Unit_2 Class Object Constructor Desrutcor

Classes and Objects, Constructor and Destructor	• C structures revisited • Specifying a class • Local Classes • Nested Classes • Defining member functions, nesting of Member functions, private member function, making outside function inline • Arrays within a class • Memory allocation for objects • Static data member • Static member functions • Arrays of objects • Objects as function arguments • Friendly functions • Returning objects • Const member function • Pointer to members
	• Characteristics of constructor • Explicit constructor • Parameterized constructor • Multiple constructor in a class • Constructor with default argument • Copy constructor • Dynamic initialization of objects • Constructing two dimensional array • Dynamic constructor • MIL, Advantage of MIL • Destructors

c structure revisited

- Structure in an User Defined Data Type.
- A Structure contains a number of data types group together.
- These data types may or may not be of same type.
- For **example**, an entity Student may have its name (string), roll number (int), marks (float).
- **Syntax of creating a structure :**
- struct [structure tag]
- {
    - member definition/declaration ; member definition/declaration ;
    - ...
    - member definition/declaration ;
- } [one or more structure variables];

C structure revisited

- **How to declare structure variables/object**
- **variable?**
- Before semicolon at structure terminates .
- At global declaration section (Global Scope) .
- Inside the main function (Local Scope) .
- **Syntax :**
  - struct <tag_name> <object_name>,[obj2,3,4….];
- **Accessing Structure Members**
- To access any member of a structure, we use the member
- access operator (.).
- The member access operator is coded as a period between the structure variable name and the structure member that we wish to access.
- You would use struct keyword to define variables of
- structure type.
- **Syntax :**
  - structure variable_name.structure_member_name

C structure revisited

```
struct Point
{
    int x, y;
}p1;  //before semicolon
Struct Point p2 //global declaration
void main()
{
struct Point p3;
// Local Variable -> The variable p3 is declared like a normal
variable
p3.x=43;
p3.y=65;
struct Point p4={10,20};

cout<<p3.x<<endl;
cout<<p3.y<<endl;

cout<<p4.x<<endl; cout<<p4.y<<endl;
}
```

Specifying a class :

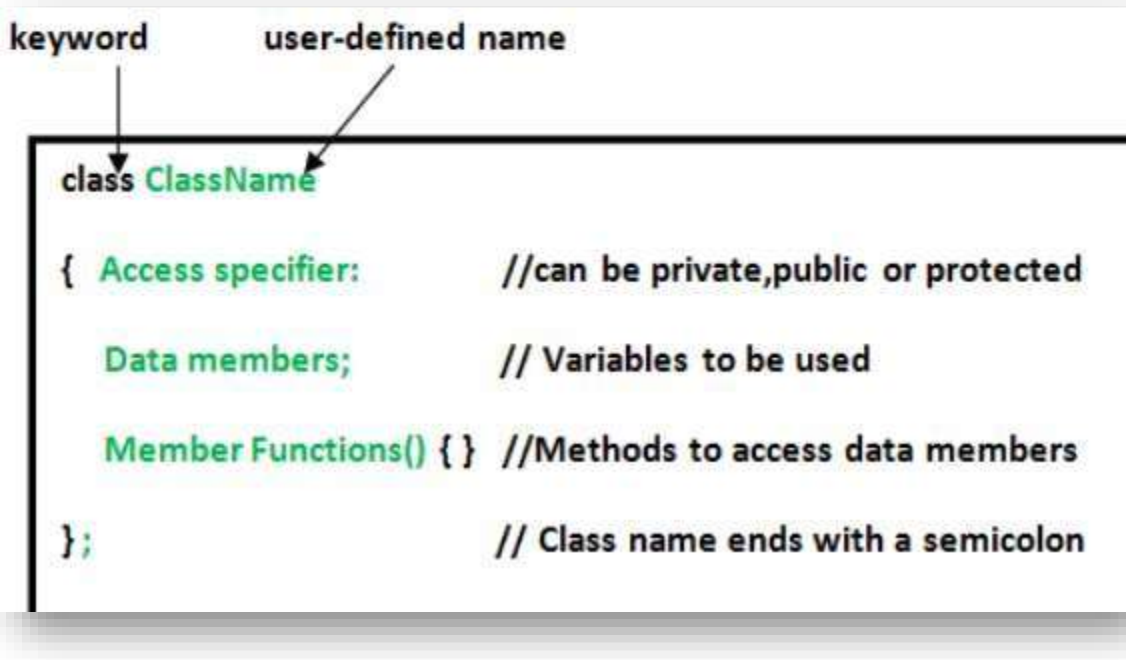
- **A class is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.**
- A C++ class is like a blueprint for an object.
  - For example: in real life, a car is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.
- Attributes and methods are basically variables and functions that belongs to the class.
- These are often referred to as "class members

