

## Methods (Member Functions) in classes

Member functions or methods are used to work on the data members of the class. The methods can be defined inside or outside of the class. If the methods are defined inside the class definition then it can be defined directly, but if defined outside the class then we have to use scope resolution operator '::' along with class name and function name.

If we define the function inside the class then no need declare it first, we can directly define the function inside the class as shown in below.

```
class Account
{
    string str;
public:
    void print_inside()    //Inside the class defination
    {
        str = "Hello! - In";
        cout << "Inside class: " << str << endl;
    }
};
```

If we define the function outside the class definition then must declare the function inside the class definition and then define the function outside the class as shown in below.

```
class Account
{
    string str;
public:
    void print_outside();    // function declration for outside class function defination
};
void Account::print_outside () //Outside the class defination
{
    str = "Hello!!! - Outside";
    cout << "Outside class: " << str << endl;
}
```

The following examples illustrates the defining the member function inside the class and outside the class.

```

main.cpp X
Account print_outside()
#include <iostream>
#include <string>
using namespace std;

class Account
{
    string str;
public:
    void print_outside(); // function decleration for outside class function definition
    void print_inside() //Inside the class defination
    {
        str = "Hello! - In";
        cout << "Inside class: " << str << endl;
    }
};

void Account::print_outside () ///Outside the class definition
{
    str = "Hello!!! - Outside";
    cout << "Outside class: "<< str << endl;
}

int main()
{
    Account obj;
    obj.print_inside();
    obj.print_outside();
    cin.get();
    return 0;
}

```

G:\CPP\first program\Debug\first program.exe

```

Inside class: Hello! - In
Outside class: Hello!!! - Outside

```

Example:

```

#include <iostream>
using namespace std;
class Account {
private: // private access specifier
    int acc_no;
    float balance;
public: // public access specifier
    void acc_details()// function definition inside the class
    {
        cout << "enter your account number:";
        cin >> acc_no;
        cout << "enter your balence:";
    }
}

```

```
        cin >> balance;
        cout << "Balance:" << balance << endl;
    }

    float deposit();//function declaration for outside class functions definition
    float withdraw();//function declaration for outside class functions definition
};

float Account::deposit();// function definition outside the class
{
    float amount;
    cout << "Enter the deposit amount:";
    cin >> amount;
    balance = balance + amount;
    return balance;
}

float Account::withdraw()                // function definition outside the class
{
    float amount;
    cout << "Enter withdraw amount:";
    cin >> amount;
    balance = balance - amount;
    return balance;
}

int main()
{
    Account obj1;
    obj1.acc_details();
    cout << "Balance after depositing:" << obj1.deposit() << endl;
    cout << "Balance after withdrawing:" << obj1.withdraw() << endl;
    system("pause");
    return 0;
}
```

```
enter your account number:21171
enter your balance:20000
Balance:20000
Enter the deposit amount:5000
Balace after depositing:25000
Enter withdraw amount:2000
Balance after withdrawing:23000
Press any key to continue . . .
```

Figure1: Output of the program

### Types of member functions

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

### Simple member functions

These are the simple basic member functions which are not required any special keyword to indicate as prefix. The syntax for simple member functions given below,

Return\_type Function\_Name(argument list)

```
{
    statements;
}
```

## **Inline member functions**

The Inline functions are enhancement feature provide in C++ programming language to increase the execution time of a program. Inline functions are similar to macros. The preprocessors are not used in C++ because it has some drawbacks. Inline functions are actual functions, which are copy the code every where the function is defined during the compilation time. All the member functions defined inside the class definition are by default inline functions, but whereas outside the class, the function are inline by using 'inline' keyword with the function name. If any change to an inline function then the program must be recompiled because the compiler needs to replace all the code at each place otherwise it will continue with old functionality.

