UNIT-II

Classes and Objects:

Introduction of Classes: Class Definition, Defining a Members, Objects, Access Control, Class Scope, Memory Allocation for Objects, Static Data Members, Static Member Functions, Arrays of Objects, Friend Functions.

Constructors and Destructors:

Introduction to Constructors, Default Constructors, Parameterized Constructors, Copy Constructors, Destructors.

Introduction to C++ Classes and Objects

The classes are the most important feature of C++ that leads to Object Oriented programming.

Definition of Class:

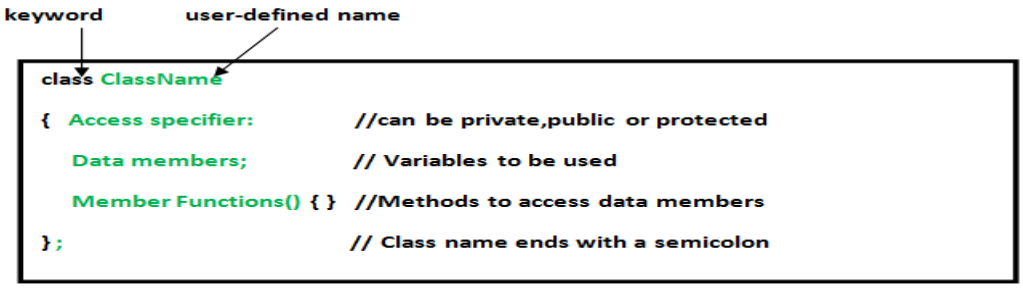
Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating instance of that class.

The variables inside class definition are called as data members and the functions are called member functions.

For example: Class of birds, all birds can fly and they all have wings and beaks. So here flying is a behavior and wings and beaks are part of their characteristics. And there are many different birds in this class with different names but they all posses this behavior and characteristics.

Similarly, class is just a blue print, which declares and defines characteristics and behavior, namely data members and member functions respectively. And all objects of this class will share these characteristics and behavior.

Syntax:



More about Classes

1. Class name must start with an uppercase letter(Although this is not mandatory). If class name is made of more than one word, then first letter of each word must be in uppercase. *Example*,

   class Study, class BankingApplication etc

2. Classes contain, data members and member functions, and the access of these data members and variable depends on the access specifiers (discussed in next section).
3. Class's member functions can be defined inside the class definition or outside the class definition.
4. Class in C++ are similar to structures in C, the only difference being, class defaults to private access control, where as structure defaults to public.
5. All the features of OOPS, revolve around classes in C++. Inheritance, Encapsulation, Abstraction etc.
6. Objects of class holds separate copies of data members. We can create as many objects of a class as we need.
7. Classes do posses more characteristics, like we can create abstract classes, immutable classes, all this we will study later.

Defining Class Members:

Class Members means class variables and member functions.

Defining class variables:

There are 2 ways to define the class members

(i) Through member functions

Ex: getdata() Function.(Used to read Student Details in Student class)

(ii).By accessing through class object in main() function

Ex: int main()

{

```
        Student s;
        s.Srno=101;
        s.Sname=”Rama”;
}
```

Defining Member Functions of the Class

Class Member functions can be defined in 2 ways.

(i) Inside the class
(ii) Outside of the class

(i) Inside the class:

We can define the member function with in the class whenever we are declaring itself.

(i.e)

Example:

```
Class Student
  {
      public:
          int Srno;
          char Sname[20];
      public:
          void getdata()
          {
```

```
        Cout<<"Enter Srno,Sname:";
        cin>>Srno>>Sname;
    }
};
```

(ii) <u>Defining at Outside of the Class:</u>

To define the function at outside of the class we need to use the following syntax:

```
return_type  ClassName:: function_name(Parameterslist if any)
{
    //Set of Statements
}
```

<u>Example:</u>

```
Class Student
    {
        public:
            int Srno;
            char Sname[20];
        public:
            void getdata()
            {
            Cout<<"Enter Srno,Sname:";
```

```
            cin>>Srno>>Sname;
        }
        void display();
    };
void Student::display()
{
    cout<< "Srno="<<Srno;
    cout<<"Sname="<<Sname;
}
```

Objects

Class is a blueprint or a template. No storage is assigned when we define a class.

