Friend Functions

A friend function is a function that is **not** a member function but has access to the private members of said class. This is done via the keyword **friend** and the function declaration is placed in the **public** section of the class.

You can think of this concept as you owning a house (the class) and giving the key to the house to some friend so they have access to everything inside. This comes with a considerable deal of trust and should be avoided except in certain situations (like overloading the insertion and extraction operators).

Here is an example program of a friend function that adds two object's private member variables together and returns another object with its private member variable equal to the sum.

```
class Num
{
    public:
        Num() : x(0) {}
        Num(int x) : x(x) {}
        int getX() const { return x; }
        //Friend function returning an object of this class
        friend Num add(const Num &Obj1, const Num &Obj2);
    private:
        int x;
};
```

Inside the class definition is the friend function. The keyword **friend** is placed before the return type. In this case, the return type is **Num**. The rest of the function declaration is similar to every function declaration that you have seen. The name of the function is then stated, followed by the formal parameters, and lastly the semicolon.

```
//Defining the friend function
Num add(const Num &Obj1, const Num &Obj2)
{
    Num Temp;
    Temp.x = Obj1.x + Obj2.x; //All objects use dot operator
    return Temp;
}
```

Since a friend function is a global function and not formally a part of the class, the definition of the friend function does **not** include the type qualifier and scope resolution operator. It is defined simply as any other global function. Also keep note that the keyword **friend** does not appear here as well. Each object will be called by constant reference as no changes will be made to those objects. A friend function's arguments may be called by (const) reference or value.

Inside the function, a new object of type **Num** is declared and has its private member variable **x** set to the sum of **Obj1's** private member variable **x** and **Obj2's** private member variable **x**. Notice how all three objects need to utilize the **dot operator** to access their respective private member variable. In a friend function, there is to be no calling object as this isn't a member function. Once the calculation is complete, the **Temp** object is returned.

Please keep in mind that if you return an object and/or use pointers/dynamic memory/class member variables, you should follow the **Rule of Three**. See the **Rule of Three** section for more information. In our example, everything works as is due to only having a primitive type as the only private member variable.

```
int main()
{
    Num A(10), B(200);
    Num C;
    C = add(A,B); //Friend function invocation
    cout << C.getX() << endl; //Displaying object C's x member var
}
```

In this example, object **A** has an **x** value of 10 and object **B** has an **x** value of 200. Object **C** has been declared and is going to be the recipient of the friend function. Since the friend function returns type **Num**, the assignment statement is utilized. This is just like having an **int** variable assigned to the result of a function that returns some integer. Notice the function invocation is the same as any global function invocation. The friend function executes adding the appropriate **x** member variables. Object **C** will have an **x** member variable with the value of 210. This is what gets displayed during the cout statement.

Insertion/Extraction Operator

In order to properly use the **insertion operator (<<)** and **extraction operator (>>)**, you must define these functions as **friend** functions of your class. This is due to the fact that the first argument of these functions is an object from the **ostream** (ex: **cout**) class or the **istream** (ex: **cin**) class and the second argument is an object of your class. Overloaded functions may only be members of the first argument's class. Therefore, it must be a friend function of your class. See the **Operator Overloading** section for more info.

Example Extraction Friend Function

Definition) **friend istream& operator >>(istream& cin, ClassName& Obj);**
Declaration) **istream& operator >>(istream& cin, ClassName& Obj)**