### **Syntax:**

```
inline void function_name()
{
    return 0;
}
```

### **Important points about inline functions:**

- Inline functions increase the speed of the program by avoiding function calling overheads.
- Inline functions must be small to have better efficiency otherwise it increases the size of the code which affects the speed also.
- The large functions must be defined outside the class definition using scope resolution '::' operator, because if we define such function inside the class then they become inline function automatically which affects the performance.
- The compiler is unable to perform inlining if the function is too complicated, so must avoid big looping conditions inside the inline functions.
- Inline functions are kept in the symbol table by the compiler and all the calls for such function taken care at the compile time.

Example for inline functions inside the class:

```
#include <iostream>

using namespace std;

class A
{
private:
    int x;
public:
    void get_value() // default inline function in C++
    {
        cout << "Enter a Value:";
        cin >> x;
    }

    void print_value() // default inline function in C++
    {
        cout << x << endl;
    }
};

int main()
{
    A obj;
    obj.get_value();
    obj.print_value();
    system("pause");
    return 0;
}
```

A screenshot of a terminal window with a black background and white text. The text shows the output of the C++ program: "Enter a Value:100", followed by "100" on the next line, and "Press any key to continue . . ." on the third line. A large, semi-transparent watermark "www.sakshieducation.com" is overlaid diagonally across the entire page, including the screenshot.

Figure2: Output of the program

In the above program 'get\_value()' and 'print\_value()' are by default inline function inside class definition and they are defined to access the private data members of the class. If we want to use inline functions outside the class then must be use the 'inline' keyword before the function name as shown below.

Example for inline functions outside the class:

```
#include <iostream>

using namespace std;

class A
{
private:
    int x;
public:
    void get_value();
    void print_value();
};

inline void A :: get_value()    // inline function outside the class
{
    cout << "Enter a Value:";
    cin >>x;
}

inline void A :: print_value() // inline function outside the class
{
    cout << x << endl;
}

int main()
{
    A obj;
    obj.get_value();
    obj.print_value();
    system("pause");
    return 0;
}
```

A screenshot of a terminal window with a black background and white text. The text shows the output of the program: 'Enter a Value:1234' followed by '1234' on the next line, and 'Press any key to continue . . .' on the third line. A large, semi-transparent watermark 'www.sakshieducation.com' is overlaid diagonally across the image.

Figure3: Output of the program

## **Friend member function**

A non member function cannot access an objects private and protected members and sometimes it force programmers to write long and complex codes. This can be overcome by

using friend function and friend class. The friend member functions are not class member functions but they are made to give private access to non class functions. These are the functions that can be made friendly with both the classes, so that these functions can have access to the private members of these classes.

If a function is defines as a friend function then the private and protected members of class can be accessed from that function. The declaration of friend function should be done inside the body of class either in private section or in public section starting with keyword friend. The compiler knows the given function is a friend function by its keyword 'friend'.

Syntax:

```
class ClassName
{
    private:
        data members;
    public:
        friend return_type FunctionName( arguments );
};
```

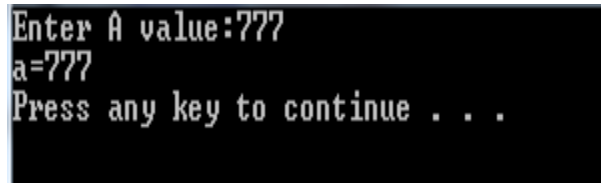
Example program working with friend function:

```
#include <iostream>
using namespace std;
class class1
{
private:
    int a;
public:
    void get_value()
    {
        cout << "Enter A value:" ;
        cin >> a;
    }
    friend void print_value(class1); // friend function declaration
};

void print_value(class1 obj1) // friend function defination
{
    cout << "a=" << obj1.a << endl;
}
```



```
int main()
{
    class1 obj1;
    obj1.get_value();
    print_value(obj1);
    system("pause");
    return 0;
}
```



Enter A value:777  
a=777  
Press any key to continue . . .

