

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

CL 103 - COMPUTER PROGRAMMING LAB

Instructors: Mr. Basit Ali, Ms. Mahrukh Khan, Ms. Ammara Yaseen, Ms. Tooba Ali, Ms. Maham Mobin, Mr. Muhammad Irfan Ayub

Email: basit.jasani@nu.edu.pk , mahrukh.khan@nu.edu.pk, ammara.yaseen@nu.edu.pk, tooba.ali@nu.edu.pk, maham.mobin@nu.edu.pk, muhammad.irfan@nu.edu.pk

Lab # 08

Outline

- Polymorphism
 - Operator Overloading
 - Operator overloading as member functions
 - Operator overloading as non-functions
 - Examples
 - Exercise
-

POLYMORPHISM

Polymorphism refers to the ability of a method to be used in different ways, that is, it can take different forms at different times (poly + morphos).

TYPES OF POLYMORPHISM

There are two types of polymorphism:

- Compile time polymorphism
- Run time polymorphism.

COMPILE TIME POLYMORPHISM

Compile time (static) polymorphism occurs when a method is overloaded.

EXAMPLE: An example of this would be suggesting different names for being the President of a country, which would get you different results each time – but they would still be called the President.

TYPES OF OVERLOADING:

- Constructor Overloading – Already discussed in previous labs.
- Function Overloading - Already discussed in previous labs.
- Operator Overloading - Is discussed below.

OPERATOR OVERLOADING

An operator is said to be overloaded if it is defined for multiple types. In other words, overloading an operator means making the operator significant for a new type.

BUILT IN OVERLOADS

Most operators are already overloaded for fundamental types.

Example:

- 1) In the case of the expression:

a / b

the operand type determines the machine code created by the compiler for the division operator. If both operands are integral types, an integral division is performed; in all other cases floating-point division occurs. Thus, different actions are performed depending on the operand types involved.

- 2) $<<$, which is used both as the stream insertion operator and as the bitwise left-shift operator.

OVERLOADS FOR USER DEFINED TYPES

Operators can be used with user-defined types as well. Although C++ does not allow new operators to be created, it does allow most existing operators to be overloaded so that, when they're used with objects, they have meaning appropriate to those objects.

Example:

The effect of $+$ operator can be stipulated for the objects of a particular class.

OPERATOR FUNCTION SYNTAX

To overload an operator, an appropriate *operator function is required*.

```
returntype operator op (arg_list)
{
    function body // task defined
}
```

- **returntype** is the type of value returned by the specified operation.
- **op** is the operator being overloaded. (+,-etc)
- **op** is preceded by the keyword **operator**.

LIST OF OPERATORS THAT CAN BE OVERLOADED IN C++

new	delete	new []	delete []					
+	-	*	/	%	^	&		~
!	=	<	>	+=	-=	*=	/=	%=
^=	&=	=	<<	>>	>>=	<<=	==	!=
<=	>=	&&		++	--	,	->*	->
()	[]							

LIST OF OPERATORS THAT CAN'T BE OVERLOADED

- **?:(conditional)**
- **.(memberselection)**
- **.*(member selection with pointer-to-member)**
- **::(scoperesolution)**
- **sizeof (object size information)**
- **typeid (object type information)**

OPERATOR OVERLOADING AS MEMBER FUNCTIONS

If the operator function of a binary operator is defined as a method inside the class, the left operand must always be an object of the class. The operator function is called for this object. The second, right operand is passed as an argument to the method. The method thus has a single parameter.

EXAMPLE

```
//Rupee.h
#include<sstream>      // The class
stringstream#include<iomanip>
#include <iostream>
using namespace std;

class Rupee
{
    private:
        long data;
    public:
        Rupee( int rupee = 0)
        {
            data = rupee;
        }
}
```

```

    Rupee operator-()const           // Negation (unary minus)
    {
        Rupee temp;
        temp.data = -data;
        return temp;
    }

    Rupee operator+( const Rupee&obj)const    // addition.
    {
        Rupee temp;
        temp.data = data + obj.data;
        return temp;
    }

    Rupee operator-( const Rupee&obj)const    // Subtraction.
    {
        Rupee temp;
        temp.data = data - obj.data;
        return temp;
    }

    Rupee& operator+=( const Rupee&obj)       // Add Rupees.
    {
        data += obj.data;
        return *this;
    }

    Rupee& operator-=( const Rupee&obj)       // Subtract Rupees.
    {
        data -= obj.data;
        return *this;
    }

    friend ostream& operator<<( ostream&os, const Rupee &e );

};

ostream& operator<<(ostream&os, const Rupee& e) //Overloading << operator
{
    os<<e.data;
    return os;
}

```

```

//TestRupee.cpp
#include "Rupee.h"
#include <iostream>
using namespace std;

int main()
{
    Rupee wholesale(20), retail;
    retail=wholesale;           // Standard assignment

    cout<< "Wholesale price:"<<wholesale;
}

```

```

cout<< "\nRetail price: "<<retail;

Rupree discount(2);
retail -= discount;
cout<< "\nRetail price including discount: "<<retail;

wholesale = 34.10;
cout<< "\nNew wholes:

retail = wholesale + 10;
cout<< "\nNew retail p rice: "<<retail;

Rupree profit( retail - w holesale);
cout<< "\nThe profit: " <<profit;

profit = -profit;
cout<< "\nThe profit after unary minus: "<<profit;

return 0;
}

```

This Statement Means:

retail = wholesale.operator+(Rupree(10)); The binary operator is always called with reference to the left hand argument and here, it must be a class object because the operator function has been defined as a class method. So, the operator function doesn't handle the following situation:

retail = 10 + wholesale;

However, if we want to convert both operands, we will need global definitions for the operator functions.

```

Wholesale price: 20
Retail price: 20
Retail price including discount: 18
New wholesale price: 34
New retail price: 44
The profit: 10
The profit after unary minus: -10
-----
Process exited after 0.03745 seconds with return value 0
Press any key to continue . . .