- A Class is a user-defined data type that has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables together,
- these data members and member functions define the properties and behaviour of the objects in a Class.
- But we cannot use the class as it is.
- We first have to create an object of the class to use its features.
- An **Object** is an instance of a Class.
- ***Note:*** *When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.*

- **Defining Class in C++**
- **syntax:**

```
class ClassName
{
        access_specifier:

        // Body of the class
        //Data Members;
        //Member Functions();
};
```



```
class ClassName

{  Access specifier:          //can be private,public or protected

   Data members;              // Variables to be used

   Member Functions() { }     //Methods to access data members

};                            // Class name ends with a semicolon
```

Example :

```
class student
{
        public:                      // Access specifier
                int age;             // data member  / variable
                void print( )        // member function / method
                {
                        cout << "Hello";
                }
};
```

Access Specifiers :

- Classes have the same format as plain data structures, except that they can also include functions and have these new things called access specifiers.
- Access specifiers are one of the following three keywords:
- private, public or protected.
- **The public members:**
- A public member is accessible from anywhere outside the class but within a program.
- **The private members:**
- A private member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.
- <u>By default all the members of a class would be private.</u>
- **The protected members:**
- A protected member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

Creating object

- In C++, Object is a real world entity.
    - for example, chair, car, pen, mobile, laptop etc.
- In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.
- Object is a runtime entity, it is created at runtime.
- Object is an instance of a class. All the members of the class
- can be accessed through object.
- An object is simply a variable of its type (class). Therefore
- creating an object is much similar to declaring a variable.
- For example, if you want to create a variable of **int type, you would write:** int **i;**
- Same as you can create object of class **student we created earlier now create object of class student.**
- **Syntax :**
    - ClassName ObjectName;
- **Example :**
    - students1, **or student s1, s2, s3;**

Accessing Data Members and Member Functions

- The data members and member functions of the class can be accessed using the dot('.') operator with the object.

- For example, if the name of the object is *obj* and you want to access the member function with the name *printName( )* then you will have to write:

- *obj.printName( )*

Example :

```
#include<iostream.h>
#include<conio.h>
class demo
{
    public:
    int age;
    void print()
    {
            cout<<age;
    }
};
void main()
{
    string a;
    clrscr();
    demo d;
    d.age=22;
    d.print();
    getch();
}
```

Local classes

- A class which is declared inside a function or block is called local class.
- A local class name can only be used in its *function and not outside it*.
- the methods of a local class must be defined *inside the class* only.
- A local class *<u>cannot</u> have static data members* but it *<u>can</u> have static functions*.
- **Syntax :**

```
return_type function_name()
{
        class cls_name
        {
                .....
                .....
        };

        class_name object_name; object_name.data_members_name;
        //member function call
}
main()
{
        function_call();
}
```

```
#include<iostream.h>
 #include<conio.h>
void function()
{
cout<<"UDF";
class demo
{
        public:
        void cls_fun()
        {
                cout<<"\nThis is local class";
        };
demo d;
d.cls_fun();
}
void main()
{
clrscr();
function();
getch();
}
```

Nested class:

- A nested class is a class that is declared in another class.
- The class defined inside the class is known as inner class and the class in which a class is defined is known as outer class.
- The nested class is also a member variable of the enclosing class and has the same access rights as the other members.
- However, the member functions of the enclosing class have no special access to the members of a nested class.