Definition of Object:

Objects are instances of class, which holds the data variables declared in class and the member functions work on these class objects.

Declaring Objects: When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

```
ClassName  ObjectName;
```

Example:

```
class Abc
{
   int x;
   void display()
```

```
    {
        // some statement
    }
};
int main()
{
    Abc obj;   // Object of class Abc created
}
```

Access Modifiers in C++

Introduction:

Access modifiers are used to implement an important feature of Object-Oriented Programming known as **Data Hiding**.

Consider a real-life example:

The Indian secret informatic system having 10 senior members have some top secret regarding national security. So we can think that 10 people as class data members or member functions who can directly access secret information from each other but anyone can't access this information other than these 10 members i.e. outside people can't access information directly without having any privileges. This is what data hiding is.

Access Modifiers or Access Specifiers in a class are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions.

There are 3 types of access modifiers available in C++:

1. **Public**
2. **Private**
3. **Protected**

Note:In C++, if we do not specify any access modifiers for the members inside the class then by default the access modifier for the members will be **Private**.

1.Public:

- All the class members declared under public will be available to everyone.
- The data members and member functions declared public can be accessed by other classes too.
- The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

```
// C++ program to demonstrate public access modifier

#include<iostream>
using namespace std;

// class definition
class Circle
{
   public:
      double radius;

      double  compute_area()
      {
         return 3.14*radius*radius;
      }

};
 // main function
int main()
{
   Circle obj;

   // accessing public datamember outside class
   obj.radius = 5.5;

   cout << "Radius is: " << obj.radius << "\n";
   cout << "Area is: " << obj.compute_area();
```

```
    return 0;
}
```

2.Private:

- The class members declared as *private* can be accessed only by the functions inside the class.
- They are not allowed to be accessed directly by any object or function outside the class.
- Only the member functions or the friend functions are allowed to access the private data members of a class.

Example:

```
// C++ program to demonstrate private  access modifier
#include<iostream>
using namespace std;
class Circle
{
        // private data member
    private:
      double radius;

        // public member function
    public:
      double  compute_area()
      {   // member function can access private
          // data member radius
          return 3.14*radius*radius;
      }

};

// main function
int main()
{
    // creating object of the class
        Circle obj;
```

```
        // trying to access private data member
         // directly outside the class
          obj.radius = 1.5;
        cout << "Area is:" << obj.compute_area();
        return 0;
}
```

3.Protected: Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by immediate subclass(derived class) of that class.

Example:

```
// C++ program to demonstrate  protected access modifier
#include <bits/stdc++.h>
using namespace std;
 // base class
class Parent
{
   // protected data members
   protected:
   int id_protected;

};

// sub class or derived class
class Child : public Parent
{
   public:
   void setId(int id)
   {

       // Child class is able to access the inherited
       // protected data members of base class

          id_protected = id;
```

```
    }

    void displayId()
    {
        cout << "id_protected is: " << id_protected << endl;
    }
};

// main function
int main() {

    Child obj1;
        // member function of the derived class can
        // access the protected data members of the base class

    obj1.setId(81);
    obj1.displayId();
    return 0;
}
```

Class Scope:

A name declared within a member function hides a declaration of the same name whose scope extends to or past the end of the member function's class.

When the scope of a declaration extends to or past the end of a class definition, the regions defined by the member definitions of that class are included in the scope of the class.

Members defined lexically outside of the class are also in this scope. In addition, the scope of the declaration includes any portion of the declarator following the identifier in the member definitions.

