CC-211L

Object Oriented Programming

Laboratory 07

Operator Overloading

Version: 1.0.0

Release Date: 14-03-2024

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

Learning Objectives:

- Overloading Unary Operators
- Overloading ++ Operator
- Overloading – Operator
- Dynamic Memory Management
- Operators as Members vs Non-Members
- Conversion between Types
- Overloading the Function call Operator

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Lab Instructor	Mam Sanam	
Teacher Assistants	Syed M.Jafar Ali Naqvi	bitf21m032@pucit.edu.pk
	M. Usman Munir	bitf19m020@pucit.edu.pk
	Aqdas Shabir	bitf21m022@pucit.edu.pk
	Ibad Nazir	bitf21m024@pucit.edu.pk
	Umer Farooq	bitf21m010@pucit.edu.pk

Background and Overview:

Operator Overloading:

In C++, we can make operators work for user-defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +.

Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.

© https://www.geeksforgeeks.org/operator-overloading-c/

Unary Operators:

Unary Operator is an operation that is performed on only one operand. Unary Operators contrast with the Binary operators, which use two operands to calculate the result. Unary operators can be used either in the prefix position or the postfix position of the operand. To perform operations using Unary Operators, we need to use one single operand. There are various types of unary operators, and all these types are of equal precedence, having right-to-left associativity.

Dynamic Memory Management:

Dynamic memory management in C++ refers to performing memory allocation manually by programmer. Dynamically allocated memory is allocated on Heap and non-static and local variables get memory allocated on Stack.

Type Conversion:

Type conversion is the process that converts the predefined data type of one variable into an appropriate data type. The main idea behind type conversion is to convert two different data type variables into a single data type to solve mathematical and logical expressions easily without any data loss.

Activities:

Pre-Lab Activities:

Overloading Unary Operators:

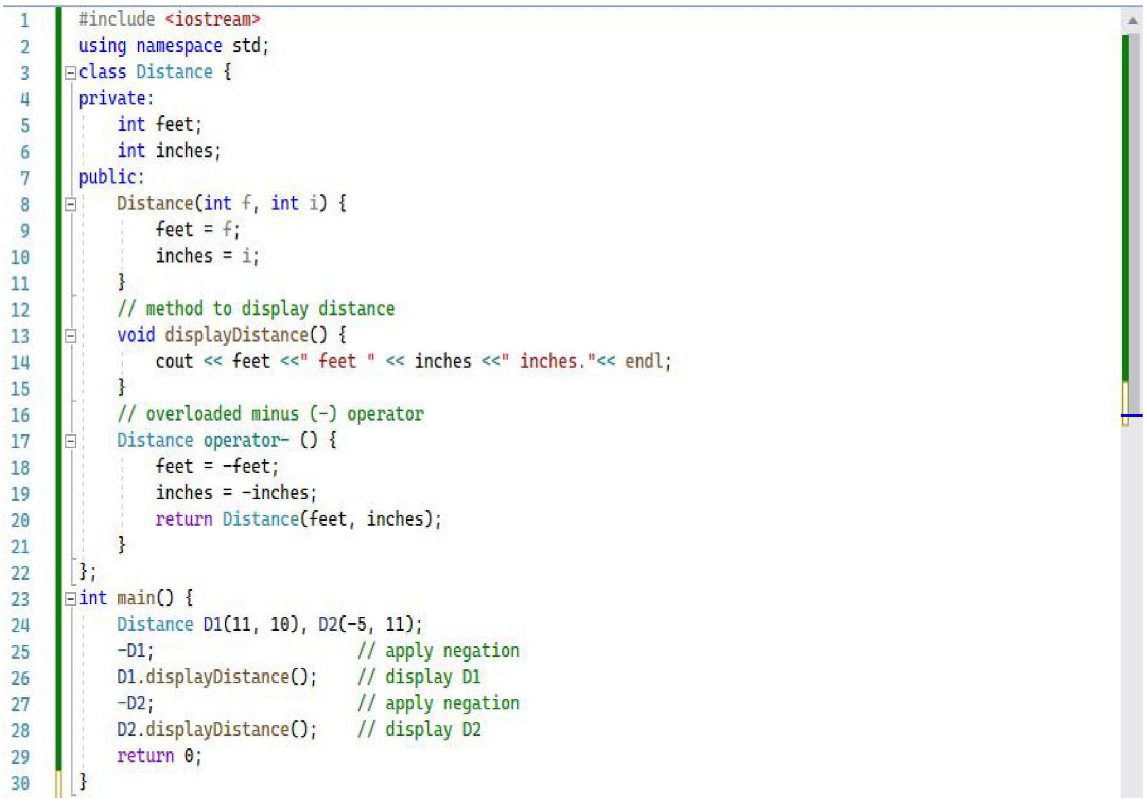
The unary operators operate on a single operand and following are the examples of Unary operators −

- The increment (++) and decrement (--) operators
- The unary minus (-) operator
- The logical not (!) operator

The unary operators operate on the object for which they were called and normally, this operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometimes they can be used as postfix as well like obj++ or obj--.

