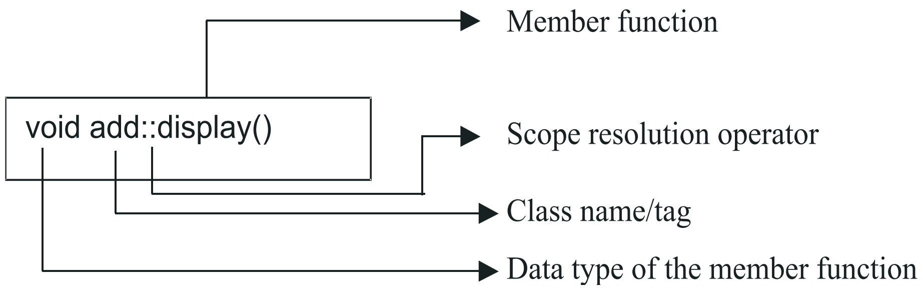
In Method 2, the member function display() is declared within the class, and defined outside the class.

Methods of a class can be defined in both ways. The members defined within the class behave like inline functions.

Member functions defined outside the class has the prototype as

type class_name :: function name();

For example:



The membership label class_name:: (add::) tells the compiler that the function function_name belongs to the class class_name. That is the scope of the function is restricted to the class specified in the function header.

The member function have some special characteristics that are often used in the program development .

- ✓ Several different classes can use the same function name. The ‘membership’ label will resolve their scope

- ✓ Member functions can access the private data of a class. A non-member function cannot do so.
- ✓ A member function can call another member function directly, without using the dot operator. (This is called as nesting of member functions)
- ✓ The member functions can receive arguments of a valid C++ data type. Objects can also be passed as arguments
- ✓ The return type of a member function can be of object data type
- ✓ Member functions can be of static type

6.8 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. Separate memory allocations for the objects are essential because the member variables will hold different data values for different objects

Look at the following class declaration:

```
    class product
{
        int code, quantity;
        float price;
        public:
            void assign_data();
            void display();
};
void main()
{
    product p1, p2;
}
```

Member functions assign_data() and display() 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.

Memory for Objects for p1 and p2 is illustrated:

objects	Data members	Memory alloted
p1	Code, quantity and price	8 bytes
p2	Code,quantity and price	8 bytes

Table 6.2 Memory Allocation for Objects

Member functions of a class can handle arguments like any other non member functions as illustrated in Program - 6.3.

6.9 Static Data Members

A data member of a class can be qualified as static

The static member variable

- ✓ Is initialized to zero, only when the first object of its class is created . No other initialization is permitted
- ✓ Only one copy of the member variable is created (as part of the common pool) and is shared by all the other objects of its class type
- ✓ Its scope or visibility is within the class but its lifetime is the lifetime of the program.

```
    // Program - 6.3
    #include<iostream.h>
    #include<conio.h>
    class product
    {
        int code, quantity;
        float price;
     public:
        void assign_data( int c, int q, float p)
        {
          code = c;
          quantity = q;
          price = p;
        }

        void display()
        {
          cout<<"\n Code : "<<code;
          cout<<"\n Quantity :"<<quantity;
          cout<<"\n Price :  "<< price;
        }
};
void main()
{
        product p;
        p.assign_data( 101, 200, 12.5);
        p.display();
}
```

```
     // Program - 6.4
     // To demonstrate the use of static member variables

     #include<iostream.h>
     #include<conio.h>
     class simple_static
     {
                 int a,b,sum;
                 static int count;
         public:
                 void accept()
                 {
                         cout<<"\n Enter values.. ";
                         cin>>a>>b;
                         sum = a+b;
                         count++;
                 }
                 void display()
                 {
                         cout<<"\n The sum of two numbers … "<<sum;
                         cout<<"\n This is addition… "<<count;
                 }
         };
int static_simple count=0;
void main()
{
         simple_static p1,p2,p3;
         p1.accept();
         p1.display();
         p2.accept();
         p2.display();
         p3.accept();
         p3.display();
}
```

OUTPUT :

```
Enter  values …… 10 20
The sum of two numbers ……….. 30
This is addition   1

Enter values……… 5 7
The sum of two numbers……………12
This is addition 2

Enter values……….. 9 8
The sum of two numbers ……………17
This is addition 3
```

The static variable count is initialized to zero only once. The count is incremented whenever the sum of the two numbers was calculated. Since the function accept() was invoked three times, count was incremented thrice and hence the value is 3. As only one copy of count is shared by all the three objects, the value of count is set to 3. This is shown in Fig. 6.2.