The name of a class member has *class scope* and can only be used in the following cases:

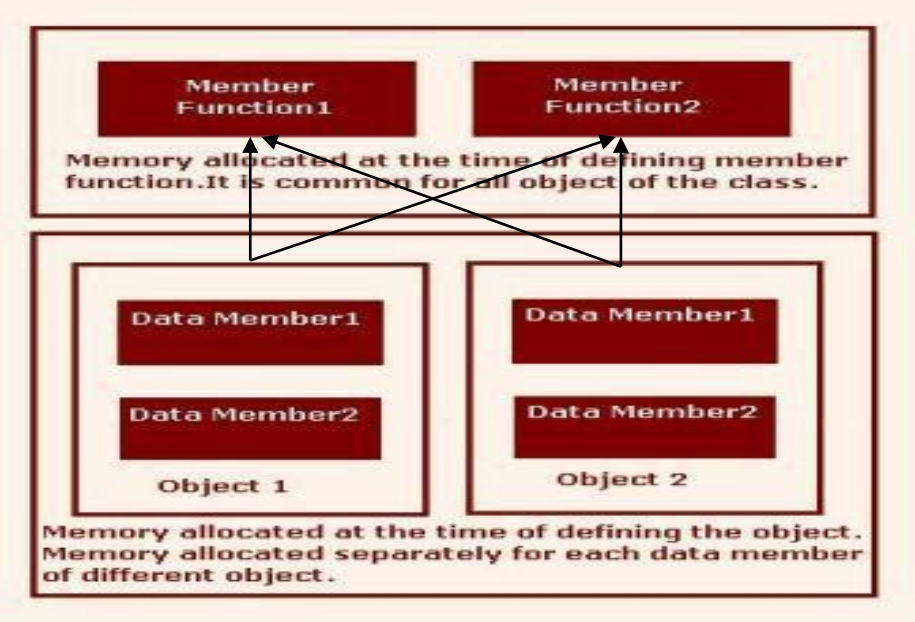
- In a member function of that class
- In a member function of a class derived from that class
- After the '.' (dot) operator applied to an instance of that class
- After the '.' (dot) operator applied to an instance of a class derived from that class, as long as the derived class does not hide the name.

- After the -> (arrow) operator applied to a pointer to an instance of that class
- After the -> (arrow) operator applied to a pointer to an instance of a class derived from that class, as long as the derived class does not hide the name
- After the :: (scope resolution) operator applied to the name of a class
- After the :: (scope resolution) operator applied to a class derived from that Class.

Memory Allocation for Objects of Class

- Once you define class it will not allocate memory space for the data member of the class.
- The memory allocation for the data member of the class is performed separately each time when an object of the class is created.
- Since member functions defined inside class remains same for all objects, only memory allocation of member function is performed at the time of defining the class.
- Thus memory allocation is performed separately for different object of the same class. All the data members of each object will have separate memory space.
-

The memory allocation of class members is shown below:



Hence data member of the class can contain different value for the different object, memory allocation is performed separately for each data member for different object at the time of creating an object.

Member function remains common for all object. Memory allocation is done only once for member function at the time of defining it.

Static Data Members in C++

- Static data members are class members that are declared using the static keyword.
- There is only one copy of the static data member in the class, even if there are many class objects.
- This is because all the objects share the static data member.
- The static data member is always initialized to zero when the first class object is created.

The syntax of the static data members is given as follows –

```
static data_type  data_member_name;

          (or)

static datatype  Var_name;
```

In the above syntax, static keyword is used. The data_type is the C++ data type such as int, float etc. The data_member_name is the name provided to the data member.

Accessing of Static Variables:

Static variables can be accessed through class name itself.

Syntax:

```
Classname::Static_Var_name;
```

A program that demonstrates the static data members in C++ is given as follows −

Example:

```
#include <iostream>
using namespace std;
class Student
{
  private:
  int rollNo;
  char name[10];
  int marks;
  public:
  static int objectCount;
  Student()
  {
    objectCount++;
  }
 void getdata()
 {
    cout << "Enter roll number: "<<endl;
    cin >> rollNo;
    cout << "Enter name: "<<endl;
    cin >> name;
    cout << "Enter marks: "<<endl;
    cin >> marks;
  }
  void putdata()
 {
    cout<<"Roll Number = "<< rollNo <<endl;
    cout<<"Name = "<< name <<endl;
    cout<<"Marks = "<< marks <<endl;
    cout<<endl;
  }
};
int Student::objectCount = 0;
int main(void)
{
  Student s1;
```

```
    cout<<"\nReadind and displaying Student1 Details";
    s1.getdata();
    s1.putdata();
    Student s2;
    cout<<"\nReading and displaying Student3 Details\n";
    s2.getdata();
    s2.putdata();
    Student s3;
    cout<<"\nReadind and displaying Student3 Details\n";
    s3.getdata();
    s3.putdata();
    cout << "Total objects created = " << Student::objectCount << endl;
    return 0;
}
```

Program Explanation:

In the above program, the class student has three data members denoting the student roll number, name and marks. The objectCount data member is a static data member that contains the number of objects created of class Student. Student() is a constructor that increments objectCount each time a new class object is created.

