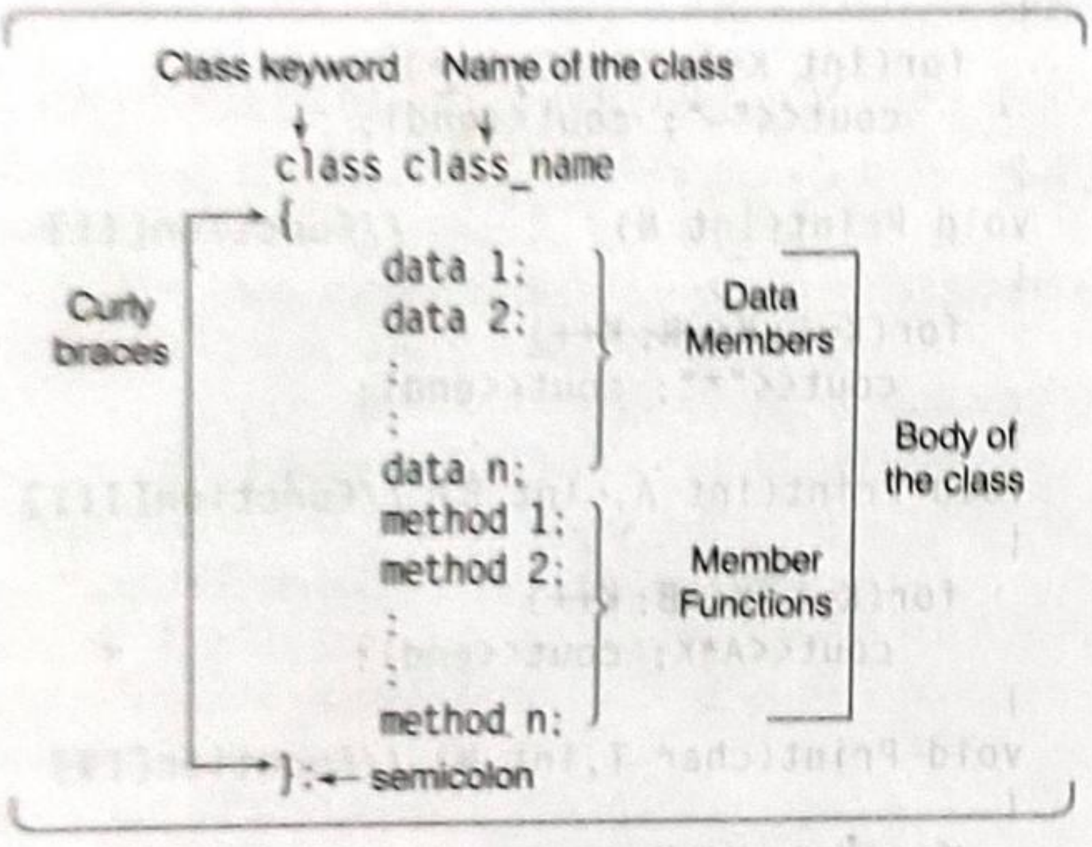
TOPIC 2 Implementation of OOPs Concept in C++

The most important feature of C++ is the "class". 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. A class is used to specify the form of an object. It combines data representation and methods for manipulating the data into one package. The data and methods within a class are called **members of the class.**

Class Definition

A class definition starts with the keyword class followed by the class name and the class body, enclosed within a pair of curly braces. A class definition must be followed by a semicolon.



Members of a Class

The two kinds of members of a class are data members and member functions.

Data Members

The variables declared inside the class definition are called as data members or member variables of the class. And, variables are said to have class scope, i.e. they will exist in the memory as long as the object of class exist. As soon as the object of class goes out of scope, these will be destroyed.

Member Functions

A member function or method of a class is a function that has its definition or its prototype within the class definition like any other variable.

e.g.

```
class Box
{
   public:
      double length;
      double breadth;
      double height;
      double getVolume(void);
      void setValues(void);
};
```

(Data Members: length, breadth, height; Member Functions: getVolume(void), setValues(void))

Arrays Within a Class

The arrays can be used as member variables in a class. The following class definition is valid:

```
const int size = 10;
   // provides value for array size
class array
{
   int a[size];
   // 'a' is int type array
   public:
      void setval(void);
      void display(void);
};
```

The array variable a[ ] declared as a private member of the class. Array can be used in the member functions, like any other variable. We can perform any operation on it. For instance, in the above class definition, the member function setval( ) sets the values of elements of the array a[ ] and display( ) function displays the value of the array a[ ]. Similarly, we may use other member functions to perform any other operations on the array values.

