Unit

3

Classes and Objects

1. Classes

A class is one of the most important concepts in Object Oriented programming.

Definition

A class is a user defined type which has data members as well as member functions. It is a blue-print for an object.

A class encapsulates data members and member functions. In C++, a class can be created using the keywords *struct* as well as *class*.

For example, a class account that has data members and operations:

```
class account
{
  public:
    void open();
    void close();
    void withdraw(int);
    void deposit(int);
  private:
    int ac_number;
    char name[80];
    float balance;
};
```

Here, account becomes a new type. To create objects of this type, we can use the name followed by the object name as shown below.

For example,

```
account acc1, acc[10];
```

1.1 Defining Classes

The general form of a class declaration is

```
class class_name
{
  public :
    //data members
     //member functions
  private :
    //data members
     //member functions
  protected :
     //data members
     //member functions
};
```

To create a class, the keyword *class* (or *struct*) should be used. The class_name is a user defined type name. The class body can contain data members and member functions which can be either private, public or protected. According to object oriented principles, the data members should be private and functions should be public. Only the member functions have access to the private data members of the class.

1. **private:** The members, which are declared in the private section, can be accessed only by the member functions of their same class and friends of this class. The private data members are not accessible outside the class. All members are private by default if we use keyword "class" to create the class.

2. **public:** The members in the public section can be accessed outside the class and by the member functions, friends, derived classes and any function outside the class. In other words, the public members of a class are accessible to everyone. All members are public by default if we use keyword "struct" to create the class.

3. **protected:** The members in the protected section can be accessed by the member functions and friends of this class, and also by member functions and friends derived from this class. Normally, when data members of a class need to be accessed from the derived class, they are declared as protected.

Example,

```
class student
{
    private:
        int rollno;
        char name[80];
        float perc;
    public:
        void accept();
        void display();
};
```

The following table lists differences between struct in C and struct in C++.

	struct in C	struct /class in C++
1.	It can only have data members.	It can have data members as well as functions.
2.	The structure name does not become a new typename.	The structure name becomes a new typename.
3.	We have to use the keyword struct as a prefix to create variables.	There is no need to use the struct or class keyword to create objects.
4.	We cannot create or extend one structure from another.	One structure can be created using another using inheritance.
5.	All structure members can be directly accessed outside.	The visibility of members can be controlled using keywords public, private and protected.

6.	*Example*: declaring the structure
	`struct student`
	`{`
	`int rollno ;`
	`char name[80] ;`
	`};`
		Example: declaring the class
		`struct student`
		`{ private :`
		`int rollno ;`
		`char name[80] ;`
		`public :`
		`void accept() ;`
		`void display() ;`
		`} ;`

7.	*Example*: creating variable `struct student s;`	*Example*: creating object `student s;`

1.2 Defining Member Functions

A class can contain data members as well as member functions encapsulated inside. There are two ways in which the member functions of a class can be defined

1. Inside the class
2. Outside the class

1. **Defining member functions inside the class:** Member functions can be defined in the way as regular functions but within the class scope i.e. within { }. The syntax is given bel

Syntax:

```
class class_name
{ //  data members ;
  access-label :
     return-type function-name(arguments)
     {
        //function body
     }
};
```

Example,

```
class student
{
   int rollno;
   char name[80];
   float perc;
public :
   void accept()
   {
      cout<<"Enter the roll number, name and percentage : ";
      cin >> rollno >> name >> perc;
   }
   void display()
   {
      cout<< "Roll number = "  << rollno << endl ;
      cout<< "Name = "  << name << endl ;
      cout<< "Percentage = "  << perc << endl;
   }
} ;
```

Normally, functions are defined within the class when the function code is very small. W member functions are defined within the class, they are treated as "*inline*".

2. **Defining member functions outside the class:** Member functions can also be defined out the class definition. In order to do this, we have to

i. Declare the member function inside the class.

ii. Define the member function outside the class using scope resolution operator (::).

The syntax is given below

Syntax:

```
class class_name
{
  //  data members ;
  access-label :
    return-type function-name (arguments); //declaration
};
return-type class-name :: function-name(arguments)
{
   // function code
}
```

Example:

```
class student
{
  int rollno;
  char name[80];
  float perc;
public:
  void accept();      //declaration
  void display();     //declaration
};
void student :: accept()
{
   cout<<"Enter the roll number, name and percentage : ";
   cin >> rollno >> name >> perc;
}
void student :: display()
{
   cout<< "Roll number = "  << rollno << endl ;
   cout<< "Name = "  << name << endl ;
   cout<< "Percentage = "  << perc << endl;
}
```

1.3 Nesting of Classes

A class definition can also include the definition of another class inside it. Such a class is called as a nested class. The outer class is called a *container* class and the inner class is called the nested or *contained* class. In such a case, the private data members of the contained class are inaccessible to the functions of the container class.