There are 2 member functions in class. The function getdata() obtains the data from the user and putdata() displays the data.

Note:

Static Data Member

- A type of data member that is shared among all objects of class is known as **static data member**.
- The static data member is defined in the class with ***static*** keyword.
- When a data member is defined as static, only *one variable is created* in the memory even if there are *many objects* of that class.
- A static data item is useful when all objects of the same class must share a common item of information.
- The characteristics of a static data member are same as normal static variable.

Static Member Functions

Definition:

The function which is declared using the keyword ‘static’ is called “Static Function”.

Note:

1. By declaring a function member as static, you make it independent of any particular object of the class.
2. A static member function can be called even if no objects of the class exist and the **static** functions are accessed using only the class name and the scope resolution operator **::**.

```
class_name:: function_name(perameter);
```

3. A static member function can only access static data member, other static member functions and any other functions from outside the class.
4. Static member functions have a class scope and they do not have access to the **this** pointer of the class.
5. You could use a static member function to determine whether some objects of the class have been created or not.

Let us try the following example to understand the concept of static function members –

```
#include <iostream>
 using namespace std;
 class Box
{
    public:
      static int objectCount;
    private:
    double length;     // Length of a box
    double breadth;    // Breadth of a box
```

```
    double height;     // Height of a box
   // Constructor definition
    Box(double l = 2.0, double b = 2.0, double h = 2.0)
    {
            cout <<"Constructor called." << endl;
            length = l;
            breadth = b;
            height = h;
            // Increase every time object is created
            objectCount++;
    }
    double Volume()
    {
        return length * breadth * height;
    }
    static int getCount()
    {
        return objectCount;
    }
};
// Initialize static member of class Box
int Box::objectCount = 0;
int main(void)
{
  // Print total number of objects before creating object.
  Cout << "I Stage Count: " << Box::getCount() << endl;
```

```
    Box Box1(3.3, 1.2, 1.5);    // Declare box1
    Box Box2(8.5, 6.0, 2.0);    // Declare box2
    // Print total number of objects after creating object.
    Cout << "Final Stage Count: " << Box::getCount() << endl;
    return 0;
}
```

Array of Objects in C++

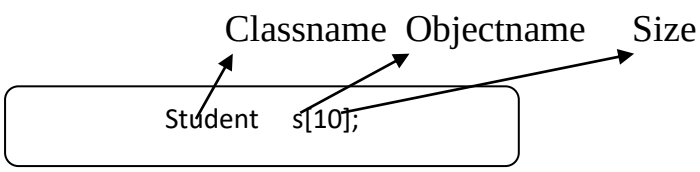
Now, suppose we have 50 students in a class and we have to input the name and marks of all the 50 students. Then creating 50 different objects(Since each object will hold one Student details itself) and then inputting the name and marks of all those 50 students is not a good option.

In that case, we will create an **array of objects** as we do in case of other data-types.

Syntax to creare Array of objects:

```
Class_Name    Object_name[Size];
```

Example:



Program: Write a C++ program to demonstrate Array of Objects.For that read 10 student details and display those.