Figure4: Output of the program

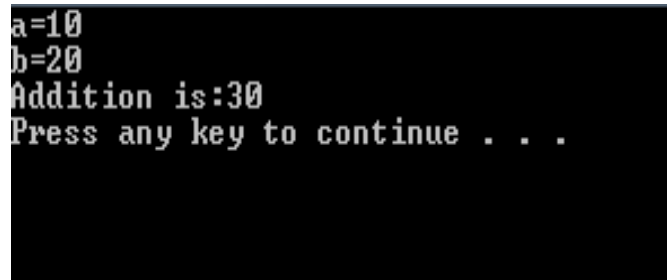
In the above example we are getting some value from user using normal member function and displaying this value on the screen using friend function.

Example program working on objects of two different classes using friend function:

In the below example the friend function for addition is declared in both the classes and getting values for 'a' from class1 and for 'b' class2 respectively, addition is performed on these numbers in the friend function definition.

```
#include <iostream>
using namespace std;
class class1; // forward declaration
class class2
{
private:
    int b;
public:
    void get_value(int y)
    {
        b = y;
    }
    void print_value()
    {
        cout << "b=" << b << endl;
    }
}
```

```
    }  
    friend void Addition(class1, class2); // friend function declaration  
};  
  
class class1  
{  
private:  
    int a;  
  
public:  
    void get_value(int x)  
    {  
        a = x;  
    }  
  
    void print_value()  
    {  
        cout << "a=" << a << endl;  
    }  
  
    friend void Addition(class1, class2); // friend function declarartion  
};  
  
void Addition(class1 obj1, class2 obj2) // friend function defination  
{  
    cout << "Addition is:" << obj1.a + obj2.b << endl;  
}  
  
int main()  
{  
    class1 obj1;  
    class2 obj2;  
    obj1.get_value(10);  
    obj1.print_value();  
    obj2.get_value(20);  
    obj2.print_value();  
    Addition(obj1,obj2);  
    system("pause");  
    return 0;  
}
```



```
a=10  
b=20  
Addition is:30  
Press any key to continue . . .
```

Figure5: Output of the program

## Friend class

A class can be friend of another class using 'friend' keyword as similar to friend function. When a class is made a friend class then all the member functions of that class becomes friend functions.

```
class1
{
    public:
        functions();
        friend class2; // class2 is friend class
};
class2
{
    public:
        functions();
}
```

In the above example the member functions of class2 will be friend functions of class1 so that any member functions of class2 can access the private data of class1.

## Static member functions

A function can be static member function 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 direct member access operator '.' (Dot), but more typically the static member function can be accessed itself using class name and scope resolution '::' operator. These functions cannot access ordinary data members and member functions, but only static data members and static member functions.

- We cannot have the static member function and non static member function with same name and type arguments.

- A static member function cannot be declared with const and volatile type qualifiers.
- A static member function cannot declare as a virtual function.
- A static member function cannot have this pointer.

Syntax:

```
static Return_data_type Function_Name()

{
    statements;
}
```

Example:

```
#include <iostream>

using namespace std;

class rectangle
{
private:
    int Length, Width;
    static int count; // static member declaration
public:
    void get_value(int, int );
    void print_values();
    void area();

    static int get_count() // static member function defination
    {
        return count;
    }
};

void rectangle :: get_value(int l, int w)
{
    Length = l;
    Width = w;
    count ++;
}

void rectangle :: print_values()
{
    cout << "Length is:" << Length << endl;
```

