Object Oriented Programming Lab

SPRING - 2024

LAB 09

FAST National University of
Computer and Emerging Sciences

Learning Outcomes

In this lab you are expected to learn the following:

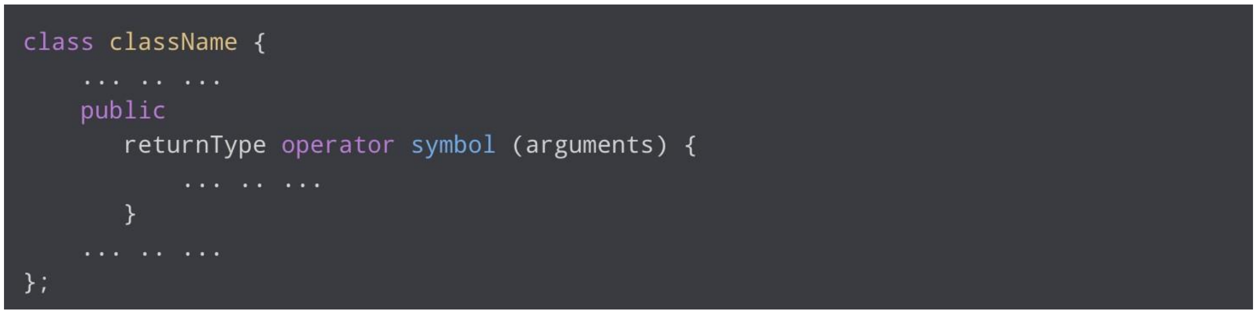
- Operator Overloading

Operator Overloading:

C++ allows you to specify more than one definition for an operator in the same scope, which is called operator overloading. Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined.

Syntax for C++ Operator Overloading:

To overload an operator, we use a special **operator** function. We define the function inside the class or structure whose objects/variables we want the overloaded operator to work with.

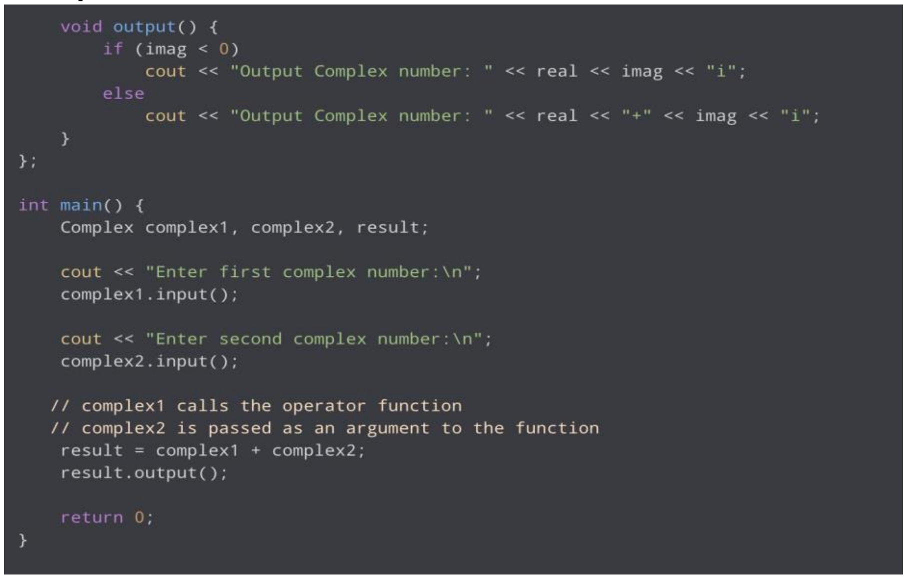


```
class className {
    ... .. ...
    public
       returnType operator symbol (arguments) {
           ... .. ...
       }
    ... .. ...
};
```

