### Member Functions in Classes

Member functions are the functions, which have their declaration inside the class definition and works on the data members of the class. The definition of member functions can be inside or outside the definition of class.

If the member function is defined inside the class definition it can be defined directly, but if its defined outside the class, then we have to use the scope resolution :: operator along with class name alng with function name.

*Example* :

class Cube

{

public:

int side;

int getVolume(); // Declaring function getVolume with no argument and return type int.

};

If we define the function inside class then we don't not need to declare it first, we can directly define the function.

class Cube

{

public:

int side;

int getVolume()

{

return side\*side\*side; //returns volume of cube

}

};

But if we plan to define the member function outside the class definition then we must declare the function inside class definition and then define it outside.

class Cube

{

public:

int side;

int getVolume();

}

int Cube :: getVolume() // defined outside class definition

{

return side\*side\*side;

}

The maine function for both the function definition will be same. Inside main() we will create object of class, and will call the member function using dot . operator.

int main()

{

Cube C1;

C1.side=4; // setting side value

cout<< "Volume of cube C1 ="<< C1.getVolume();

}

Similarly we can define the getter and setter functions to access private data members, inside or outside the class definition.

### Types of Member Functions

We already know what member functions are and what they do. Now lets study some special member functins present in the class. Following are different types of Member functions,

1. Simple functions
2. Static functions
3. Const functions
4. Inline functions
5. Friend functions

#### Simple Member functions

These are the basic member function, which dont have any special keyword like static etc as prefix. All the general member functions, which are of below given form, are termed as simple and basic member functions.

return\_type functionName(parameter\_list)

{

function body;

}

#### Static Member functions

Static is something that holds its position. Static is a keyword which can be used with data members as well as the member functions. We will discuss this in details later. As of now we will discuss its usage with member functions only.

A function is made static by using static keyword with function name. These functions work for the class as whole rather than for a particular object of a class.

It can be called using the object and the direct member access . operator. But, its more typical to call a static member function by itself, using class name and scope resolution :: operator.

*Example*:

class X

{

public:

static void f(){};

};

int main()

{

X::f(); // calling member function directly with class name

}

These functions cannot access ordinary data members and member functions, but only static data members and static member functions.

It doesn't have any "this" keyword which is the reason it cannot access ordinary members. We will study about "this" keyword later.

#### Const Member functions

We will study Const keyword in detail later, but as an introduction, Const keyword makes variables constant, that means once defined, there values can't be changed.

When used with member function, such member functions can never modify the object or its related data members.

//Basic Syntax of const Member Function

void fun() const {}

#### Inline functions

All the member functions defined inside the class definition are by default declared as Inline. We will study Inline Functions in details in the next topic.

#### Friend functions

Friend functions are actually not class member function. Friend functions are made to give **private** access to non-class functions. You can declare a global function as friend, or a member function of other class as friend.

*Example :*

class WithFriend

{

int i;

public:

friend void fun(); // Global function as friend

};

void fun()

{

WithFriend wf;

wf.i=10; // Access to private data member

cout << wf.i;

}

int main()

{

fun(); //Can be called directly

}

Hence, friend functions can access private data members by creating object of the class. Similarly we can also make function of other class as friend, or we can also make an entire class as friend class.

class Other

{

void fun();

};

class WithFriend

{

private:

int i;

public:

void getdata(); // Member function of class WithFriend

friend void Other::fun(); // making function of class Other as friend here

friend class Other; // making the complete class as friend

};

When we make a class as friend, all its member functions automatically become friend functions.

Friend Functions is a reason, why C++ is not called as a pure Object Oriented language. Because it violates the concept of Encapsulation.