The following *example* illustrates how we can define nested classes.

```
class student
{
   int rollno ;
   char name[80];
   float perc;
   class date
   {
      int dd, mm, yy;
      public:
           void acceptdate();
           void displaydate();
      }birth_date, admission_date;
   public:
      void accept();      //declaration
      void display();     //declaration
};
```

In the above *example*, we have defined the date class as a nested class of the class student. T class is used to create two objects birth_date and admission_date which are members of the class.

Note The date class can only be used with the student class. It has no independent existence and to create objects of the date class separately, we have to use the container class name and scope resolution operator – student::date.

2. Objects

Objects represent real-life entities in an object oriented program. They are instances of a class

Definition:

An object is a software entity which models a real-life entity in an object oriented program.

To create objects, we can create variables of the class just like we create variables of built-in The process of creating objects of a class is called as *instantiation*.

Syntax:

```
class-name object-name;
```

Example,

```
student s1, s2;
```

For example,

```
class student
{
    ...
} s1, s2;
```

An object can be dynamically created using the allocation operator – new. Such an object will exist as long as its memory is not released using the delete operator.

For example,

```
student *p;
p = new student;        // p points to a student object
    ...
```

2.1 Accessing Members

Once we create objects of a class, we need to access data members of the objects and perform operations on the object using member functions. The member access operator (.) is used to access object members. The syntax is given below.

Syntax to access data member using object:

```
object-name.data-member;
```

Syntax to invoke member function using object:

```
object-name.function-name(arguments);
```

Example,

```
student s;
s.rollno = 10; //allowed if rollno is declared public in class
s.accept();
s.display();
```

To access data members using a pointer, a different syntax should be used.

Syntax to access data member using pointer:

```
pointer-name->data-member;
```

Syntax to invoke member function using pointer:

```
pointer-name->function-name(arguments);
```

Example.

```
student *p = new student;
p->rollno = 10;        //accessing data member
p->display();          // invoking member function
```

Note A member function must be called by an object of the class or by a pointer to the class.

The following program illustrates how we can create objects in different ways and acce members and member functions of a class using the object. In the program, s is a student ob p is a pointer to a student object.

Program: Creating objects

```
//This is a C++ program using classes and objects
#include<iostream.h>
class student
{
  int rno;
  char name[20];
  float perc;
 public:
  void accept();
  void display();
};
void student::accept()
{
 cout << "Enter the roll number, name and percentage : ";
 cin >> rno >> name >>perc;
}
void student::display()
{
 cout << "The roll number is : " << rno << endl;
 cout << "The name is : " << name <<endl;
 cout << "The percentage is : " <<perc<<endl;
}
//function main
int main()
{
 student s;
 s.accept();
 s.display();
 student *p;
 p=new student;
 p->accept();
 p->display();
 return 0;

}
```

Output

```
Enter the roll number, name and percentage
 ABC
75.6
The roll number is : 10
The name is : ABC
The percentage is : 75.6
Enter the roll number, name and percentage
DEF
80.2
The roll number is :20
The name is : DEF
The percentage is: 80.2
```

3. String Class

A string is a sequence of characters terminated by the NULL character (\0). The 'C' programming language includes several functions for performing operations on strings.

C++ provides a predefined "**string**" class which supports all the operations on strings and also provides additional functionality. To use this class, we must include the <string> header file.

```
#include<string>
```

A string object can be created in the same way as any other object.

Syntax:

```
string object-name;
```

Example,

```
string s1, s2;
```

Apart from this, the string class provides several **constructors** to create string objects in several ways. *For example:*

```
string s1;                  //   s1 = ""
```
An empty string.

```
string s2( "abcde" );       //   s2 = "abcde"
```
A string with 5 characters.

```
string s3( s2 );            //   s3 = "abcde"
```
A string which is same as another string.

```
string s4( s2, 2 );         //   s4 = "cde"
```
A string having characters starting from position 2 (3rd character) of s2.

```
string s5( s2, 3, 2 );      //   s5 = "de"
```
A string having 2 characters starting from position 3 (4th character) of s2.

```
string s6( 10, '-' );       //   s6 = "----------"
```
A string having 10 '-' characters.

3.1 String Functions

The member functions of the string class perform various operations on the string objects. Some important functions and their purpose are listed in the table below.