Following example explain how minus (-) operator can be overloaded for prefix as well as postfix usage.

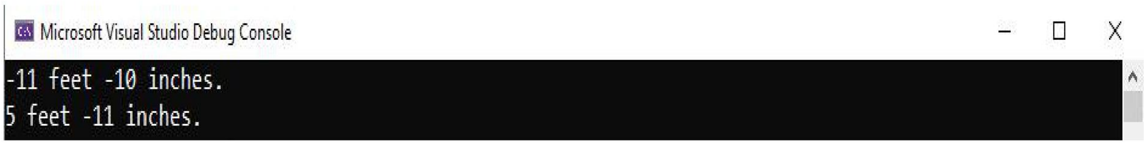
Example:



```cpp
#include <iostream>
using namespace std;
class Distance {
private:
    int feet;
    int inches;
public:
    Distance(int f, int i) {
        feet = f;
        inches = i;
    }
    // method to display distance
    void displayDistance() {
        cout << feet <<" feet " << inches <<" inches."<< endl;
    }
    // overloaded minus (-) operator
    Distance operator- () {
        feet = -feet;
        inches = -inches;
        return Distance(feet, inches);
    }
};
int main() {
    Distance D1(11, 10), D2(-5, 11);
    -D1;                    // apply negation
    D1.displayDistance();   // display D1
    -D2;                    // apply negation
    D2.displayDistance();   // display D2
    return 0;
}
```

Fig. 01 (Overloading (-) Operator)

Output:



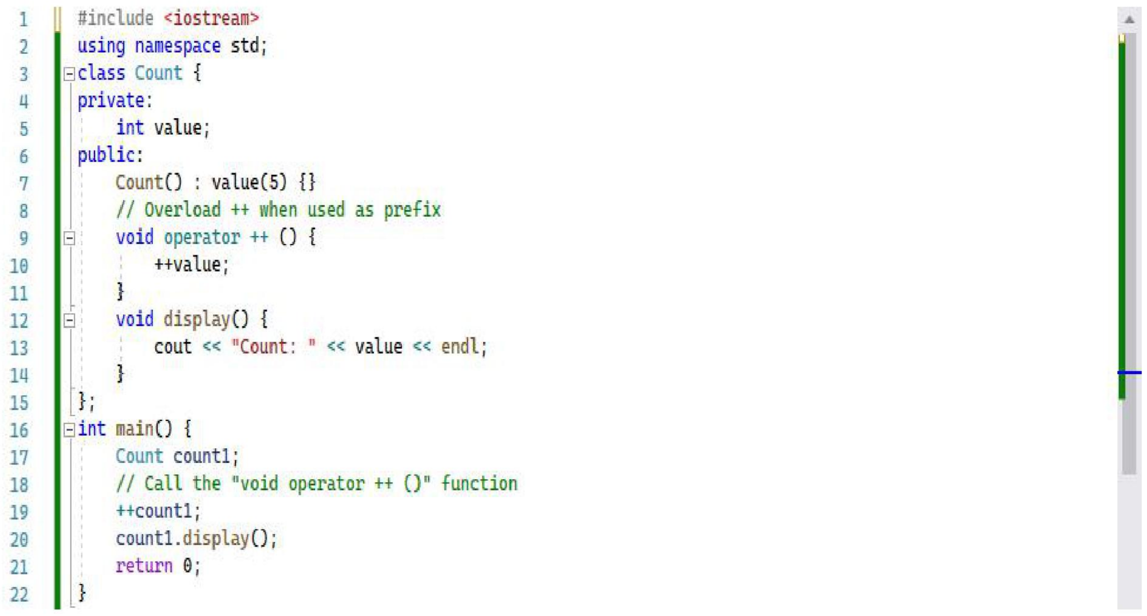
```
-11 feet -10 inches.
5 feet -11 inches.
```

Fig. 02 (Overloading (-) Operator)

Overloading Increment and Decrement Operators:

Following example explain how Increment (++) operator can be overloaded for prefix usage.