```

OPERATOR OVERLOADING AS NON-FUNCTIONS

GENERAL SYNTAX:

```

TYPE1 operator OP(TYPE2lhs, TYPE3rhs)
{
}

```

EXAMPLE:

```

//Rupree.h
//Globally Defined
Rupree operator+( const Rupree& e1, const Rupree& e2) // addition.
{
    Rupree temp(e1);
    temp += e2;
    return temp;
}

```

ISSUE:

A global function cannot access the private members of the class i.e. data. The function operator+() shown above therefore uses the += operator, whose operator function is defined as a public method. A global operator function can be declared as a “**friend**” of the class to allow it access to the private members of that class.

EXAMPLE:

```
//Inside class Rupee
friend Rupee operator+( const Rupee& e1, const Rupee& e2);

//Globally Defined
Rupee operator+( const Rupee& e1, const Rupee& e2) // addition.
{
    Rupee temp; temp.data = e1.data + e2.data;
    return temp;
}
```

LAB 08 EXERCISES

QUESTION#1

You are required to develop a location based System as per the following requirements:

- 1) Create a class (Location), that takes two inputs latitude (integer) and longitude (integer). The values for both inputs are set through the constructor.
- 2) The "Location" class is capable of displaying both the values without being able to alter them through Display().
- 3) Create another class (Details) which extends the functionality of Location class by displaying another location related attribute i.e. address through the function Display().The function being unable to alter the values. Set the value of address through constructor.
- 4) The "int main()" function should be coded as follows:
 - a) Create one initialized instance (details) of "Details" class and three initialized instances (obj1, obj2 and obj3) of Location class having the following values (10,20),(5,30) and (90,90) respectively.
 - b) Display the contents of each instance.
 - c) Perform the pre increment operation on obj1 and display the result.
 - d) Perform the post increment operation on obj1 , assign it to obj2 and display the result forobj2.
 - e) Add "10" to obj1 (keeping "10" the operand on right hand side), assign the result to obj2 and display the result forobj2.
 - f) Add "10" to obj1 (keeping "10" the operand on left hand side), assign the result to obj2 and display the result forobj2.
 - g) Assign the contents of obj3 to obj1 and obj2 in a single statement and display the results for all three instances.
 - h) Reference the instance of "Details" class by a pointer to the "Location" class. Use this pointer to display the address.
 - i) Return0.

QUESTION#2

Define a class named PrimeNumber that stores a prime number. The default constructor should set the prime number to 1. Add another constructor that allows the caller to set the prime number. Also, add a function to get the prime number. Finally, overload the prefix ++ and -- operators so they return a PrimeNumber object that is the next largest prime number (for ++) and the next smallest prime number (for --). For example, if the object's prime number is set to 13, then invoking ++ should return a PrimeNumber object whose prime number is set to 17. Create an appropriate test program for the class.

QUESTION#3

Write a class Time which represents time. the class should have three fields for hours, minutes and seconds. It should have constructor to initialize the hours, minutes and seconds. A function print Time() to print the current time. Overload the following operators:

- Plus operator (+) to add two time objects based on 24-hour clock.
- Operator< to compare two time objects.

QUESTION#4

Complete the following tasks:

- a. Design a PhoneCall class that holds a phone number to which a call is placed, the length of the call in minutes, and the rate charged pe rminute.
- b. Overload the == operator to compare two PhoneCalls. Consider one PhoneCall to be equal to another if both calls are placed to the same number.
- c. Create a main() function that allows you to enter 10 PhoneCalls into an array. If a PhoneCall has already

been placed to a number, do not allow a second PhoneCall to the same number. Save the file as PhoneCall.cpp.

QUESTION#5

Design a class called NumDays. The class's purpose is to store a value that represents a number of work hours and convert it to a number of days. For example, 8 hours would be converted to 1 day, 12 hours would be converted to 1.5 days, and 18 hours would be converted to 2.25 days. The class should have a constructor that accepts a number of hours, as well as member functions for storing and retrieving the hours and days. The class should also have the following overloaded operators:

- The addition operator +. The number of hours in the sum of two objects is the sum of the number of hours in the individual objects.
- The subtraction operator -. The number of hours in the difference of two objects X and Y is the number of hours in X minus the number of hours in Y.
- Prefix and postfix Increment operators ++. The number of hours in an object is incremented by 1.

Prefix and postfix decrement operators --. The number of hours in an object is decremented by 1.

QUESTION#6

Complete the following tasks:

- a. Design a Meal class with two fields—one that holds the name of the entrée, the other that holds a calorie count integer. Include a constructor that sets a Meal's fields with parameters, or uses default values when no parameters are provided.
- b. Include an overloaded insertion operator function that displays a Meal's values.
- c. Include an overloaded extraction operator that prompts a user for an entrée name and calorie count for a meal.
- d. Include an overloaded operator+()function that allows you to add two or more Meal objects. Adding two Meal objects means adding their calorie values and creating a summary Meal object in which you store "Daily Total" in the entrée field.
- e. Write a main()function that declares four Meal objects named breakfast, lunch, dinner, and total. Provide values for the breakfast, lunch, and dinner objects. Include the statement total = breakfast + lunch + dinner; in your program, then display values for the four Meal objects. Save the file asMeal.cpp.
- f. Write a main()function that declares an array of 21 Meal objects. Allow a user to enter values for 21 Meals for the week. Total these meals and display the calorie total for the end of the week. (Hint: You might find it useful to create a constructor for the Meal class.) Save the file asMeal2.cpp.