- **Syntax :**

```
class OuterClass
{
    class InnerClass
    {
        //Code
    };
};
```

Example :

```
class inner
{

};
```

```
#include<iostream.h>
#include<conio.h>
class outer
{
    public:
    void out_fun()
    {
            cout<<"\nouter";
    }
    class inner
    {
            public:
            void in_fun()
            {
                        cout<<"\ninner class";
            }
    };
};
void main()
{
    clrscr();
    outer o;
    outer::inner i;
    o.out_fun();
    i.in_fun();
    getch();
```

Defining member function

- A Member function is a function that is declared as a member of a class. It is declared inside the class in any of the visibility modes like : public, private, and protected, and it can access all the data members of the class.
- The functions can be defined at two places:

1. Inside the class
2. Outside the class

- If the member function is defined inside the class definition, it can be defined directly inside the class.
- If we want to defined outside the class definition , we need to use the scope resolution operator (::) to declare the member function in C++ outside the class.
- The main aim of using the member function is to provide modularity to a program, which is generally used to improve code reusability and to make code maintainable.

Defining member function

Member Function Inside the Class :

- If you want to declare the function body Inside the class.
- There is no need to function Prototype.
- It will take automatically from function definition.
- A member function will be called using a dot operator (.) on a object where it will manipulate data related to that object .

Syntax for Member Function inside the Class :

```
class className
{
public:
    // Member function 1
    returnType1 functionName1(arguments1,..)
    {
            /* Function Definition
                        ………….
                        ………….
            */
    }
    // Member function 2
    returnType2 functionName2(arguments1,..)
     {
            /* Function Definition
                        ………….
                        ………….
            */
    }

};
```

Example for Member Function inside the Class :

```
class data
{
        int x;
        int y;
        public:
                void assign(int a,int b)
                {
                        x=a;
                        y=b;
                }
                void display()
                {
                        cout<<x<<endl;
                        cout<<y<<endl;
                }
};
void main()
{
        data d;
        d.assign(10,43);
        d.display();
}
```

Syntax for Member Function Outside the Class :

```
class className{
public/private:
  returnType  memberFunctionName (arguments); //prototype only
};

returnType className :: memberFunctionName (arguments)
{
   /* Statements
          .............
          .............
          .............
   */
}

void main(){ className
  object;

  object.memberFunctionName(arguments);
}
```

Example for Member Function

Outside the Class :

```
#include<iostream.h>
#include<conio.h>
class data  {
        int x;
        int y;
        public:
        void assign(int,int);
        void display();
};
void data::assign(int a,int b)
{
        x=a;
        y=b;
}
void
data::display()
{
        cout<<x<<endl;
        cout<<y<<endl;
}
void main()
{
        clrscr();
        data d;
        d.assign(10,43);
        d.display();
        getch();
}
```

Nesting of member functions :

- A member function of a class can be called only by an object of that class using a dot operator.
- If a member function calls another member function of its class, it is known as nesting of member functions.
- A member function can be called by using its name inside another member function of the same class.
- When a function calls another member function of its own class, it does not need to use dot (.) operator to call it.

Example :

```
#include<iostream.h>
#include<conio.h>
class data
{
        void fun()
        {
                 cout<<"hello function"<<endl;
        }
        public:
                 void display()
                 {
                          fun();        //this is nesting of function
                 }

};

void main()
{
   clrscr();
        data d;
        d.display();
        getch();
}
```

Private Member Functions :

- Generally the member variables are kept private and the functions are kept public so that the object cannot access the variables but can call the functions.
- If we make a member function private, it cannot be called by its object.
- So we can restrict access to a member function if we don’t want to allow objects to directly call it.
- The private member function can be called by its member function without using objects.

Example private member function :

```
#include<iostream.h>
#include<conio.h>
class data
{
        private:
        int x;
        int y;
        void assign(int a,int b)
                {
                        x=a;
                        y=b;
                }
        public:
                void display()
                {
                        assign(32,45);
                        cout<<x<<endl;
                        cout<<y<<endl;
                }
};

void main()
{
        clrscr();
        data d;
        d.display();
        getch();
}
```

Making Outside Function inline :

- C++ also allows you to declare the inline functions within a class.

- These functions need not be explicitly defined as inline as they are, by default, treated as inline functions.

- All the features of inline functions also apply to these functions.
- However, if you want to explicitly define the function as inline, you can easily do that too.

- You just have to declare the function inside the class as a normal function and define it outside the class as an inline function using the inline keyword.