Visibility Modes or Access Specifiers

C++ offers the flexibility to decide which data members or class methods should be accessible outside the class or which should not. An access specifier plays an important role to define the boundary between accessible and inaccessible parts of the class. The access restriction to the class members is specified by the **public, private** and **protected** sections within the class body. The keywords public, private and protected are called **access specifiers** or **access modifiers**. A class can have multiple public, protected or private labeled sections. Each section remains in effect until either another section label or the closing right curly braces of the class body is seen.

The general form of class with access specifier are as follows:

```
class class_name
{
   public:
      // public members here
   protected:
      // protected members here
   private:
      // private members here
};
```

Public Members

The public members are accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function. The public data variables provide the concept of abstraction as they contain the important details of the class available to user.

Private Members

The private member variables or functions cannot be accessed or even viewed from outside the class. Only the class and friend functions can access private members, i.e. private members can be accessed indirectly outside the class by calling them inside public members of the class and then calling those public members outside the class through objects of that class.

The data that is declared inside the private section are secure and hidden from the user. The private section of the class hides the complexity or background details, which cannot accessed by the user. This feature is called encapsulation. User can use the private data but cannot make accidental manipulation.

Note *By default, all the members of a class would be private.*

Protected Members

The protected member variables or functions are very similar to a private member but it provides one additional benefit that they can be accessed in **derived classes.**

We will have a detailed discussion on protected access specifier in chapter 4 (Inheritance).

e.g. a complete definition of class with public, private, protected and default visibility mode.

```
class  ABC
{
   private:
      int a;
      void f1();
   public:
      int c;
      void f3();
   protected:
      int d;
      void f4();
};
```

by default private members cannot be accessible outside the class but accessible inside the class.

private members, cannot access outside the class but accessible inside the class.

public members can be accessible outside the class but with the help of class variable.

protected members cannot be accessible outside the class but accessible within subclass of this class.

Member Function Definition

In the above example, we just declared only data variable and member function. We did not defined the member function that will operate on data. In this topic, we will discuss that how to define the member functions. The member functions can be defined in two ways as follows:

Member Function Inside the Class Definition

When a function is defined inside a class, we do not require to place a membership label alongwith the function. In this method, the function is defined in the class itself at the place of declaration. This function is treated as inline function even if you do not use the inline specifier.

In case of inline function, the compiler inserts the code of body of the function at that place where it is invoked (called) and doing so the program execution is faster but memory penalty is there. We can also define a function inline by writing keyword inline before the function definition inside the class but noted that it does not work for the following situations:

(i) For functions that return values and are having a loop or a switch or a goto.

(ii) For functions not returning values; if a return statement exists.

(iii) If functions contain static variables.

(iv) If a function is recursive (a function that calls itself).

e.g. here, is an example of function definition inside the class to find the value of a Box.

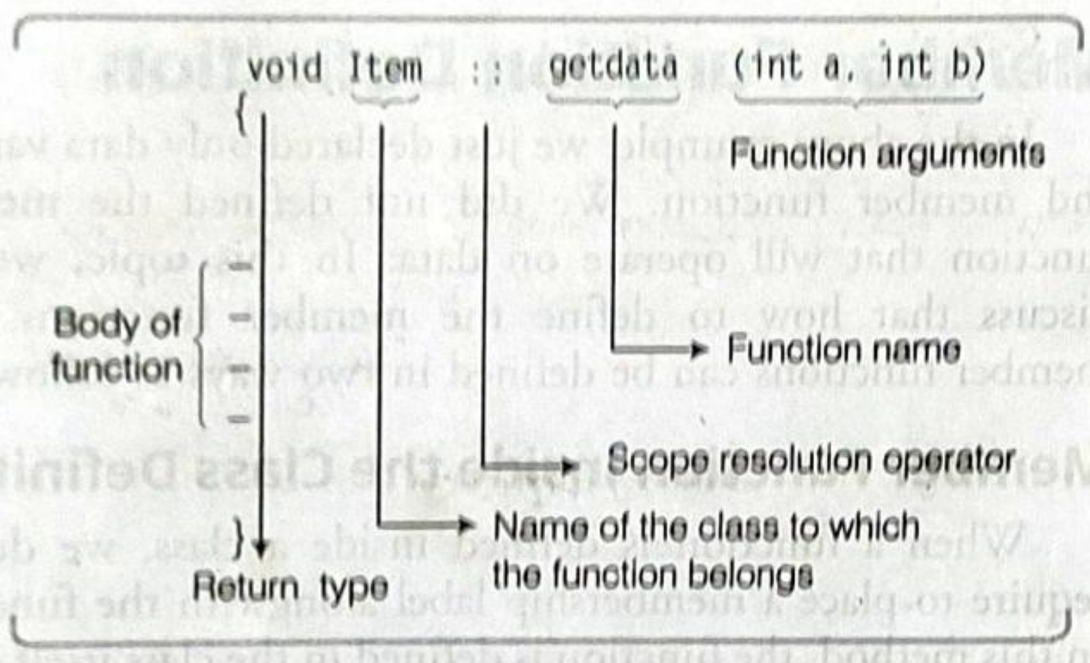
```
class Box
{
   public:
      double length;  //Length of a box
      double breadth; //Breadth of a box
      double height;  //Height of a box
      double getVolume(void)
      {
         return length*breadth*height;
      }
};
```

