**Lab # 6**

**Operator Overloading in C++**

**Objective:**

The objective of this lab is to understand the concept of **operator overloading** in C++. By the end of this lab, you should be able to:

1. Overload unary and binary operators.
2. Understand the arguments and return values of overloaded operators.
3. Apply operator overloading in real-world scenarios like arithmetic operations and string concatenation.

C++ allows you to specify more than one definition for a **function** name or an **operator** in the same scope, which is called **function overloading** and **operator overloading** respectively.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definitions (implementation).

When you call an overloaded **function** or **operator**, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called **overload resolution**.

1. **Operator Overloading in C++**

You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with **user-defined** types as well.

Overloaded operators are functions with special names, the keyword **operator** followed by the **symbol for the operator** being defined. Like any other function, an overloaded operator has a return type and a parameter list.

**Syntax**



**Example**



Declares the addition operator that can be used to add two Box objects and returns final Box object.

Most overloaded operators may be defined as **ordinary non-member functions** or as **class member functions.** In case we define above function as non-member function of a class then we would have to pass two arguments for each operand as follows:



The following is the example to show the concept of operator over loading using a member function. Here an object is passed as an argument whose properties will be accessed using **this** object, the object which will call this operator can be accessed using this operator as explained below:

**Example**

class Box {

public:

// Method to calculate volume

double getVolume() {

return length \* breadth \* height;

}

// Setter methods

void setLength(double len) { length = len; }

void setBreadth(double bre) { breadth = bre; }

void setHeight(double hei) { height = hei; }

// Overloading + operator to add two Box objects

Box operator+(const Box& b) {

Box box;

box.length = this->length + b.length;

box.breadth = this->breadth + b.breadth;

box.height = this->height + b.height;

return box;

}

private:

double length; // Length of the box

double breadth; // Breadth of the box

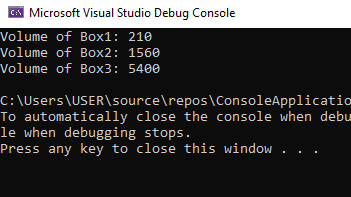
double height; // Height of the box

};

int main() {

Box Box1, Box2, Box3;

double volume = 0.0;



// Box 1 specifications

Box1.setLength(6.0);

Box1.setBreadth(7.0);

Box1.setHeight(5.0);

// Box 2 specifications

Box2.setLength(12.0);

Box2.setBreadth(13.0);

Box2.setHeight(10.0);

// Display volume of Box 1

volume = Box1.getVolume();

cout << "Volume of Box1: " << volume << endl;

// Display volume of Box 2

volume = Box2.getVolume();

cout << "Volume of Box2: " << volume << endl;

// Adding two Box objects

Box3 = Box1 + Box2;

// Display volume of Box 3

volume = Box3.getVolume();

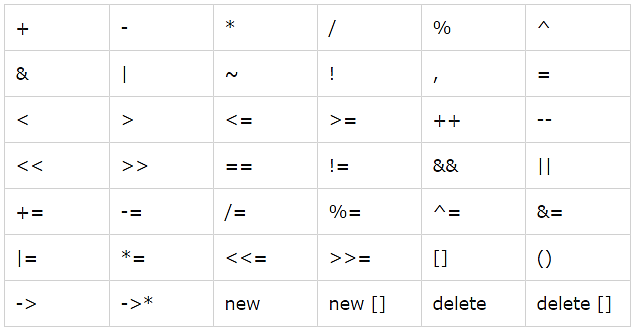
cout << "Volume of Box3: " << volume << endl;

return 0;

}

1. **Overloadable/Non-overloadable Operators**

The following is the list of operators which can be overloaded:



The Following is the list of operators, which cannot be overloaded:



1. **Unary operators overloading**

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 sometime they can be used as postfix as well like **obj++** or **obj--**.

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

**Example**

class Distance {

private:

int feet; // 0 to infinite

int inches; // 0 to 12

public:

// Default constructor

Distance() : feet(0), inches(0) {}

// Parameterized constructor

Distance(int f, int i) : feet(f), inches(i) {}

// Method to display distance

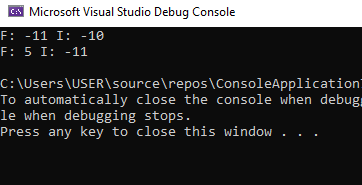
void displayDistance() const {

cout << "F: " << feet << " I: " << inches << endl;

}

// Overloaded unary minus (-) operator

Distance operator-() const {

 return Distance(-feet, -inches);

}

};

int main() {

Distance D1(11, 10), D2(-5, 11);

D1 = -D1; // Apply negation

D1.displayDistance(); // Display D1

D2 = -D2; // Apply negation

D2.displayDistance(); // Display D2

return 0;

}

1. **Binary Operators Overloading**

The binary operators take two arguments and the following are the examples of Binary operators. You use binary operators very frequently like addition (+) operator, subtraction (-) operator and division (/) operator.