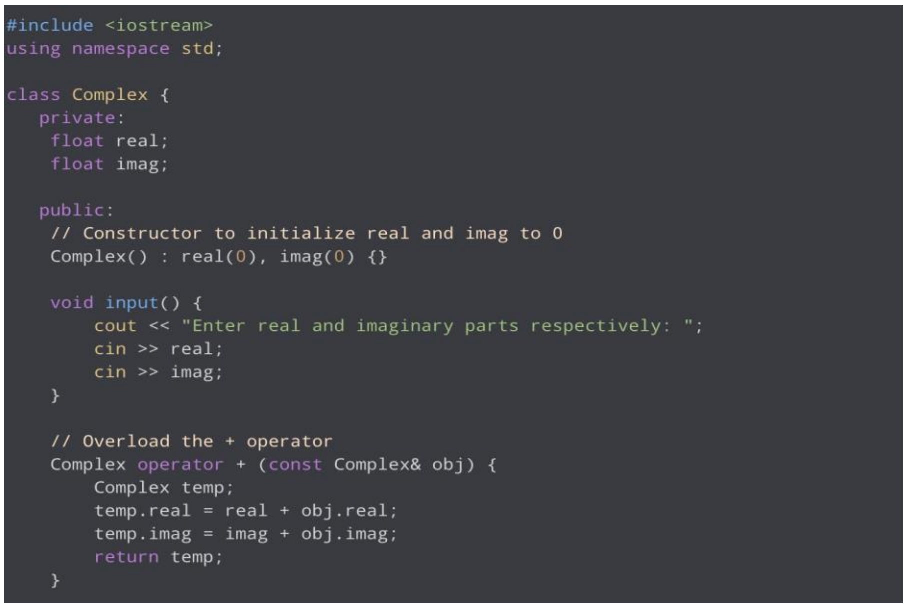
Here,

- **returnType** is the return type of the function.
- **operator** is a keyword.
- **symbol** is the operator we want to overload. Like: +, <, -, ++, etc.
- **arguments** is the arguments passed to the function.

Example:



```cpp
    void output() {
        if (imag < 0)
            cout << "Output Complex number: " << real << imag << "i";
        else
            cout << "Output Complex number: " << real << "+" << imag << "i";
    }
};

int main() {
    Complex complex1, complex2, result;

    cout << "Enter first complex number:\n";
    complex1.input();

    cout << "Enter second complex number:\n";
    complex2.input();

    // complex1 calls the operator function
    // complex2 is passed as an argument to the function
    result = complex1 + complex2;
    result.output();

    return 0;
}
```



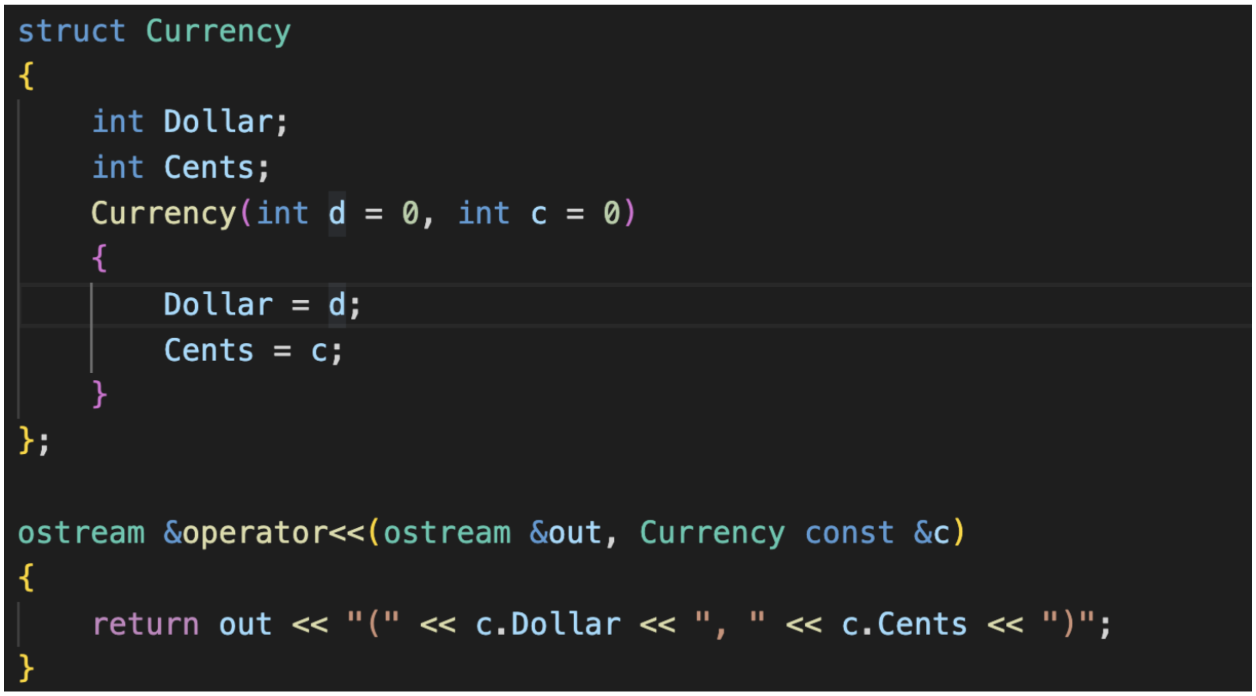
```cpp
#include <iostream>
using namespace std;

class Complex {
   private:
    float real;
    float imag;

   public:
    // Constructor to initialize real and imag to 0
    Complex() : real(0), imag(0) {}

    void input() {
        cout << "Enter real and imaginary parts respectively: ";
        cin >> real;
        cin >> imag;
    }

    // Overload the + operator
    Complex operator + (const Complex& obj) {
        Complex temp;
        temp.real = real + obj.real;
        temp.imag = imag + obj.imag;
        return temp;
    }
```