Member Function Definition Outside the Class

The member functions of a class can be defined outside the class definition. It is only declared inside the class but defined outside the class. The general form of member function definition outside the class definition is:

```
Return_type Class_name :: Function_name
                          (argument list)
{
   Function body
}
```

e.g.



The scope resolution operator (::) specifies the class to which the member being declared belongs, granting exactly the same scope properties as if this function definition was directly included within the class definition.

```
class Box
{
   public: //Data members
      double length;
      double breadth;
      double height;
            // Member functions declaration
      double getVolume(void);
      void setLength(double len);
      void setBreadth(double bre);
      void setHeight(double hei);
};
            //  mber functions definition
double Box :: getVolume(void)
{
   return length * breadth * height;
}
void Box :: setLength(double len)
{
   length = len;
}
```

The main difference between the member function and a normal function is that the member function requires a membership "Identity label" to tell the compiler, which class the functions belong to while there is no need of any kind of identity label to define a normal function.

These are the following characteristics of member function given below:

(i) By using the membership identity label, more than two classes can have same function name.

(ii) The member function can be the private data of that class into which it is declared.

(iii) It can call the another member function without using any special kind of operator (as called the normal function).

Object

Objects are the variables of a user defined data type called class. In other words, the class acts as a data type and the object act as its variable. When we define a class, it does not define or create objects of that class, rather it only specifies what type of information the objects of this class type will be contained. Once a class has been defined its object can be defined like any other variable.

Declaration of Objects

The objects can be declared in two ways:

(i) First defining a class and declare the name(s) of object after closing the right curly braces. i.e. between right curly braces and semicolon.

e.g.

```
class student
{
   int rollNo;
   int age;
   void calcmarks();
   public:
      void getdata();
      void setdata();
} S1, S2, S3 ... SN;
   // memory for S1, S2, S3... SN
                        are created
```

Here, S1, S2, S3, ... SN are objects of type student.

or S1, S2, S3 ... SN are called instance variables or class variables of type student. This is not more useful method to declare an object. So, we prefer second method of creating object when we required to create.

(ii) Second way of declaring an object is to write the name of the class and then name(s) of object.

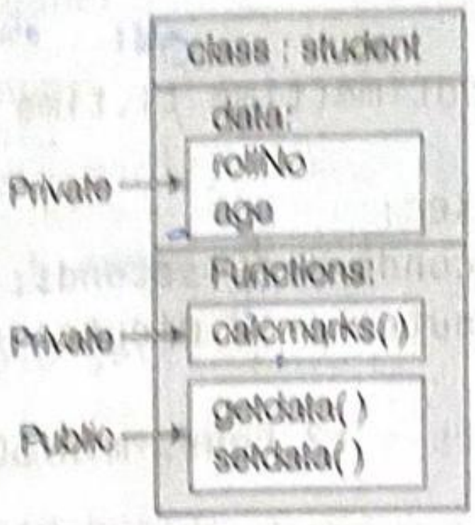
Syntax

```
classname objectname(s);
```

e.g.

```
student S1; // memory for S1 is created
student S1, S2, S3... SN;
```

Here, S1, S2, S3,.. SN are also called instance variables or class variables or objects of class student.



Representation of objects in memory

Accessing class members from object

Dot operator (.) is used to access public member either data member or member function.

The member functions that are declare to be public can be accessed from outside the class.