The following example explains how addition (+) operator can be overloaded. Similar way, you can overload subtraction (-) and division (/) operators.

**Example**

class Box {

private:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

public:

// Default constructor

Box() : length(0), breadth(0), height(0) {}

// Parameterized constructor

Box(double len, double bre, double hei) : length(len), breadth(bre), height(hei) {}

// Function to calculate volume

double getVolume() const {

return length \* breadth \* height;

}

// Setter functions

void setLength(double len) { length = len; }

void setBreadth(double bre) { breadth = bre; }

void setHeight(double hei) { height = hei; }

// Overload + operator to add two Box objects

Box operator+(const Box& b) const {

return Box(length + b.length, breadth + b.breadth, height + b.height);

}

};

int main() {

// Declare Box1 and Box2 using parameterized constructor

Box Box1(6.0, 7.0, 5.0);

Box Box2(12.0, 13.0, 10.0);

// Calculate and display volume

cout << "Volume of Box1: " << Box1.getVolume() << endl;

cout << "Volume of Box2: " << Box2.getVolume() << endl;

// Add two boxes

Box Box3 = Box1 + Box2;

// Display volume of Box3

cout << "Volume of Box3: " << Box3.getVolume() << endl;

A screenshot of a computer

AI-generated content may be incorrect. return 0;

}

**Conclusion:**

In this lab, we learned:

1. How to overload unary and binary operators.
2. How to use operator overloading for arithmetic operations and string concatenation.
3. The importance of operator overloading in simplifying code and improving readability.

**Additional Notes:**

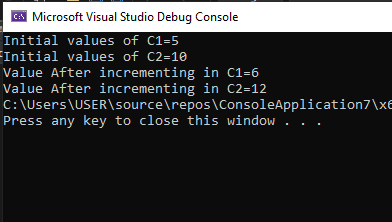
* Operator overloading should be used judiciously to maintain code clarity.
* Overloaded operators must follow the natural semantics of the operator (e.g., + should perform addition, not subtraction).

**Lab Tasks/ Homework**

**Task 01:** Write a program by using a class Counter to create two objects c1 and c2 of that class. Use constructor to initialize the value of count. Display the initial count value in the objects.

Then use the overloaded ++ operator (overloaded incrementing unary operator) to increment c1 once and c2 twice and display the resulting values.

The **output** of the program should be like:



**Solution:**

Enter your code here…

#include<iostream>

using namespace std;

class Operator {

private:

int value;

public:

Operator() {

value = 0;

}

Operator(int n) {

value = n;

}

Operator operator++(int) {

value++;

return value;

}

void initialvaule()

{

cout << "Initial Value: " << value << endl;

}

void finalvaule()

{

cout << "Initial Value: " << value << endl;

}

};

int main() {

Operator obj1(8), obj2(10);

cout << "First object: " << endl;

obj1.initialvaule();

obj1++;

obj1.finalvaule();

cout << "Second Object: " << endl;

obj2.initialvaule();

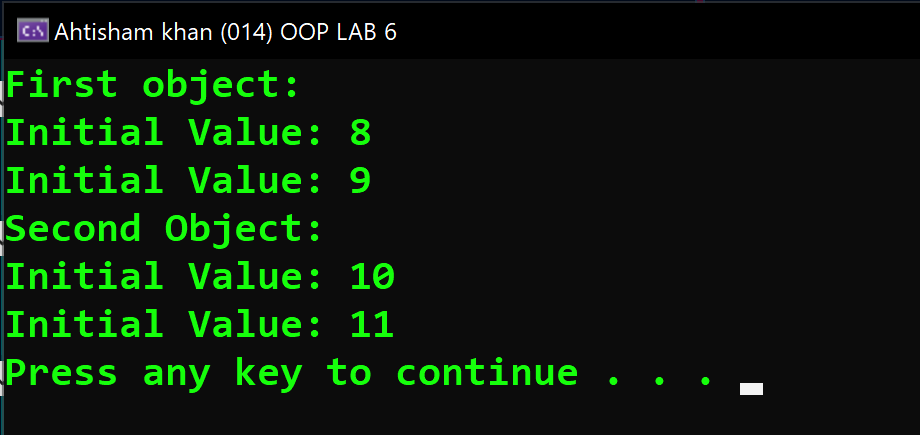
obj2++;

obj2.finalvaule();