Code:

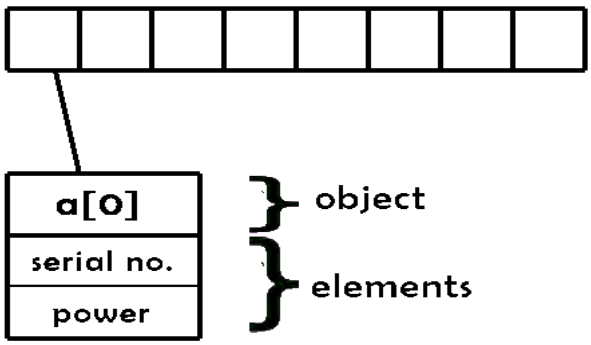
```
#include <string>
using namespace std;
```

```
    class Student
    {
                string name;
                int marks;
                public:
                void getName()
                {
                   getline( cin, name );
                }
                void getMarks()
                {
                     cin >> marks;
                }
               void displayInfo()
               {
                     cout << “Name : “ << name << endl;
                     cout << “Marks : “ << marks << endl;
               }
};
int main()
{
```

```
        Student st[5];
       for( int i=0; i<5; i++ )
      {
          cout << “Student “ << I + 1 << endl;
          cout << “Enter name” << endl;
         st[i].getName();
        cout << “Enter marks” << endl;
        st[i].getMarks();
   }
   for( int i=0; i<5; i++ )
   {
      cout << “Student “ << I + 1 << endl;
     st[i].displayInfo();
}
return 0;
}
```

Explanation:

Student st[5]; - We created an array of 5 objects of the Student class where each object represents a student having a name and marks.The first for loop is for taking the input of name and marks of the students. **getName()** and **getMarks()** are the functions to take the input of name and marks respectively.
The second for loop is to print the name and marks of all the 5 students. For that, we called the **displayInfo()** function for each student.



Friend Functions In C++

Introduction:

C++ supports the feature of encapsulation in which the data is bundled together with the functions operating on it to form a single unit. By doing this C++ ensures that data is accessible only by the functions operating on it and not to anyone outside the class.

This is one of the distinguishing features of C++ that preserves the data and prevents it from leaking to the outside world.

But in some real-time applications, sometimes we might want to access data outside the bundled unit. For example, an outsider class might want to access private and protected data of a C++ class.

C++ provides a facility for accessing private and protected data by means of a special feature called “friend” function.

Definition of Friend Function:

When the function is declared as a friend, then which can access the private and protected data members of the class is called “Friend Function”.

A friend function in C++ is a function that is preceded by the keyword “friend”.

A friend function is declared inside the class with a friend keyword preceding as shown below.

```
Class className
{

        ……

        friend returnType functionName(arg list);
};
```

As shown above, the friend function is declared inside the class whose private and protected data members are to be accessed. The function can be defined anywhere in the code file and we need not use the keyword friend or the scope resolution, operator.

There are some points to remember while implementing friend functions in our program:

- A friend function can be declared in the private or public section of the class.
- It can be called like a normal function without using the object.
- A friend function is not in the scope of the class, of which it is a friend.
- A friend function is not invoked using the class object as it is not in the scope of the class.
- A friend function cannot access the private and protected data members of the class directly. It needs to make use of a class object and then access the members using the dot operator.
- A friend function can be a global function or a member of another class.

Example Of A Friend Function

```
#include<iostream>

#include <string>

using namespace std;

class Sample

{

  int length, breadth;

  public:
```

```
  Sample(int length, int breadth)
  {
       this.length=length;
       this.breadth=breadth)
   }
  friend void calcArea(Sample s); //friend function declaration
};
//friend function definition
void calcArea(Sample s)
{
      cout<<"Area = "<<s.length * s.breadth;
 }
int main()
  {
       Sample s(10,15);
        calcArea(s);
        return 0;
   }
```

Constructors and Destructors

Constructors

Introduction:

Basically in C++,class data members can be initialized in 2 ways.

1. Through class Member Functions.

Ex: getdata() Function.(Used to read Student Details in Student class)

2.By accessing through class object in main() function

Ex: int main()

```
{
        Student s;
        s.Srno=101;
        s.Sname=”Rama”;
}
```

But,whenever we want to initialize the class members while creation of the class object itself,then we can use the “Constructors”.

Definition of Constructor:

A special function consists the same name as class name and used to initialize class members while object creationis called “Constructor”.

Syntax:

```
Classname(Parameterslist if any)
{
        //Set of Statements
 }
```

Example:

```
Student()
{
    Cout<<”Hi,I am Constructor without Parameters”;
}
```

Important Points:

1.Constructor will have same name as class name.

2.Constructor will not return any value even ‘void’ also.

3.Constructor used to initialize the class members.