We can access them using the following syntax:

```
objectname.datamembername;
                or
objectname.datamemberfunction( );
```

e.g. `student S1;`

`S1.getdata();`

Here, S1 is an instance of student and getdata() is a member function in student class.

The data member that are declared to be public can be accessed from outside the class.

Syntax

```
objectname.datamembername
                        = some value;
```

e.g.

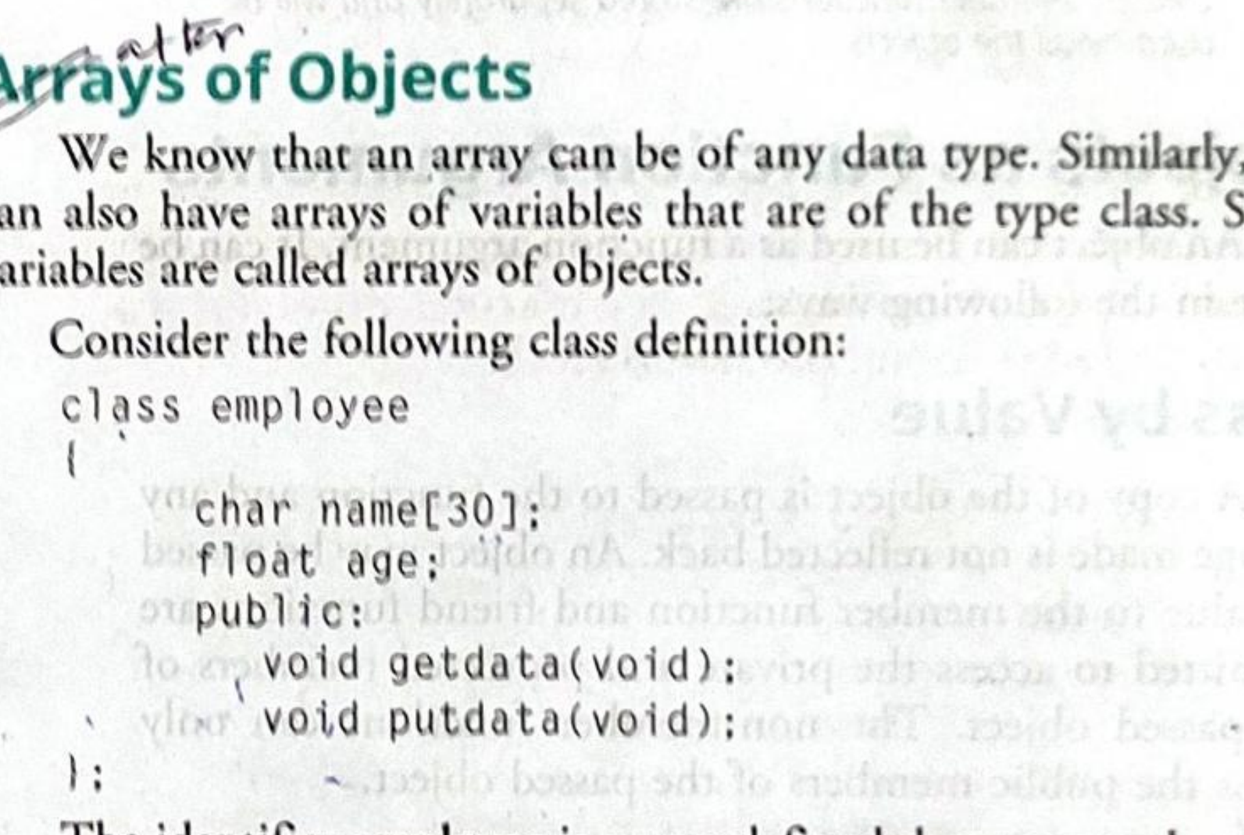
```
student S1;
S1.rollNo = 1001;
   or
cin>>S1.rollNo;
```

Memory Allocation for Objects

The member functions are created and placed in the memory space only once when they are defined as a part of a class specification. Since, all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created.

Space for member variables is allocated separately for each objects. Separate memory locations for the objects are essential, because the member variables will hold different data values for different objects.

This is shown in figure:



Memory for objects

Arrays of Objects

We know that an array can be of any data type. Similarly, we can also have arrays of variables that are of the type class. Such variables are called arrays of objects.

