Friend Operator Overloading

Lab Sections

Friend Operator Overloading

1. Objectives

After you complete this experiment you will be able to overload an operator as a friend function of a class.

2. Introduction

When you overload an operator you provide your personal implementation for the operator. It comes in handy when you create a new object and you need to refine the implementation of an existing operator to match your needs. Overloading operators is really effective when the original meaning of the operator can be ported to the new object. For example, the “+” operator should mean some type of addition will take place. For strings and chars addition is concatenation; for doubles and ints it is numeric summation and for a new type of array class it may be the summation of all the corresponding elements in two adjacent arrays. Overloading C++ operators is an optional topic. However, we think it is important that you understand how to implement operator overloading, so we will cover it.

3. Definitions & Important Terms

We will define several terms you need to know to understand classes. They are as follows:

a. The **arity** of an operator is the number of parameters (operands) it requires. The arity of an operator cannot change.
b. A **friend** is a **non-member** function that has access to the state (private area) of a class. The class must give the function permission to be its friend.
c. **Chaining** occurs when a C++ statement contains several instances of the same overloaded operator.

4. Declaration Syntax

```
friend return_type function_name(formal parameter list);
//place this form of a prototype in your class declaration
```

Notice the location of the friend statement in the class declaration and the syntax for the implementation of the friend function body in the following code:

```
class Class_name
{
public:
      constructors
      destructor
      member functions
            accessors
            mutators
      friend return_type function_name(formal parameter list);_
      public data
private:
      helper functions
      data
};

--------------------------------

return_type function_name(formal parameter list)
{
      body
}
```

More information on classes can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the implementation for overloading the operator "<<" as a friend function with chaining. Enter, save, compile and execute the following program in MSVS. Call the new project "FriendOpOverloadingExp1" and the program "FriendOpOverloading1.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

const int SIZE = 10;

class Bank_Acct
{
public:
      Bank_Acct( );  //default constructor
      Bank_Acct(double new_balance, string Cname); //explicit value
                                                   //constructor

        //overloading operator<< as a friend function with chaining
      friend ostream & operator<<(ostream & output, Bank_Acct & Org);
private:
      double balance;
      string name;
};

Bank_Acct::Bank_Acct()
{
      balance = 0;
      name = "NoName";
}

Bank_Acct::Bank_Acct(double amount, string Cname)
{
      balance = amount;
      name = Cname;

}
```

```
ostream & operator<<(ostream & output, Bank_Acct & Org)
{
        output<<endl<<"Object "<<Org.name;
        output<<endl<<"The new balance is "<<Org.balance<<endl;
        return output;
}

int main()
{
        Bank_Acct my_Acct;

        Bank_Acct DrB(2000.87, "Dr. Bullard");

        //the following statement contains chaining
        cout<<DrB<<endl<<my_Acct<<endl;

        return 0;
}
```

Question 1: Please explain how chaining works? (hint: Look at the cout statement in main)

Question 2: What is the arity of the "<<" operator?

Question 3: What variables are initialized by the explicit-value constructor? What values are the variables initialized to?

Question 4: What is the return type of the operator<< function?

Question 5: Why isn't there a "Bank_Acct::" prefixed to the header of the operator<< function?

Step 2: In this experiment you will investigate the implementation for overloading the operator "<<" as a friend function without chaining. Enter, save, compile and execute the following program in MSVS. Call the new project "FriendOpOverloadingExp2" and the program "FriendOpOverloading2.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

const int SIZE = 10;

class Bank_Acct
{
public:
	Bank_Acct( );  //default constructor
	Bank_Acct(double new_balance, string Cname); //explicit value
	//constructor overloading operator<< as a friend function with chaining

	friend void operator<<(ostream & output, Bank_Acct & Org);
private:
	double balance;
	string name;
};

Bank_Acct::Bank_Acct()
{
	balance = 0;
	name = "NoName";
}

Bank_Acct::Bank_Acct(double amount, string Cname)
{
	balance = amount;
	name = Cname;

}

void operator<<(ostream & output, Bank_Acct & Org)
{
	output<<endl<<"Object "<<Org.name;
	output<<endl<<"The new balance is "<<Org.balance<<endl;

}

int main()
{
	Bank_Acct my_Acct;

	Bank_Acct DrB(2000.87, "Dr. Bullard");

	//the following statement does not contain chaining
	cout<<DrB;

	return 0;
}
```

Question 6: Did the program in Step 2 execute without any errors? Make a statement about chaining and this program.

Question 7: Replace the cout statement in the main function of the program in Step 2 with the following statement:

```
cout<<DrB<<endl<<my_Acct<<endl;
```

Are there any errors when you try to compile the program? Explain your observations.

Question 8: Referring to the program in Step 1, please re-write new cout statements representing the one cout statement in the main function that was used in chaining.

Question 9: Please remove the cout statement in the program in Step 2 and replace it with the cout statements you wrote for Queston 8. Execute the new program and explain (if any) errors that may have occured.