Lab No.8 Friend Functions and Classes

8.1 Objectives of the lab:

Introducing the concepts of friendship such as

32 Friend functions
33 Friend classes

8.2 Pre-Lab

8.2.1 Friend Functions

1 A regular C-style function accessing the non-public members of the objects of a class to which it is a friend
   - Not a member function
   - Can access non-public members of the objects to which it is a friend
2 Declaration inside the class preceded by keyword **friend**
   - Keyword friend is not used with definition. //Compiler error
3 Definition outside the class but like the definition of normal C-style functions
4 **this** is NOT Available to friend Functions
5 A friend function is not called in the manner member functions are called. Friend function is called like normal C-style functions and takes the object of the class to which it is a friend
6 A friend can be declared in **public**, **private**, or **protected** portions.
   - Member access specifiers have no effect on friend function
   - Friend function is not a member
7 Friend functions are not inherited

8.2.2 Example

```
#include <iostream.h>
class test
{
private:
    int     data;
public:
    test(): data(0)
    {}

    void show()
    {       cout<<data<<endl;
    }

    friend void set_data(test &);
};
```

```
void set_data(test &obj)
{
    obj.data=3;
}

void main()
{
    test     t;
    set_data(t);     //called like normal function
    t.show();
}
```

8.2.3 Friend classes

- Not directly associated with the Class
- All the member functions of the friend class can access non-public data members of the original class
  - If class A is the friend of class B. all its member functions can access the nonpublic members of class B
- Declaration

```
class B
{
friend class A;
…
}
```

- Can be friend of more than one classes
- Violation of encapsulation
- SHOULD BE USED ONLY WHEN REQUIRED

8.2.4 Example

```
#include <iostream.h>
#include <string.h>
class A
{
private:
    int      x;
    friend class B;
public:
    A(): x(0)
    {}

    void showA()
    {
            cout<<x<<endl;
    }
};
class B
```

```
{
private:
        int     y;
public:
        B(): y(0)
        {}

        void showB(A &a)
        {
                a.x=4;
                cout<<a.x<<endl;
                cout<<y<<endl;
        }
};

void main()
{
    A       obja;
    B       objb;
    objb.showB(obja);
    cout<<"after calling showB()..."<<endl;
    obja.showA();
}
```

Q. What would happen if we write the statement friend class B; in public or protected portions? Would it have any effect on behavior of the class?

8.3 In-Lab

8.3.1 Activity

Create a class **RationalNumber** that stores a fraction in its original form (i.e. without finding the equivalent floating pointing result). This class models a fraction by using two data members: an integer for numerator and an integer for denominator. For this class, provide the following functions:

a) A **no-argument constructor** that initializes the numerator and denominator of a fraction to some fixed values.

b) A **two-argument constructor** that initializes the numerator and denominator to the values sent from calling function. This constructor should prevent a 0 denominator in a fraction, reduce or simplify fractions that are not in reduced form, and avoid negative denominators.

c) A **display** function to display a fraction in the format a/b.

d) Provide the following operator functions as **non-member <u>friend</u> functions.**

i. An overloaded **operator +** for addition of two rational numbers.

Two fractions a/b and c/d are added together as:

$$\frac{a}{b}+\frac{c}{d}=\frac{(a*d)+(b*c)}{b*d}$$

ii. An overloaded **operator -** for subtraction of two rational numbers.

Two fractions a/b and c/d are subtracted from each other as:

$$\frac{a}{b}-\frac{c}{d}=\frac{(a*d)-(b*c)}{b*d}$$

iii. An overloaded **operator *** for subtraction of two rational numbers.

Two fractions a/b and c/d are multiplied together as:

$$\frac{a}{b}*\frac{c}{d}=\frac{a*c}{b*d}$$

iv. An overloaded **operator /** for division of two rational numbers.

If fraction a/b is divided by the fraction c/d, the result is

$$\frac{a}{b}\Big/\frac{c}{d}=\frac{a*d}{b*c}$$

v. Overloaded relational operators

a. **operator >**: should return a variable of type **bool** to indicate whether 1st fraction is greater than 2nd or not.

b. **operator <**: should return a variable of type **bool** to indicate whether 1st fraction is smaller than 2nd or not.

c. **operator >=**: should return a variable of type **bool** to indicate whether 1st fraction is greater than or equal to 2nd or not.

d. **operator <=**: should return a variable of type **bool** to indicate whether 1st fraction is smaller than or equal to 2nd or not.

vi. Overloaded equality operators for **RationalNumber** class

a. **operator==**: should return a variable of type **bool** to indicate whether 1st

fraction is equal to the 2^{nd} fraction or not.

b. **Operator!=:** should a **true** value if both the fractions are not equal and return a **false** if both are equal.

