

CLASSES AND OBJECTS

INTRODUCTION

The most important feature of C++ is the “class”. Its significance is highlighted by the fact that Stroustrup initially gave the name “C with Classes” to his new language. A class is an extension of the idea of structure used in C. It is a new way of creating and implementing a user-defined data type.

LIMITATION OF C STRUCTURE

The standard C does not allow the struct data type to be treated like built-in types. For example, consider the following structure

struct complex

{

float x;

float y;

};

struct complex c1,c2,c3;

The complex numbers **c1**, **c2** and **c3** can easily be assigned values using the dot operator. But we cannot add two complex numbers or subtract one from the other. For example

c3=c1+c2;

Is illegal in C.

Another important limitation of C structures is that they do not permit data hiding. Structure members can be directly accessed by the structure variables by any function anywhere in their scope.

EXTENSION OF STRUCTURE

C++ supports all the features of structures as defined in C. But C++ has expanded its capabilities further to suit its OOP philosophy. It attempts to bring the user-defined types as close as possible to the built-in data types, and also provides a facility to hide the data which is one of the main precepts of OOP.

In C++, a structure can have both variables and functions as members. It can also declare some of its members as ‘private’ so that they cannot be accessed directly by the external functions.

C++ incorporates all these extensions in another user-defined type known as class. There is very little syntactical difference between structures and classes in C++ and, therefore, they can be used interchangeably with minor modification. Since class is a specially introduced data type in C++, most of the C++ programmers tend to use the structures for holding only data, and class to hold both the data and functions.

SPECIFYING A CLASS

A class is a way to bind data and its associated function together. It allows the data and function to be hidden, if necessary, from external use. When defining a class, we are creating a new abstract data type that can be treated like any other built-in data type. Generally a class specification has two parts.

1. **Class declaration**
2. **Class function definition**

The class declaration describes the type and scope of its members. The class function definitions describe how the class functions are implemented.

The general form of a class declaration is

```
class class-name
{
private:
variable declaration;
function declaration;
public:
variable declaration;
function declaration;
};
```

The class declaration is similar to a **struct declaration**. The keyword **class** specifies that what follows is an abstract data of type of **class_name**. the body of a class is enclosed within braces and terminated by a semicolon. The class body contains the declaration of variables and functions. These functions and variables are collectively called members. They are usually grouped under two sections, namely, private and public to denote which of the members are private and which of them are public. The keywords private and public are known as visibility labels. Note that these keywords are followed by **colon(:)**;

The members that have been declared as private can be accessed only from within the class. On the other hand, public members can be accessed from outside the class also. The data hiding (using private declaration) is the key feature of object – oriented programming. The use of the keyword private is optional. By default, the member of a class are private. If both the labels are missing. Then, by default, all the members are private. Such a class is completely hidden from the outside world and does not serve any purpose.

The variable declared inside the class are known as data members and the function are known as member functions. Only the member functions can have access to the private data members and private functions. However, the public members (both functions and data) can be accessed from outside the class. The binding of data and function together into a single class-type variable is referred to as encapsulation.

Class example

Class item

```
{
Int numbe;           //variables declaration
Float coast;         //private by default
Public:
Void getdata(int a int b); // function declaration
Void putdata(void);       //using prototype
};
```

Creating Objects

Once a class has been declared, we can create variable of that type by using the class name (like any other built-in type variable). For example

```
Item x;           // memory for x is created
```

Creates a variable x of type item. In C++, the class variable are known as objects. Therefore, x is called an object of type item.

```
Item x, y, z;
```

The declaration of an object is similar to that of a variable of any basic type. The necessary memory space is allocated to an object at this stage. Like a structure, provides only a template and does not create any memory space for the objects.

Accessing class members

The private data member of a class can be accessed only through the member function of that class. The `main()` cannot contain statement that access private data member directly.

Following is the format for calling a member function.

Object_name.function_name(actual arguments);

For example

x.getdata(100,75.5);

is valid and assign the value 100 to **number** and 75.5 to **cost** of the object **x** by implementing the **getdata()** function.

Similarly the statement

x.putdata();

Defining member functions

Member function can be defined in two places:

- 1) **Outside the class definition**
- 2) **Inside the class definition**

It is obvious that, irrespective of the place of definition, the function should perform the same task. Therefore, the code for the function body would be identical in both the cases. However, there is a subtle difference in the way the function header is defined.

Outside the class definition

Member function that are declared inside a class have to be defined separately outside the class. Their definition are very much like the normal function. They should have a function header and a function body.

An important difference between a member function and a normal function is that a member function incorporated a membership 'identity label' in the header. This 'label' tells the compiler which class the function belongs to.

The general form of a member function definition is

Return-type class-name::function-name (argument declaration)

{

Function body;

}

The membership label `class-name::` 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-name specified in the header line. The symbol `::` is called the **scope resolution operator**.