Function	Purpose	Example
`const char * data()`	Returns a C-style null-terminated string of characters representing the contents of the string.	string s1("Hello"); char arr[] = s1.data();
`unsigned int length()` `unsigned int size()`	Returns the length of the string.	string s1("Hello"); cout<<s1.length();
`bool empty()`	Returns true if the string is empty, false otherwise.	If(s1.empty()) ...
`void swap(other_string)`	Swaps the contents of this string with the contents of other_string.	string s1("Hello"), s2("Bye"); s1.swap(s2);

`string & append(other_string)`	Appends other_string to this string, and returns a reference to the result string.	string s1("Hello"), s2("Bye");
		cout<<s1.append(s2);
		//gives HelloBye

`string & insert(position, other_string)`	Inserts other_string into this string at the given position, and returns a reference to the result string.	string s1("Hello"), s2("Bye");
		cout<<s1.insert(3,s2);
		//gives HelByelo

`string & erase(position, count)`	Removes count characters from this string, starting with the character at the given position.	string s1("Hello"), s2("Bye");
		cout<<s1.erase(1,2);
		//Removes el from He

`unsigned int find(other_string, position)`	Finds other_string inside this string and returns its position. If position is given, the search starts there in this string, otherwise it starts at the beginning of this string.	string s1("Computer"), s2("Comp");
		pos=s1.search(s2);
		//returns 0

`string substr(position, count)`	Returns the substring starting at position and of length count from this string.	string s1("Computer");
		cout<<s1.substr(2,4);
		//gives mput

2 String Operators

veral operators can be used with string objects. The meaning and *example* of the various operators given in the table below.

Operator	Purpose	Example
=	Copies contents of one string object to another.	string s1("Hello"), s2; s2=s1;
+	Concatenates one string after another string.	string s1("Hello"), s2("Bye"); s3=s1+s2; //Gives HelloBye
[]	Indexing: Gives the character at the specified position.	string s1("Hello"); char ch = s1[1];
== != < <= > >=	String Comparison.	If(s1==s2) etc.

e following program shows some operations on string objects.

```
nclude<iostream>
nclude<string>
t main()


 string s1 = "Hello";
 string s2 = "World";
 string s3;
 int len;
 // string length
 cout<<"Length of s1= "<<s1.length()<<endl;
 // copy s1 into s3
 s3 = s1;
 cout << "s3 : " << s3 << endl;

 // concatenates s1 and s2
 s3 = s1 + s2;
 cout << "s1 + s2 : " << s3 << endl;

 // total length of s3 after concatenation
 cout << s3.length()<< endl;
 //swapping s1 and s2
 s1.swap(s2);
 cout << "After swap s1= "<s1 <<" s2= "<< s2<< endl;
 return 0;
```

Array of Objects

Syntax:

```
class-name array-name[size];
```

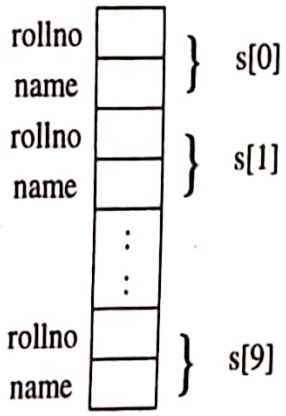
Example:

```
student s[10];
```

Here, s is an array of 10 student objects s[0] to s[9].

The arrays contains 10 objects namely, s[0], s[1] ... s[9] of type *student*. Each object of the a data members and member functions which can be accessed using the dot(.) operator as e above.

All objects are stored in memory in contiguous memory locations. When an object is stored, is allocated only for the data items of the object. Member functions are stored separately used by all the objects of the array. The following diagram shows how the array of student o stored in memory. Each object has data members roll number and name.



Storage of objects in an array

In the following program, we will create an array of n objects of the student class, accept det each and display them. Each student object will have data members – roll number, name, bi and percentage. Birthdate is an object of the user defined date class.

Program: Array of Objects

```
#include<iostream.h>
class date
 {
   int dd,mm,yy;
   public:
     void accept();
     void display();
 };
class student
{
 int rollno;
 char name[20];
 date bdate;
```

```
 float perc;
public:
   void accept();
   void display();
};
void date::accept()
{
    cin>>dd>>mm>>yy;
}
void date::display()
{
   cout<<dd<<"/"<<mm<<"/"<<yy<<endl;
}
void student::accept()
{
  cout<<"Enter roll number, name:";
  cin>>rollno>>name;
   cout<<"Enter birthdate (dd,mm,yy):";
  bdate.accept();
  cout<<"Enter percentage :";
  cin>>perc;
}
void student::display()
{
 cout<<endl<<endl;
 cout<<"Roll Number = "<<rollno<<endl;
 cout<<"Name = "<<name<<endl;
 cout<<"Birthdate = "; bdate.display();
 cout<<"Percentage = "<<perc<<endl;
}
int main()
{
   student s[10];
  int i,n;
  cout<<"How many student objects? :";
  cin>>n;
  for(i=0; i<n; i++)
     s[i].accept();
  cout<<"The student details are:\n";
  for(i=0; i<n; i++)
     s[i].display();
  return 0;
}
```