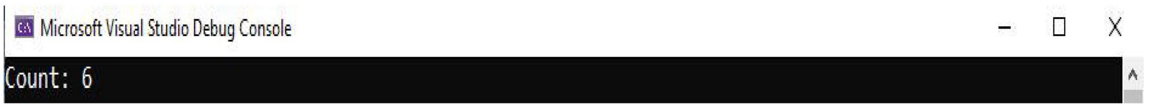
Example:



```cpp
#include <iostream>
using namespace std;
class Count {
private:
    int value;
public:
    Count() : value(5) {}
    // Overload ++ when used as prefix
    void operator ++ () {
        ++value;
    }
    void display() {
        cout << "Count: " << value << endl;
    }
};
int main() {
    Count count1;
    // Call the "void operator ++ ()" function
    ++count1;
    count1.display();
    return 0;
}
```

Fig. 03 (Overloading ++ Operator)

Output:



```
Count: 6
```

Fig. 04 (Overloading ++ Operator)

In the above example, when we use ++count1, the void operator ++ () is called. This increases the value attribute for the object count1 by 1. It works only when ++ is used as a prefix. To make ++ work as a postfix we use this syntax.

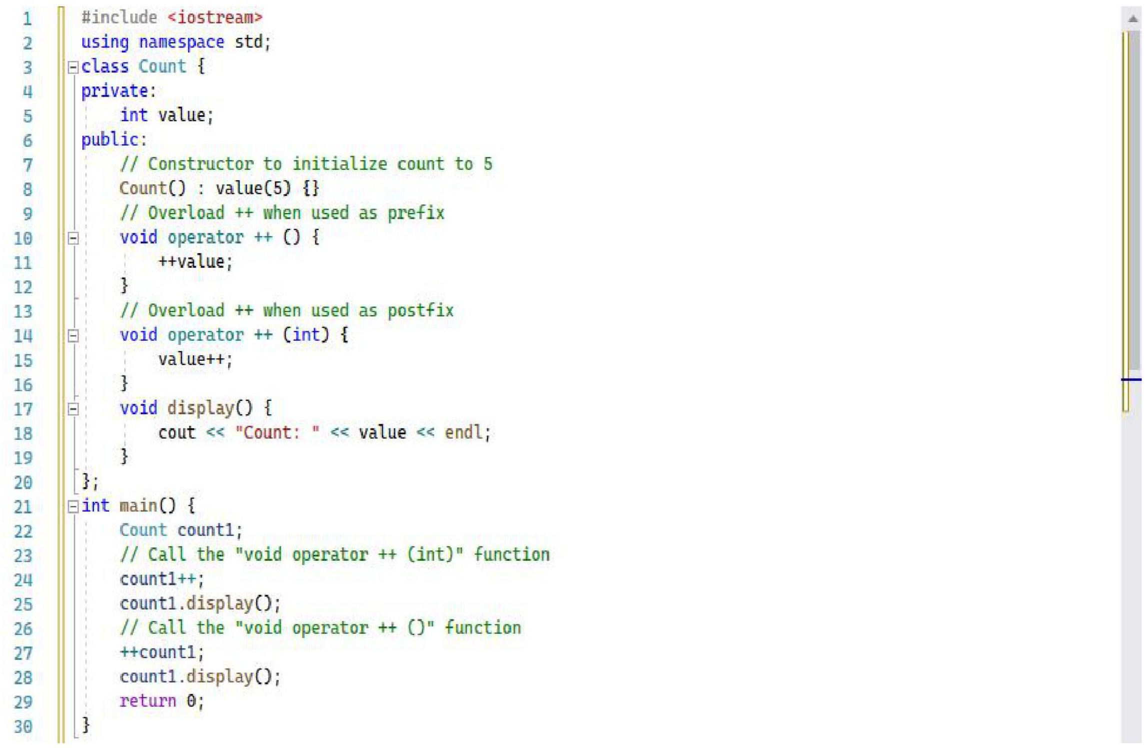
void operator ++ (int){

//code

}

Notice the int inside the parentheses. It's the syntax used for using unary operators as postfix; it's not a function parameter.

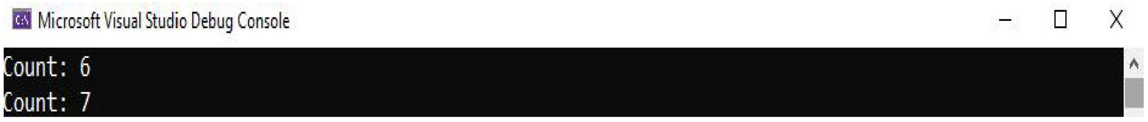
Example:



```cpp
#include <iostream>
using namespace std;
class Count {
private:
    int value;
public:
    // Constructor to initialize count to 5
    Count() : value(5) {}
    // Overload ++ when used as prefix
    void operator ++ () {
        ++value;
    }
    // Overload ++ when used as postfix
    void operator ++ (int) {
        value++;
    }
    void display() {
        cout << "Count: " << value << endl;
    }
};
int main() {
    Count count1;
    // Call the "void operator ++ (int)" function
    count1++;
    count1.display();
    // Call the "void operator ++ ()" function
    ++count1;
    count1.display();
    return 0;
}
```