Example :

```
class demo_cls
{
public:
    int func(int n); // function declaration inside the
   class as inline
};
inline int demo_cls::func(int n) // defining the
   function as inline using inline keyword
                  return n+100;
}
void main()
{
          demo_cls  d;
         cout<<"sum is "<<d.func(101);
}
```

Arrays within a class :

- Arrays can be declared as the members of a class.
- The arrays can be declared as private, public or protected members of the class.
- **Syntax :**
  - class class_name
  - {
    - access modifier:
      - data_type array[size];
  - }

- **Example :**

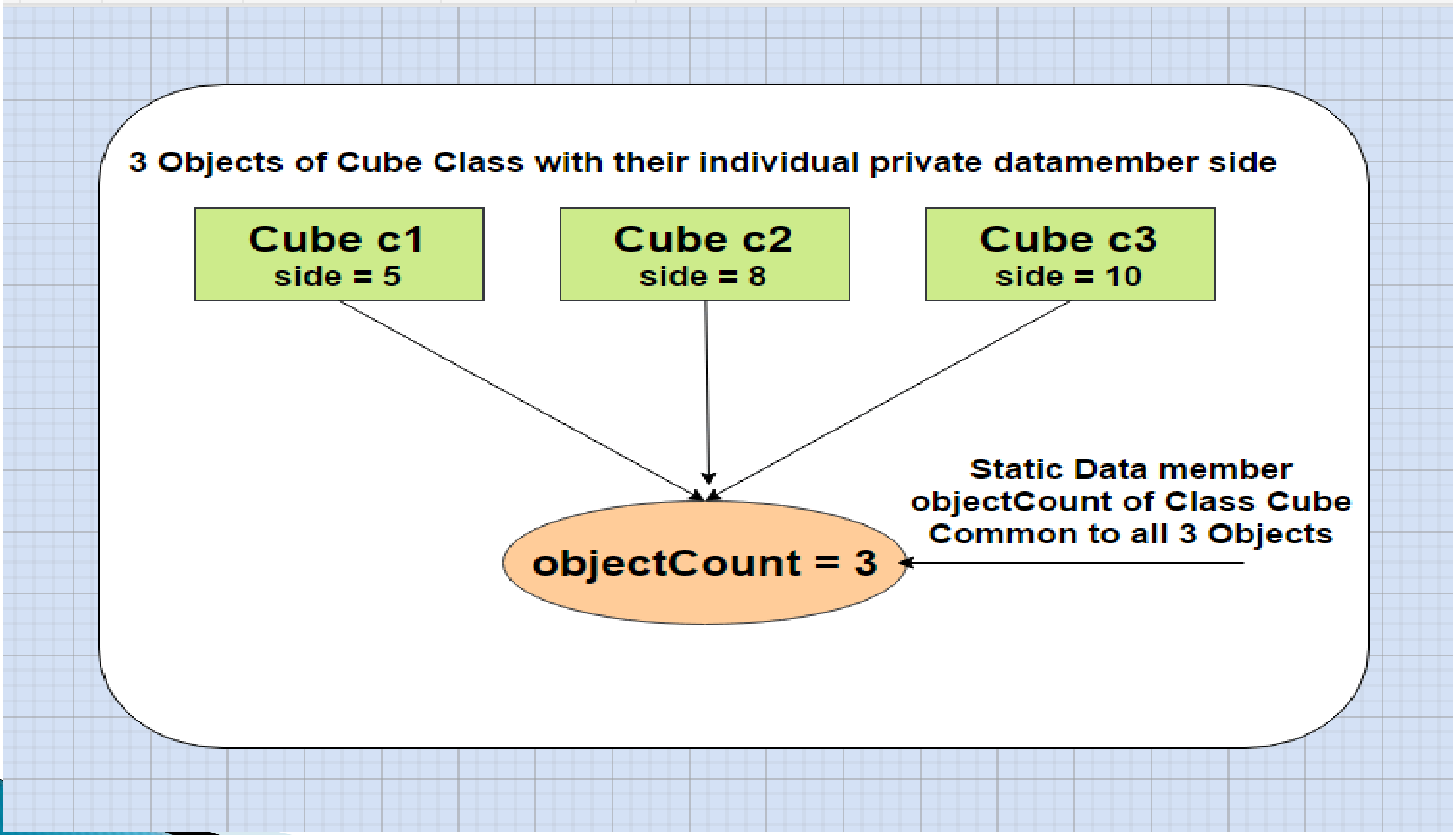
```
#include<iostream.h>
#include<conio.h>
class demo
{
   int arr[5];
   public :
         void value();
         void show();
};
void demo::value()
{
   cout<<"enter Value for Array
   ";
   for(int i=0;i<5;i++)
   {
         cin>>arr[i];
   }
}
```

```
void
demo::show()
{
   cout<<"values of array is ";
   for(int i=0;i<5;i++)
   {
         cout<<arr[i]<<endl;
   }
}


void main()
{
   clrscr();
   demo d;
   d.value();
   d.show();
   getch();
}
```