Output

```
How many student objects?: 2
Enter roll number, name: 1  ABC
Enter birthdate (dd,mm,yy): 12  3
1990
Enter percentage: 78.5
Enter roll number, name: 2  DEF
Enter birthdate (dd,mm,yy): 20  8
1989
Enter percentage: 82.5
The student details are:
Roll Number = 1
Name = ABC
Birthdate = 12/3/1990
Percentage = 78.5
Roll Number = 2
Name = DEF
Birthdate = 20/8/1989
Percentage = 82.5
```

5. "this" Pointer

In C++, there is a special pointer called "this". When a member function is called by an object, the object is automatically passed to the function as an implicit argument.

Definition:

"**this**" is a predefined pointer which points to the implicit object i.e. the object which invokes a member function.

The data members of the implicit object as well as the object itself can be accessed using the "this" pointer. Suppose that you create an object named objA of class A, and class A has a member function `f()`. If you call the function `objA.f()`, the keyword **`this`** in the body of `f()` stores the address of objA.

To access data members or other member functions of the object using the "this" pointer, the following syntax should be used:

Syntax:

```
this->data-member;
this->member-function(args);
```

In some cases, you may want the member function to return the implicit object itself. In such cases, the implicit object can be referred using the "this" pointer. The entire object can be accessed using the syntax:

Syntax to access entire implicit object:

```
*this
```

Look at the following *example* to understand how 'this' pointer is used. In each of the member functions, this pointer is used to access the object's members. In the function named "increment", we are incrementing the implicit object and also returning it so that it can be assigned to another object in main.

Program: Illustrate use of "this" Pointer

```
#include<iostream.h>
class A
{
   int x;
   public:
     void assign()
     {
```

```
    {
      cout<<x;
    }
    A increment();
};
int main ()
{
  A obj ;
  obj.assign();
  obj.display();
  return 0;
}
```

Some points regarding "this"

1. "this" pointer stores the address of the class instance i.e. implicit object.
2. "this" enables pointer to pointer access the members of the class.
3. It cannot be declared.
4. It is automatically initialized with the address of the implicit object.
5. Static member functions belong to the class and not the object. Hence, this pointer cannot be used in static member functions.
6. The size of the "this" pointer is not taken into consideration when calculating the object size.
7. "this" pointers are not modifiable.
8. As friend functions are not member functions, the "this" pointer cannot be used in friend functions.

EXERCISES

A. Short answer questions

1. Define class.
2. Define object.
3. "A class can be created using the struct keyword. State True/False.
4. "The 'this' pointer must be declared in the program". State True/False.
5. "Member functions of a class can be defined outside the class". State True/False.
6. State the use of "this" pointer.
7. Give the syntax for creating an array of objects.
8. State the use of public, private and protected keywords.

B. Review Questions

1. Define class. Give the syntax of creating a class in C++.
2. Define object. Explain how an array of objects can be created in C++.
3. Differentiate between 'struct' keyword and 'class' keyword in C++.
4. Explain various "string" class functions with examples.
5. Explain various "string" class operators with examples.
6. Write a short note on 'this' pointer.
7. Explain array of objects using a suitable example.
8. Explain how member functions can be defined outside the class with example.
9. Explain the use of public and private keywords with an example.

C. Programming Exercises

1. Write a C++ program to define a class "student" having members: roll number, name, marks of 6 subjects, percentage and functions accept() and display (). Create 5 objects of the above class and display them using appropriate output formats.
2. Create a class called House. Inside this, define two classes- Furniture and person, Each class should have functions accept() and display(). In main(), create an instance of House class and call the display() function for each one.
3. Write the definition for a class called Rectangle that has floating point data members length and width. The class has the following member functions:

   void setlength(float) to set the length of data member

   void setwidth(float) to set the width of data member

   float perimeter() to calculate and return the perimeter of the rectangle

   float area() to calculate and return the area of the rectangle

   void show() to display the length and width of the rectangle

   Write main function to create two rectangle objects and display each rectangle and its area and perimeter.
4. Define a class Fraction having two data members: numerator and denominator. Define member functions accept() and display(). Create 10 objects of the fraction class and accept and display them.