4. Constructors can be defined either inside the class definition or outside class definition using “class name” and “scope resolution :: operator”.

Example:

```
class A
{
   public:
   int i;
   A(); // constructor declared
};
// constructor definition
A::A()
{
   i = 1;
}
```

Types of Constructors in C++

Constructors are of three types:

1. Default Constructor
2. Parametrized Constructor

3. Copy Constructor

1.Default Constructor

The Constructor which does not take any parameters is called “Default Constructor”.

Syntax:

```
class_name()
{
    // constructor Definition
}
```

Example:

```
class Cube
{
        public:
        int side;
        Cube()
        {
           side = 10;
        }
};
int main()
{
   Cube c;
   cout << c.side;
```

}

Note:

1. Whenever the class object is created constructor will be invoked.
2. A default constructor is so important for initialization of object members, that even if we do not define a constructor explicitly, the compiler will provide a default constructor implicitly.

2. Parametrized Constructor

The constructor which takes parameters is called "Parameterized Constructor".

- Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

Example:

```
class Cube
{
   public:
   int side;
   Cube(int x)
   {
      side=x;
   }
};
int main()
```

```
{
   Cube c1(10);
   Cube c2(20);
   Cube c3(30);
   cout << c1.side;
   cout << c2.side;
   cout << c3.side;
}
```

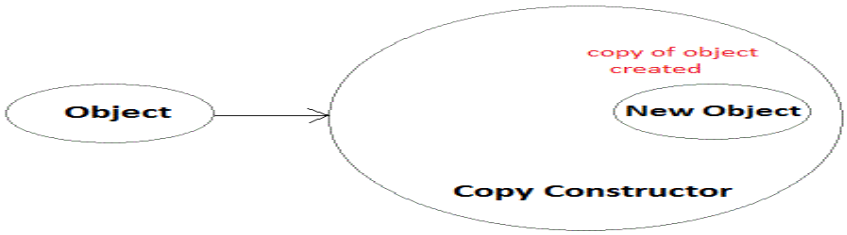
3.Copy Constructors

Definition: These are special type of Constructors which takes an object as argument, **and is used to copy values of data members of one object into other object.**

Syntax of Copy Constructor

```
Classname(classname & objectname)
{
   . . . .
}
```

It creates a new object, which is exact copy of the existing copy, hence it is called **copy constructor**.



Invoking Copy Constructor:

Syntax:

Class_name Objectname2=Objectname1;

(or)

Class_name Objectname2(Objectname1);

Example:

Student s2=s1;

(or)

Student s2(s1);

Example:

```
#include<iostream>
using namespace std;
class Samplecopyconstructor
{
   private:
   int x, y;   //data members
```

```
    public:
    Samplecopyconstructor(int x1, int y1)
    {
        x = x1;
        y = y1;
    }
    /* Copy constructor */
    Samplecopyconstructor (Samplecopyconstructor &sam)
    {
        x = sam.x;
        y = sam.y;
    }
    void display()
    {
        cout<<x<<" "<<y<<endl;
    }
};
/* main function */
int main()
{
    Samplecopyconstructor obj1(10, 15);     // Normal constructor
```

```
Samplecopyconstructor obj2 = obj1;     // Copy constructor
        (or)
Samplecopyconstrucror obj2(&obj1)
cout<<"Normal constructor : ";
obj1.display();
cout<<"Copy constructor : ";
obj2.display();
return 0;
}
```

Destructors in C++

- Destructor is a special class function which destroys the object as soon as the scope of object ends.
- The destructor is called automatically by the compiler when the object goes out of scope.
- The syntax for destructor is same as that for the constructor, the class name is used for the name of destructor, with a **tilde** ~ sign as prefix to it.

```
~Classname()
{
   . . . .
}
```

Example:

```
class A
```

```
{
    // constructor
    A()
    {
        cout << "Constructor called";
    }

    // destructor
    ~A()
    {
        cout << "Destructor called";
    }
};

int main()
{
    A obj1;   // Constructor Called
    int x = 1
    if(x)
    {
        A obj2;  // Constructor Called
```

```
    }   // Destructor Called for obj2
} //  Destructor called for obj1
```