Member Operator Overloading

Lab Sections

Member Operator Overloading

1. **Objectives**

   After you complete this experiment you will be able to overload an operator as a member 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 **non-member** function of a class does not have access to the state (private area) of a class.
   c. **Chaining** occurs when a C++ statement contains several instances of the same overloaded operator.
   d. The **current object** is the object that performed the function invocation.
   e. **"this"** is a pointer to the current object.
   f. **All member functions of a class have access to the "this" pointer.**
   g. **Non-member and friend functions of a class do not have access to the "this" pointer.**
   h. **"*this"** is the current object.
   i. When an object is passed **"implicitly"** it is passed through the "**this" pointer**.
   j. When an object is passed **"explicitly"** it is passed through its corresponding formal parameter.

4. Declaration Syntax

Notice that the non-member function does not have a prototype in the class declaration and the syntax for the implementation of the non-member function body in the following code:

```
class Class_name
{
public:
      constructors
      destructor
      member functions
            accessors
            mutators
      public data
private:
      helper functions
      data
};
--------------

return_type Class_name::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 member function of a class with chaining. Enter, save, compile and execute the following program in MSVS. Call the new project "MemberOpOverloadingExp" and the program "MemberOpOverloading.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
      void Print( ); //accessor function
      Bank_Acct & operator+(double amount); //mutator function

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 Bank_Acct::Print()
{
      cout<<endl<<"Object "<<name;
      cout<<endl<<"The new balance is "<<balance<<endl;
}

Bank_Acct & Bank_Acct::operator+(double amount)
{
      balance += amount;
      return *this;
}

int main()
{
      Bank_Acct my_Acct;

      cout.setf(ios::showpoint);
      cout.setf(ios::fixed);
      cout.precision(2);

      cout<<"Original balance of my_Acct"<<endl;
      my_Acct.Print( );

      //the following statement contains chaining
      my_Acct + 18.75 + 14.35 + 10054.96;

      cout<<"The balance of my_Acct after addition to balance 3 times"<<endl;
      my_Acct.Print();

      return 0;
}
```

Question 1: Referring to the program in Step 1, if the arity of the "+" operator is two, why is there only one formal parameter in operator+'s function header?

Question 2: What is the return type of the operator+ function?

Question 3: Referring to the operator+ function, what is the name of the Bank_Acct object it returned?

Question 4: Why is there a “Bank_Acct::” prefixed to the header of the operator+ function?

Question 5: Can you explain how chaining is performed for the operator+ function in the program in Step 1.

Question 6: Please explain why the operator “<<” cannot be overloaded as a member function? (hint: modify the code that was used in the laboratory “Classes: Part 2” to overload the “<<” operator as a friend member function.)