**Lab Sections**

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

|  |
| --- |
|  |

Non-Member Operator Overloading

**Non-member Operator Overloading**

1. **Objectives**

**After you complete this experiment you will be able to overload an operator as a non-member function of a class.**

1. **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.

1. **Definitions & Important Terms**

We will define several terms you need to understand in order to implement operator overloading as a non-member function. They are as follows:

1. The **arity** of an operator is the number of parameters (operands) it requires. The arity of an operator cannot change.
2. A **non-member** function of a class does not have access to the state (private area) of a class.
3. **Chaining** occurs when a C++ statement contains several instances of the same overloaded operator.
4. **Declaration Syntax**

**Notice the location of the non-member function of the class 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 function\_name(formal parameter list)

{

body

}

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

1. **Experiments**

**Step 1: In this experiment you will investigate the implementation for overloading the operator “<<” as a non-member function of a class with chaining. Enter, save, compile and execute the following program in MSVS. Call the new project “NonMemberOpOverloadingExp” and the program “NonMemberOpOverloading.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(ostream & out); //accessor 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(ostream & output)

{

output<<endl<<"Object "<<name;

output<<endl<<"The new balance is "<<balance<<endl;

}

ostream & operator<<(ostream & output, Bank\_Acct & Org)

{

Org.Print(output);

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;

}

1. Please explain how chaining works? (hint: look at the cout statement in main)
2. What is the arity of the “<<” operator in the program?
3. What is the return type of the operator<< function in the program?
4. What is the purpose of the Print function in the program?
5. Why isn’t there a “Bank\_Acct::” prefixed to the header of the operator<< function?
6. What are the similarities and differences you observed in overloading the operator<< as a friend function and as a non-member function of a class? **[Answer after completing the Friend Operator Overloading lab.]**