Consider the following class definition:

```
class employee
{
    char name[30];
    float 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 employees.

e.g.

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

The array manager contains three objects(managers), namely 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 (worker). Since, an array of objects behaves like any other array, we can use the usual array accessing methods to access individual elements and then the dot operator to access the member functions.

The friend Keyword

A friend function of a class is defined outside the class's scope, but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, are not member functions. A friend can be a **function, function template** or **class template**, in that case where the entire class and all of its members are friends. To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend** as follows:

```
class Box
{
   double width;
   public:
      double length;
      friend void printWidth(Box box);
      void setWidth(double wid);
};
```

To declare all member functions of class ClassTwo as friends of class ClassOne, place the following declaration in the definition of class ClassOne.

```
friend class ClassTwo;
```

Program 4. *To illustrate the use of friend keyword.*

```
#include<iostream.h>
#include<conio.h>
class Box
{
   double width;
   public:
      friend void printWidth(Box box);
      void setWidth(double wid);
};
void Box :: setWidth(double wid)
{
   width = wid;
}
          // Note: printWidth() is not a
          // member function of any class.
void printWidth(Box box)
{
   /* Because printWidth() is a friend of Box, it
   can directly access any member of this class*/
   cout<<"Width of box "<<box.width<<endl;
}
void main()
{
   Box box;
   box.setWidth(10.0);
   /* Use friend function to print the
   width*/
   printWidth(box);
   getch();
}
```

When the above code is compiled and executed, it produces following result:

```
Width of box 10
```

Static Members of a C++ Class

Static Variables

We can define class members as static using 'static' keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member is used.

A static member is shared by all objects of the class. All static data is initialised to zero, when the first object is created, if no other initialisation is present. We can't put it in the class definition but it can be initialised outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator (::) to identify which class it belongs to.

It is generally used to store the values common to the whole class.

The static data members differ from ordinary data member in the following way:

(i) Only a single copy of the static data member is used by all the objects.

(ii) It can be used within the class but its lifetime scope is the whole program.

The definition of static data members outside the class is must as these are stored separately rather than as part of an object.

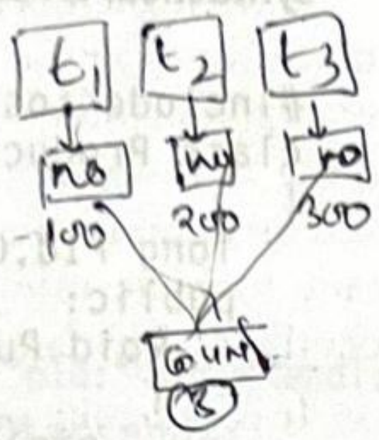
Program 5. *To show the use of static variables.*

```
#include<iostream.h>
#include<conio.h>
class item
{
   static int count;
   int number;
   public:
      void getdata(int a)
      {
         number = a;
         count++;
      }
      void getcount(void)
      {
         cout<<"count ";
         cout<<count<<"\n";
      }
};
int item :: count;
void main()
{
   item a,b,c; //count is
               //initialised to zero
```

```
    a.getcount();
    b.getcount();
    c.getcount();
    a.getdata(100);
    b.getdata(200);
    c.getdata(300);
    cout<<"After reading data\n";
    a.getcount();
    b.getcount();
    c.getcount();
    getch();
}
```

Output

```
count 0
count 0
count 0
After reading data
count 3
count 3
count 3
```



Static Functions

By declaring a function member as static, you make it independent of any particular object of the class.

A static member function can be called even if no objects of the class exist. The static functions are accessed using only the class name and the scope resolution operator (::).

A static member function can only access static data members, other static member functions and any other functions from outside the class.

Static member functions have a class scope and they do not have access to the 'this' pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

In C++, static member function differ from other member function in the following ways:

(i) It can access only static data members and static member functions.

(ii) It is called by the name of the class rather than an object.

Program 6. *To show the use of static function .*

```
#include<iostream.h>
#include<conio.h>
class test
{
    int code;
    static int count;
  public:
    void setcode(void)
    {
      code = ++count;
    }
    void showcode(void)
    {
      cout<<"object number ";
      cout<<code<<endl;
    }
    static void showcount(void)
    {
      cout<<"count "<<count<<endl;
    }
};
int test :: count;
void main()
{
  test t1, t2;
  t1.setcode();
  t2.setcode();
  test :: showcount();
    //accessing static function
  test t3;
  t3.setcode();
  test :: showcount();
  t1.showcode();
  t2.showcode();
  t3.showcode();
  getch();
}
```

Output

```
count 2
count 3
object number 1
object number 2
object number 3
```