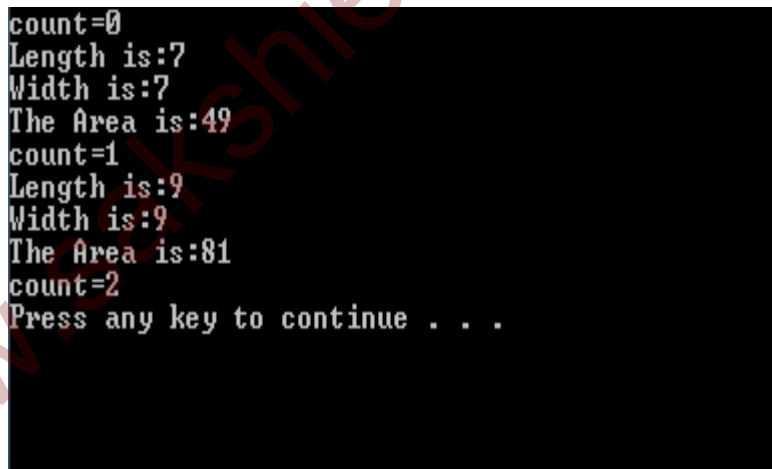
```

        cout << "Width is:" << Width << endl;
    }
void rectangle :: area()
{
    cout << "The Area is:" << Length * Width << endl;
}

int rectangle::count = 0; // static member initialization

int main()
{
    rectangle obj,obj2;
    cout << "count=" << rectangle::get_count() << endl; // calling static member
function
    obj.get_value(7,7);
    obj.print_values();
    obj.area();
    cout << "count=" << rectangle::get_count() << endl; // calling static member
function
    obj.get_value(9,9);
    obj.print_values();
    obj.area();
    cout << "count=" << rectangle::get_count() << endl; // calling static member
function
    system("pause");
    return 0;
}

```



```

count=0
Length is:7
Width is:7
The Area is:49
count=1
Length is:9
Width is:9
The Area is:81
count=2
Press any key to continue . . .

```

Figure6: Output of the program

## Const member functions

Const keyword makes the variable constant, which means if the variable once defined then there values will be constant and cannot be changed throughout the program. A function

becomes const when the const keyword is used with function declaration. Const keyword with member functions can never modify the object or its related data members. It is useful to make as many functions const as possible so that accidental changes to objects are avoided. When a function is declared as const, then it can be called by any type of object but where as non const functions can only be called by non const objects.

Syntax:

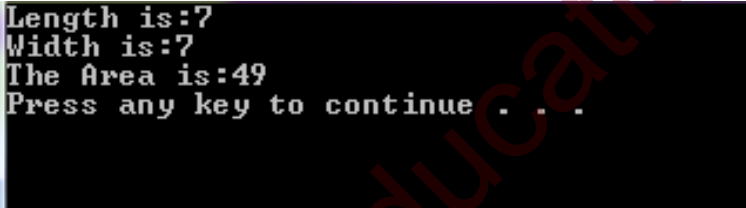
```
Return_type Function_Name const ()  
{  
    statements;  
}
```

Example:

```
#include <iostream>  
using namespace std;  
  
class rectangle  
{  
private:  
    int Length, Width;  
  
public:  
    void get_value(int, int );  
    void print_values();  
    void area() const; // Const member function declaration  
};  
  
void rectangle :: get_value(int l, int w)  
{  
    Length = l;  
    Width = w;  
}  
  
void rectangle :: print_values()  
{  
    cout << "Length is:" << Length << endl;  
    //Length = 10; ---> Here we can access the object and can be  
change the member variable  
    //cout << "Length is:" << Length << endl;  
    cout << "Width is:" << Width << endl;  
}
```

```
void rectangle :: area() const // const member function defination
{
    //Length = 2; ----> Here we cannot change the member variables becuz it is
const member function
    cout << "The Area is:" << Length * Width << endl;
}
```

```
int main()
{
    rectangle obj;
    obj.get_value(7,7);
    obj.print_values();
    obj.area();
    system("pause");
    return 0;
}
```

A screenshot of a terminal window showing the output of a C++ program. The text is white on a black background. It displays: Length is:7, Width is:7, The Area is:49, and Press any key to continue . . . .

```
Length is:7
Width is:7
The Area is:49
Press any key to continue . . . .
```

Figure7: Output of the program