Memory Allocation of Objects :

- When the object of the class is created, memory is allocated to the object according to the member variable of the class.
- But the memory space for the member function is allocated when they are defined.
- So the complete memory allocation is done when an object is created.
- Individual memory is allocated for each object created.
- But the common memory is allocated for the member functions means no separate memory space is allocated for member function.



3 Objects of Cube Class with their individual private datamember side

Cube c1
side = 5

Cube c2
side = 8

Cube c3
side = 10

objectCount = 3

Static Data member
objectCount of Class Cube
Common to all 3 Objects

Static Data Member

- Static data members are class members that are declared using **static** keywords.
- A static member has certain special characteristics which are as follows:
- Static variable was initialized with *zero* value when object is created first
- time.
- 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 initialized before any object of this class is created, even before the main
- starts outside the class itself.
- It is visible can be controlled with the class access specifiers.
- Its lifetime is the entire program.
- The static variable is connected with all the object of class that why we can
- say the static variable is the *CLASS VARIABLE* in OOP.
- **Syntax**
    - className
    - {
        - **static** data_type data_member_name;
        - [illegible]
    - }

Example :

```
#include<iostrea.h>
#include<conio.h>
class demo
{
    static int a;
    public:
    void fun()
    {
            a++;
            cout<<"\nvalue of a is \t"<<a;
    }
};
int demo::a;
void main()
{
    demo d1,d2,d3;
    clrscr();
    d1.fun();
    d2.fun();
    d3.fun();
    getch();
}
```

Static Member Function :

- Static Member Function in a class is the function that is declared as static
- A static member function is independent of any object of the class.
- A static member function can be called even if no objects of the class
- exist.
- A static member function can also be accessed using the class name
- through the scope resolution operator.

- A static member function can access static data members and static member functions inside or outside of the class.

- Static member functions have a scope inside the class and cannot access the current object pointer.

- You can also use a static member function to determine how many
- objects of the class have been created.
- **The reason we need Static member function:**
- Static members are frequently used to store information that is shared by all objects in a class.

- For instance, you may keep track of the quantity of newly generated objects of a specific class type using a static data member as a counter.

- This static data member can be increased each time an object is generated to keep track of the overall number of objects.

Static Member Function :

```
#include<iostream.h>
#include<conio.h>
class demo
{
    public:
    static int a;
    int b;
    void  fun()
    {
            cout<<"\nvalue of static a is \t"<<a;
            cout<<"\nvalue of normal b is \t"<<b;
    }
    static  void  f()
    {
            demo ds;
            cout<<"\nf()\nvalue of static a :"<<a
            <<endl<<"value of normal b :"<<ds.b
            <<endl;
            //a is static member
            //b is non-member,
            //to use it we need to use object of class
    }
};
int demo::a;
//int demo::a=10;
```

```
void  main()
{
    demo obj;
    clrscr();
    //assign value of data member
     obj.b=20;//normal data member
    //obj.a=29;//static data member
    //demo::a=10;//static data member

    //member function calling
    cout<<"\ncall normal member FUN";
    obj.fun();

    cout<<"\ncall static membern fun without
    object\n";
    demo::f();

    cout<<"\nstatic member call with object\n";
    obj.f();

    getch();
}
```

Arrays of Object :

- In C++, an **array of objects** is a collection of objects of the same class type that are stored in contiguous memory locations.
- Since each item in the array is an instance of the class, each one's member variables can have a unique value.
- This makes it possible to manage and handle numerous objects by storing them in a single data structure and giving them similar properties and behaviours.
- We can think of array of objects as a single variable that can hold multiple values.
- Each value is stored in a separate element of the array, and each element can be accessed by its index.
- Arrays in C++ are typically defined using square brackets [ ] after the type.
- The index of the array, which ranges from 0 to n - 1, can be used to access each element.