}

**Output:**

Your Output here…



In case of output snippet please make sure output snippet contains student name and id. AliAhmed\_123\_Lab06\_T1.exe

**Task 02:** Write a program to overload the arithmetic operations (+, -, \*, /) using the concept of operator overloading.

The output of the program should be like:

A computer screen shot of a black and white screen

AI-generated content may be incorrect.

**Solution:**

Enter your code here…

#include <iostream>

using namespace std;

class Arthimatic {

private:

int num1;

int num2;

int result; // Store the operation result

public:

Arthimatic() : num1(0), num2(0), result(0) {}

Arthimatic(int a, int b) : num1(a), num2(b), result(0) {}

void getvalue1();

void getvalue2();

int add();

int sub();

int multiplication();

int division();

Arthimatic operator+(const Arthimatic& b);

Arthimatic operator-(const Arthimatic& d);

Arthimatic operator\*(const Arthimatic& e);

Arthimatic operator/(const Arthimatic& f);

int getResult() const { return result; } // Getter for result

};

void Arthimatic::getvalue1() {

cout << "Enter number 1: ";

cin >> num1;

}

void Arthimatic::getvalue2() {

cout << "Enter number 2: ";

cin >> num2;

}

int Arthimatic::add() {

result = num1;

return result;

}

int Arthimatic::sub() {

result = num1;

return result;

}

int Arthimatic::multiplication() {

result = num1;

return result;

}

int Arthimatic::division() {

/\*if (num2 == 0) {

cout << "Division by zero error!" << endl;

result = 0;

return 0;

}\*/

result = num1;

return result;

}

// Operator Overloads functions

Arthimatic Arthimatic::operator+(const Arthimatic& b) {

Arthimatic result;

result.num1 = num1 + b.num2;

result.num2 = 0; // Not needed

return result;

}

Arthimatic Arthimatic::operator-(const Arthimatic& d) {

Arthimatic result;

result.num1 = num1 - d.num2;

result.num2 = 0;

return result;

}

Arthimatic Arthimatic::operator\*(const Arthimatic& e) {

Arthimatic result;

result.num1 = num1 \* e.num2;

result.num2 = 0;

return result;

}

Arthimatic Arthimatic::operator/(const Arthimatic& f) {

Arthimatic result;

if (f.num2 == 0) {

cout << "Division by zero error in operator!" << endl;

result.num1 = 0;

return result;

}

result.num1 = num1 / f.num2;

result.num2 = 0;

return result;

}

int main() {

Arthimatic obj1, obj2, obj3;

obj1.getvalue1();

obj2.getvalue2();

int choice;

do {

cout << "\nEnter your choice:\n";

cout << "1. Addition\n";

cout << "2. Subtraction\n";

cout << "3. Multiplication\n";

cout << "4. Division\n";

cout << "5. Exit\n";

cout << "Choice: ";

cin >> choice;

switch (choice) {

case 1: {

obj3 = obj1 + obj2;

cout << "The Addition of Two numbers using Operator Overloading: " << obj3.add() << endl;

break;

}

case 2: {

obj3 = obj1 - obj2;

cout << "The Substraction of Two numbers using Operator Overloading: " << obj3.sub() << endl;

break;

}

case 3: {

obj3 = obj1 \* obj2;

cout << "The Multiplication of Two numbers using Operator Overloading: " << obj3.multiplication() << endl;

break;

}

case 4: {

obj3 = obj1 / obj2;

cout << "The Division of Two numbers using Operator Overloading: " << obj3.division() << endl;

break;

}

case 5:

cout << "Thank you!" << endl;

break;

default:

cout << "Enter a valid choice!" << endl;