Fig. 05 (Overloading ++ Operator)

Output:



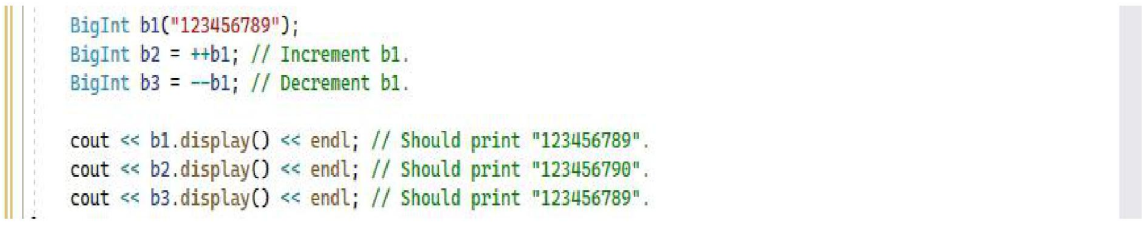
```
Count: 6
Count: 7
```

Fig. 06 (Overloading ++ Operator)

Task 01: Big Integer **[Estimated time 30 minutes / 20 marks]**

- Write a C++ class called BigInt that represents a large integer. The BigInt class should overload the unary ++ (pre-increment) and -- (pre-decrement) operators.
- The BigInt class should be able to represent integers of arbitrary size. You can do this by storing the digits of the integer as an array of integers, where each element in the array represents a single digit. For example, the integer 12345 would be represented as the array {5, 4, 3, 2, 1}.
- When the ++ operator is called, it should increment the BigInt object's integer value by 1 and return the updated BigInt object.
- When the -- operator is called, it should decrement the BigInt object's integer value by 1 and return the updated BigInt object.

Here's an example of how the BigInt class should be used:



```
BigInt b1("123456789");
BigInt b2 = ++b1; // Increment b1.
BigInt b3 = --b1; // Decrement b1.

cout << b1.display() << endl; // Should print "123456789".
cout << b2.display() << endl; // Should print "123456790".
cout << b3.display() << endl; // Should print "123456789".
```

Task 02: Index operator **[Estimated time 30 minutes / 20 marks]**

Create a class Array. It will have a D-array and a data member size in in as private data members.

Create default constructor, parameterized constructor and a display function.

Create a function Input to input values in D- array in class.

- Overload index operator [ ] for this class which will receive index number in it as parameter and will return the number stored at that index. If the index is out of bound, then throw exception runtime error and return -1.
- Overload + operator to add Two Array class objects. Two Array class objects will add in such a way that each respective index value of one array gets added to respective index of the second array. Return a new Object of Array class from this operator which will hold sum of respective indexes of first and second Array object whom with which the + operator is called.
- Similarly, create - operator as well.

Submissions:

- For In-Lab Activity:
  - Save the files on your PC.
  - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
  - Submit the .cpp file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's

Division of Pre-Lab marks:	**[20 marks]**
Task 01: Big Integer	[20 marks]
Division of In-Lab marks:	**[80 marks]**
Task 01: Students GPA	[20 marks]
Task 02: Cast Complex Numbers	[30 marks]
Task 03: Overloading Polynomials	[30 marks]
Division of Post-Lab marks:	**[30 marks]**
Task 01: Convert 2D to 1D	[30 marks]

References and Additional Material:

- **Unary Operator Overloading**
  https://www.tutorialspoint.com/cplusplus/unary_operators_overloading.htm
- **Dynamic Memory Management**
  https://www.programiz.com/cpp-programming/memory-management
- **Overloading Function Call Operator**
  https://www.ibm.com/docs/en/i/7.2?topic=only-overloading-function-calls-c

Lab Time Activity Simulation Log:

Slot – 01 – 00:00 – 00:15:	Class Settlement
Slot – 02 – 00:15 – 00:40:	In-Lab Task
Slot – 03 – 00:40 – 01:20:	In-Lab Task
Slot – 04 – 01:20 – 02:20:	In-Lab Task
Slot – 05 – 02:20 – 02:45:	Evaluation of Lab Tasks
Slot – 06 – 02:45 – 03:00:	Discussion on Post-Lab Task