- class className
- {
  - //variables and functions
- };
- className arrayObjectName[arraySize];

- **className** is the name of the class that the object belong tp.
- **arrayName** is the name of the array of objects.
- **arraySize** is the number of objects in the array or the size of array, specified as a constant expression

Example :

```
#include<iostream
.h>
#include<conio.h>
class stud
{
    int roll;
    char name[30];
    public:
    void get_data()
    {
      cout<<"Enter Roll Number : ";
      cin>>roll;
      cout<<"Enter Name : ";
      cin>>name;
    }
    void show_data()
    {
      cout<<endl<<"Roll number : "
    <<roll;
     cout<<endl<<"Name : "<<name;
          }

};
void main()
{
    clrscr();
    stud obj[2];
    for(int i=0;i<2;i++)
    {
          obj[i].get_data();
    }
    for(int j=0;j<2;j++)
    {
          obj[j].show_data();
    }
    getch();
}
```

Object as Function Argument :

- We have seen examples of member functions having arguments. Just like any other normal variables, we can also pass object as function arguments.
  - ▫ *A copy of the entire object is passed to the function.*
  - ▫ *Only the address of the object is transferred to the function.*
- As the objects are the variables of type class, you have to specify the class name as the type of the object arguments.
- In previous chapter, we discussed about call by value and call by reference functions.
- The same concept applies to the functions having objects as arguments.
- If we pass address of the object to the function it is called by reference. So any changes made on the object will also affect the passing object values.
- But if you pass object normally it is called by value. So the changes made on the object will not reflect to the original object.

Example :

```
#include<iostream
.h>
#include<conio.h>
class demo_cls
{
   int a;
   public:
   void data(int);
   void sum(demo_cls,demo_cls);
};

void demo_cls::sum(demo_cls
   a_obj1,demo_cls a_obj2)
{
   cout<<"Sum of 2 object is :
   "<<a_obj1.a+a_obj2.a;
}
```

```
void demo_cls::data(int x)
{
   a=x;
}

void main()
{
   clrscr();
   demo_cls obj1,obj2,obj3;
   obj1.data(10);
   obj2.data(20);

   obj3.sum(obj1,obj2);
   getch();
}
```

Friendly function in C++ :

- Normally, the private members cannot be accessed by external functions.
- Means a function which is not a member function of the class
- cannot have access to the private member (variable and function) of the class.
- C++ introduces a kind of functions known as friend functions which behaves like
- friend of the class.
- We can define a function friendly to one or more classes allowing the function to access the public as well as *private* / *protected* member of all the class to which it is declared as friend.

- Byusing the keyword **friend** compiler knows the given function is a friend
- For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.
- **Declaration of friend function :**
  - **class** class_name
  - {
    - **friend** data_type function_name(argument/s)// syntax of friend function.
  - };

- In the above declaration, the friend function is preceded by the keyword friend.

- The function canbe defined anywhere in the program like a normal C++

- The function definition *does not use* either the keyword **friend or scope resolution operator**.

Characteristics of friend function

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object(UDF).
- It cannot access the member names directly and has to use an object name and **dot** membership operator with the member name.
  (obj_name**.**datamember)
- It can be declared either in the private or the public part.

Example :

```
#include<iostream.h>
#include<conio.h> class
Point
{
int x;
 int y;
 public:
friend void sum_fun(Point);
void add_data(int x1 = 0, int y1 = 0)
{
 x = x1;
 y = y1;
}
void display()
{
 cout<<"x = "<< x <<"\n";
 cout<<"y = "<< y <<"\n";
}

};
```

```
void sum_fun(Point obj1)
{
            int s;
            s=obj1.x+obj1.y;
            cout<<"Sum of 2 numbers using friend
                        function : "<<s<<endl;
}
void main()
{
clrscr();
  Point p1;

  p1.add_data(5,3);

  cout<<"Point 1\n";
  p1.display();

  cout<<"The sum of the two points is:\n";
  p1.display();

  sum_fun(p1);
  getch();
}
```