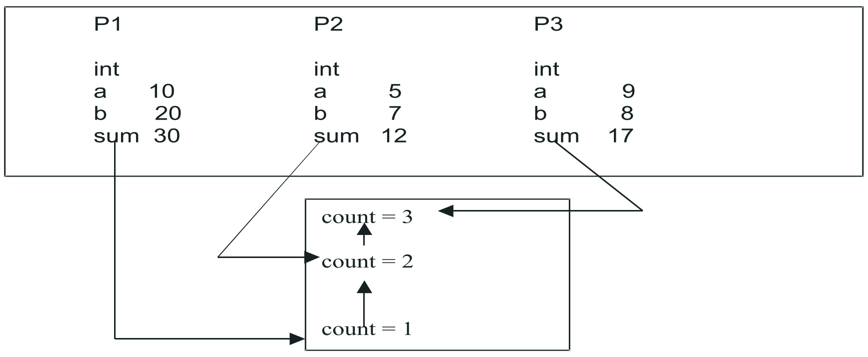


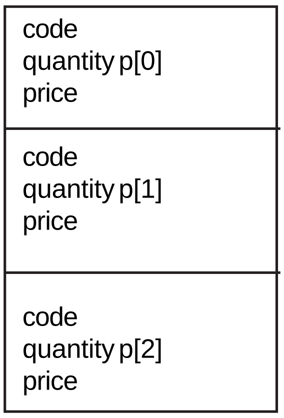
Fig. 6.2 Static Member Variable - count

The initial value to a static member variable is done outside the class.

6.10 Arrays of objects

Consider the following class definition and its corresponding memory allocation:

```
class product
{
    int code,quantity;
    float price;
public :
   void assign_Data();
   void display();
} p[3];

void main()
{
  p[0].assign_Data();
   p[0].display();
}
```



Exercises

I. Identify and correct the errors in the following

```
class x
{
        public:
                int a,b;
                void init()
                {
                        a =b = 0';
                }
                int sum();
                int square();
};
int sum()
{
        return a+b;
}
int square()
{
        return sum() * sum()
}
```

Solution :
int x::sum() and int x::square()

II

```
#include<iostream.h>
class simple
{
                int num1, num2 , sum = 0;
        protected:
                accept()
                {
                        cin>>num1>>num2;
                }
        public:
                display()
                {
                        sum = num1 + num2;
                }
};
void main()
{       simple s;
        s.num1=s.num2= 0;
        s.accept();
        display();
}
```

Solution:

1) The member sum cannot be initialized at the time of declaration
2) The member variable num1 and num2 cannot be accessed from main() as they are private
3) s.accept() is invalid. The method accept() is defined under protected
4) display() should be invoked through an object

III

```
#include<iostream.h>
#include<conio.h>
class item
{
        private:
                int code,quantity;
                float price;
                void getdata()
                {
                        cout<<"\n Enter code, quantity, price ";
                        cin>>code>>quantity>>price;
                }
                public:
                        float tax='\0';
                        void putdata()
                        {
                                cout<<"\n Code : "<<code;
                                cout<<"\n Quantity : "<<quantity;
                                cout<<"\n Price :  "<<price;
                                if( quantity >100 )
                                        tax = 2500;
                                else
                                        tax =1000;
```

```
                    cout<<" \n Tax :"<<tax;
            }
};

void main()
{ item i;   }
```

Complete the following table based on the above program

Memory allocation for instance i	Private data members	Public data members	Methods or data members that can be accessed by i

IV

1) Define a class employee with the following specification private members of class employee

    empno- integer
    ename – 20 characters
    basic – float
    netpay, hra, da, float

calculate () – A function to find the basic+hra+da with float return type
public member functions of class employee
havedata() – A function to accept values for empno, ename, basic, hra, da and call calculate() to compute netpay

dispdata() – A function to display all the data members on the screen

2) Define a class MATH with the following specifications

private members
num1, num2, result – float
init() function to initialize num1, num2 and result to zero

protected members
add() function to add num1 and num2 and store the sum in result
prod() function to multiply num1 and num2 and store the product in the result

public members
getdata() function to accept values for num1 and num2
menu() function to display menu

1. Add…
2. Prod…

invoke add() when choice is 1 and invoke prod when choice is 2 and also display the result.