}

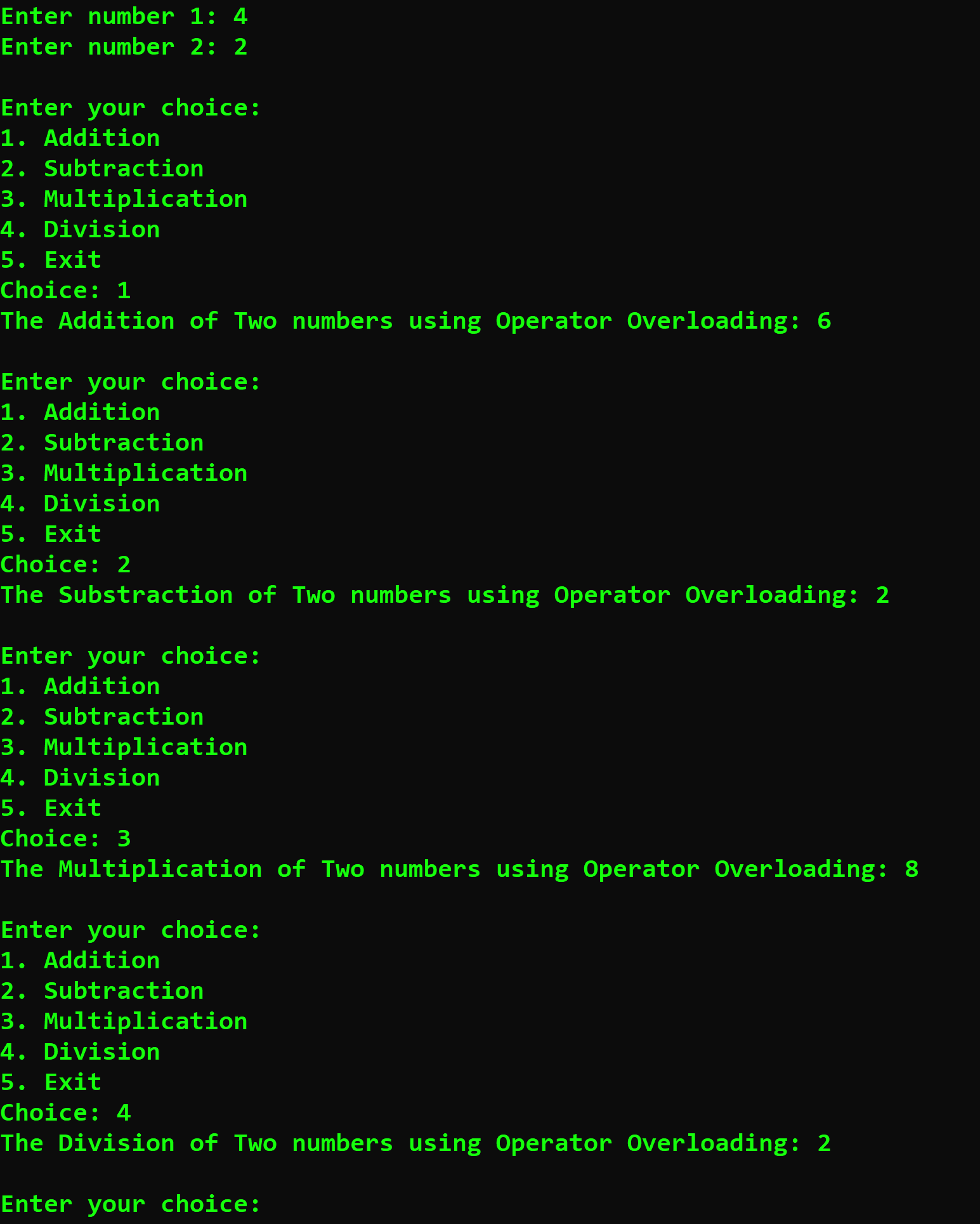
} while (choice == 1|| choice==2 ||choice==3 ||choice==4);

return 0;

}

**Output:**

Your Output here…



In case of output snippet please make sure output snippet contains student name and id. AliAhmed\_123\_Lab06\_T2.exe

**Task 03:** Write a program to concatenate (join) two strings by using overloaded + operator.

The **output** of the program should be like:

A screenshot of a computer

AI-generated content may be incorrect.

Use string copy and string concatenation built in functions.

**Hint:**

**“strcpy\_s” :** It is used to copy the contents of one string to another string variable. Its **syntax** is:**strcpy\_s (string 1, string 2);**

**“strcat\_s”:** It is used to append or combine the contents of one string to another string variable. Its **syntax** is:

**strcat\_s(string 1, string 2);**

**Solution:**

Enter your code here…

**Output:**

Your Output here…

In case of output snippet please make sure output snippet contains student name and id. AliAhmed\_123\_Lab06\_T3.exe