Stream Insertion and Extraction Operator Overloading

In C++, stream insertion operator “<<” is used for output and extraction operator “>>” is used for input.

We must know the following things before we start overloading these operators:

1. **cout** is an object of **ostream** class and **cin** is an object of **istream** class
2. These operators must be overloaded as **global functions**. And if we want to allow them to access private data members of the class, we must make them a **friend**.



```cpp
struct Currency
{
    int Dollar;
    int Cents;
    Currency(int d = 0, int c = 0)
    {
        Dollar = d;
        Cents = c;
    }
};

ostream &operator<<(ostream &out, Currency const &c)
{
    return out << "(" << c.Dollar << ", " << c.Cents << ")";
}
```

Lab Tasks

Submission Instructions:

1. Create a new folder with name *ROLLNO_SEC_LAB09* **e.g. i23XXXX_A_LAB09**
2. Move all of your **.cpp and .h files** to this newly created directory and compress it into a **.zip file**.
3. Now you have to submit this zipped file on Google Classroom.
4. If you don't follow the above-mentioned submission instruction, you will be marked **zero.**
5. Plagiarism in the Lab Task will result in **zero** marks in the whole category.

Q1. Write a class **Matrix**.

This class has three private data members:

- rows: An integer that holds the numbers of rows for matrix
- columns: An integer that holds the numbers of columns for matrix
- matrix: An integer pointer to pointer that points to a 2D array (rows * columns).

The class has the following member functions:

- Copy constructor: Creates a new Matrix object by copying from an existing matrix object. `Matrix(const Matrix& other)`

Matrix (int r, int c)	Constructs a new Matrix object to represent the given matrix
operator =	Overload = operator to assign values
operator ==	Overload == operator to compare whether matrices are equal or not
M2=M1+1	Overload + operator which takes integer as argument. It preforms scalar addition.
M2=M1-4	Overload - operator which takes integer as argument. It preforms scalar subtraction.
M3=M1+M2	Overload + operator which takes matrix object as argument. It adds two matrixes and returns the result.
M3=M1-M2	Overload - operator which takes matrix object as argument. It subtracts two matrixes and returns the result.

- Write the destructor to deallocate the memory

  ~Matrix()

Q2. Define a class **Fraction** with the following data members:

- numerator (int): Represents the numerator of the fraction.
- denominator (int): Represents the denominator of the fraction.

1. Write a default constructor:
- Initializes `numerator` to 0 and `denominator` to 1, representing the fraction 0/1.

2. Write a parameterized constructor:
- Accepts parameters for `numerator` and `denominator` to initialize the respective data members.

3. Write a copy constructor:
Creates a new fraction object by copying the numerator and denominator from an existing fraction object

4. Overload the following operators:

a. Equality operator (==):
  - Compares two fractions for equality based on their numerator and denominator.

b. Greater than operator (>):
  - Compares two fractions based on their magnitudes.

c. Less than operator (<):
  - Compares two fractions based on their magnitudes.

d. Input stream operator (>>):
  - Reads a fraction from the input stream, setting the numerator and denominator accordingly.

e. Output stream operator (<<):
  - Writes a fraction to the output stream in the format "numerator/denominator".

f. Addition operator (+):
  - Adds two fractions and returns the result.

g. Subtraction operator (-):

- Subtracts one fraction from another and returns the result as a simplified fraction.

h. Multiplication operator (*):

- Multiplies two fractions and returns the result as a simplified fraction.