**Lab 09**

**Friend Function and Classes**

1. **Objectives**

To familiarize the students with the concept of friend functions, friend classes and how to implement them in object oriented programming approach.

1. **Outcome**

After this lab the students should be able to access private or protected data from non-member functions i.e. friend functions. They should learn how to make a class to access private and protected members of another class using the concept of friend classes.

1. **Introduction** 
   1. **Friend Function:**

The concepts of encapsulation and data hiding dictate that nonmember functions should not be able to access an object’s private or protected data. The policy is, if you’re not a member, you can’t get in. However, there are situations where such rigid discrimination leads to considerable inconvenience. So, there is mechanism built in C++ programming to access private or protected data from non-member functions.

This is done using a friend function or/and a friend class. The member functions of a class can all be made friends at the same time when you make the

entire class a friend.

* 1. **Access specification and syntax:**

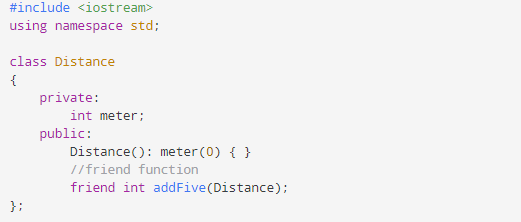
If a function is defined as a friend function then, the private and protected data of a class can be accessed using the function.

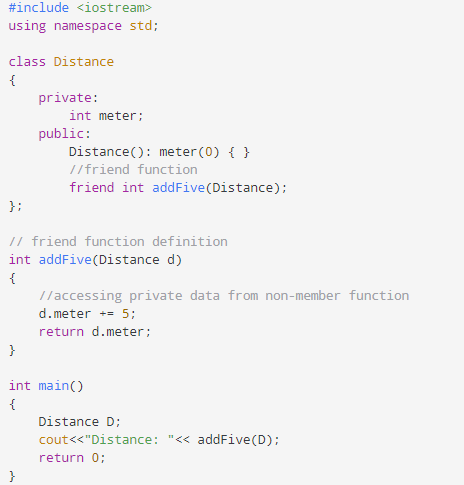
The complier knows a given function is a friend function by the use of the **keyword friend.**

For accessing the data, the **declaration** of a friend function should be made inside the body of the class (can be anywhere inside class either in private or public section).

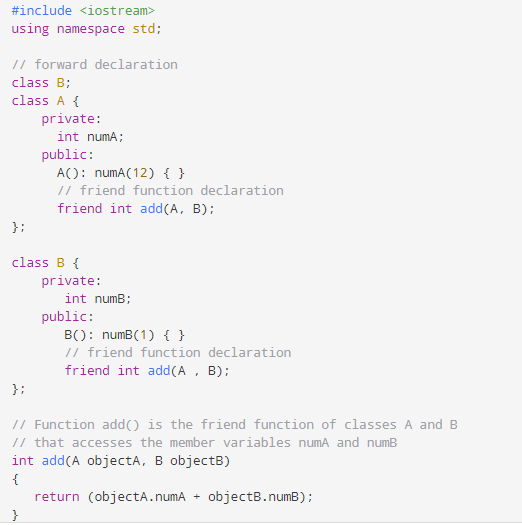
A friend function of a class is **defined** outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

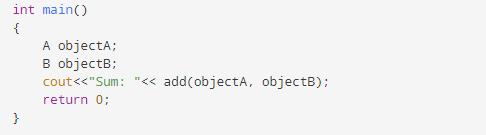
1. **Examples :**
   1. **This example demonstrates the how friend function can access private data member.**





* 1. **This example demonstrates the Addition of members of two different classes using friend Function.**





**4.3 This example demonstrate the functioning of friend classes.**

#include<iostream.h>

class Rectangle

{

int L,B;

public:

Rectangle()

{

L=10;

B=20;

}

friend class Square; //Statement 1

};

class Square

{

int S;

public:

Square()

{

S=5;

}

void Display(Rectangle Rect)

{

cout<<"\n\n\tLength : "<<Rect.L;

cout<<"\n\n\tBreadth : "<<Rect.B;

cout<<"\n\n\tSide : "<<S;

}

};

void main()

{

Rectangle R;

Square S;

S.Display(R); //Statement 2

}

1. **Lab Tasks**

**5.1** Write a program to convert a decimal number into its binary equivalent with a friend function.

**5.2** Make a class distance with data members feet and inches. Now , suppose we want a function that will square (multiply by itself) an object of the Distance class and return the result in square feet, as a type float. Using the concept of friend function implement this Distance class.

**5.3** Write a program to convert temperature from Celsius to Fahrenheit scale by creating two different classes for each temperature scale and making both classes friend of each other.

1. **Post Lab Tasks :**

**6.1** In the Distance class, implemented in the in lab task, create an overloaded \* operator so that two distances can be multiplied together. Make it a friend function so that you can use such expressions as Wdist1 = 7.5 \* dist2;

You’ll need a one-argument constructor to convert floating-point values into Distance

values. Write a main() program to test this operator in several ways.