For example

Void item::getdata(int a, float b)

{

Number =a;

Cost=b;

}

Voit item::putdata(void)

{

Cout<<"Number="<<number;

Cout<<"Cost="<<cost;

}

The member functions 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 the class. A non-member function cannot do so. (however an exception to this rule is a friend function)
- A member function can call another member function directly, without using the dot operator.

Making an outside function inline

One of the objectives of OOP is to separate the details of implementation from the class definition. It is therefore good practice to define the member function outside the class.

We can define a member function outside the class definition and still make it inline by just using the qualifier inline in the header line of function definition.)

```
Class item
{
Int number;
Float cost;
Public:
Void getdata(int,float);
};
Inline item::getdata(int x, float y)
{
Number=x;
Cost=y;
}
```

Nesting of member functions

A member function can be called by using its name inside another member function of the same class. This is known as nesting of member functions.

Private member function

Although it is normal practice to place all the data item in a private section and all the functions in public, some situation may require certain functions to be hidden (like private data) from the outside calls. Tasks such as deleting an account in customer file, or providing increment to an employee are events of serious consequences and therefore the function handling such tasks should have restricted access. We can place these functions in the private sections.

A private member function can only be called by another function that is a member of its class. Even an object cannot invoke a private function using the dot operator.

Static data members

A data members of a class can be qualified as static. The properties of a static member variable are similar to that of a C static variable. A static member variable has certain special characteristics:

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
- It is visible only within the class, but its lifetime is the entire program

Static variables are normally used to maintain values common to the entire class.

Int item :: count; // definition of static data member

Note that the type and scope of each static member variable must be defined outside the class definition. This is necessary because the static data members are stored separately rather than as a part of an object. Since they are associated with the class itself rather than with any class object they are also known as class variables.

Static member function

Like static member variable, we can also have static member functions. A member function that is declared static has the following properties.

- A static function can have access to only other static member (functions or variables) declared in the same class.
- A static member function can be called using the class name (instead of its objects) as follows:

Class-name :: function-name;

Array of objects

An array can be of any data type including struct. Similarly, we can also have arrays of variables that are of the type class. Such variables are called arrays of objects.

For example

Class employee

```
{  
Char name[30];  
Int age;  
Public:  
Void getdata(void);  
Void putdata(void);  
};
```

The identifier employee is a user-defined data type and can be used to create objects that relate to different categories of the employee. For example

```
Employee manager[3];      // array of manager  
Employee foreman[15];     //array of foreman  
Employee worker[75];      // array of worker
```

The array manager contain three **objects(managers)**, **manager[0],manager[1] and manager[2]**, of type employee class. Similarly, the **foreman** array contains **15 objects(foreman)** and the worker array contains 75 objects(**workers**).

Since an array objects behaves like any other array, we can use the usual array-accessing methods to access individual elements and then the dot member operator to access the member functions. For example

Manager[i].putdata();

Will display the data of the ith element of the array manager. That is, this statement requests the objects manager[i] to invoke the member function **putdata()**.

An array of objects is stored inside the memory in the same way as a multi-dimensional array.

Objects as function arguments

Like any other data type, an object may be used as a function argument. This can be done in two ways
A copy of the entire object is passed to the function

Only the address of the object is transferred to the function

The first method is called pass-by-value. Since a copy of the object is passed to the function, any change made to the object inside the function do not affect the object used to call the function. The second method is called pass-by-reference. When the address of the object is passed, the called function works directly on the actual object used in the call. This means that any changes made to the object inside the function will reflect in the actual objects. The pass-by-reference method is more efficient since it requires to pass only the address of the object and not the entire object.

FRIENDLY FUNCTION

The private members cannot be accessed from outside the class. That is, a non-member function cannot have an access to the private data of a class

For example, consider a case where two classes, manager and scientist, have been defined. We would like to use a function `income_tax()` to operate on the objects of both these classes. In such situations, C++ allows the common function to be made friendly with both the classes, thereby allowing the function to have access to the private data to these classes. Such a function need not be a member of any of these classes.

To make an outside function “friendly” to a class, we have to simply declare this function as a friend of the class as shown below:

```
Class ABC
{
.....;
.....;
Public:
.....;
.....;
Friend void xyz(void); //declaration
};
```

The function declaration should be preceded by the keyword `friend`. The function is defined elsewhere in the program like a normal C++ function. The function definition does not use either the keyword `friend` or the scope operator `::`. The functions that are declared with the keyword `friend` are known as friend functions. A function can be declared as a friend in any number of classes. A friend function, although not a member function, has full access rights to the private members of the class.

A friend function processes certain special characteristics

- It is not in the scope of the class to which it has been declared as friend
- Since it is not scope of the class, it cannot be called using the object of that class. It can be invoked like a normal function without the help of any object.
- Unlike member functions, it cannot access the member names directly and has to use an object name and do membership operator with each member name.
- It can be declared either in the public or the private part of a class without affecting its meaning.
- Usually, it has the objects as arguments.