8.3.2 Activity

Create a class called **Time** that has separate int member data for hours, minutes, and seconds. Provide the following member functions for this class:

a) A **no-argument constructor** to initialize hour, minutes, and seconds to 0.

b) A **3-argument constructor** to initialize the members to values sent from the calling function at the time of creation of an object. Make sure that valid values are provided for all the data members. In case of an invalid value, set the variable to 0.

c) A member function **show** to display time in 11:59:59 format.

d) Provide the following functions as **friends**

   a. An overloaded **operator+** for addition of two Time objects. Each time unit of one object must add into the corresponding time unit of the other object. Keep in view the fact that minutes and seconds of resultant should not exceed the maximum limit (60). If any of them do exceed, subtract 60 from the corresponding unit and add a 1 to the next higher unit.

   b. Overloaded operators for **pre- and post- increment**. These increment operators should add a 1 to the **seconds** unit of time. Keep track that **seconds** should not exceed 60.

   c. Overload operators for **pre- and post- decrement**. These decrement operators should subtract a 1 from **seconds** unit of time. If number of seconds goes below 0, take appropriate actions to make this value valid.

A **main()** programs should create two initialized **Time** objects and one that isn't initialized. Then it should add the two initialized values together, leaving the result in the third **Time** variable. Finally it should display the value of this third variable. Check the functionalities of ++ and -- operators of this program for both pre- and post-incrementation.

8.3 Home-Lab

8.3.3 Activity

Let us model digital storage. Digital data is stored in the form of bits. 8 bits constitute one byte. Create a class **Storage** that specifies the size of a file. This class has two integer data members: bits and bytes.

a) Provide a **no-argument constructor** to initialize the size of some file to 0 bits and 0 bytes.

b) Provide a **two-argument constructor** to initialize the size of a file to values specified at the time of creation of an object.

c) Provide the following functions as friends:

   a. Provide an overloaded **operator +** that is used to indicate the size of the resultant file obtained as a result of merging two files.

   b. Provide an overloaded **operator +=** that is used to indicate the size of a file if another file is concatenated at the end of it.

   c. Provide overloaded **post-increment and pre-increment** operators to increment the size of a file by one bit. (You must write the functions to accommodate statements like f2=++f1; and f2=f1++; where f1 and f2 are instances of the class Storage)

   d. Provide an overloaded **operator >** to determine whether one file is larger in size than the other or not. This function should return a **bool** type variable.

Write a driver program to test the functionality of this class.

8.3.4 Activity

Create a class called **IntegerSet**. Each object of class **IntegerSet** can hold integers in the range 0 through 49. A set is represented internally as an array of ones and zeros. Array element a[i] is 1 if integer i is in the set. Array element a[j] is 0 if integer j is not in the set. The default constructor initializes a set to the so-called “empty set,” i.e., a set whose array representation contains all zeros.

Provide **friend** functions for the common set operations. For example,

1. Provide an **operator +** non-member function that creates a third set which is the set-theoretic union of two existing sets (i.e., an element of the third set’s array is set to 1 if that element is 1 in either or both of the existing sets, and an element of the third set’s array is set to 0 if that element is 0 in each of the existing sets).

2. Provide an **operator*** function that creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set’s array is set to 0 if that element is 0 in either or both of the existing sets, and an element of the third set’s array is set to 1 if that element is 1 in each of the existing sets).

3. Provide an **insertElement** member function that inserts a new integer k into a set (by setting a[k] to 1).

4. Provide a **deleteElement** member function that deletes integer m (by setting a[m] to 0).

5. Provide a **setPrint** member function that prints a set as a list of numbers separated by spaces. Print only those elements that are present in the set (i.e., their position in the array has a value

of 1).

6. Provide an **operator==** non-member function that determines if two sets are equal.

7. Provide an **operator~** non-member function that determines the complement of a set. Complement is computed by placing a 1 at the index where there was a 0 before and vice-versa in the complement set.

Now write a driver program to test your **IntegerSet** class. Instantiate several **IntegerSet** objects. Test that all your member functions work properly.

8.4 References

21 Class notes

22 **Object-Oriented Programming in C++ by *Robert Lafore***

23 **How to Program C++ by *Deitel & Deitel***