BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

LAB FINAL

Course NO : 122

Course Name : Object Oriented Programming Language Lab

Submission Date : 14.11.23

Submitted To

Name: Khan Md. Hasib
Assistant Professor
Department of Computer Science & Engineering

Submitted By

Name : Abu Talha

ID: 22235103036 INTAKE:51 SECTION:1

Problem 1

Manchester United is a class with two private integer member variables coach and player, and a public void member function getData (). Create an object named ‘ronaldo’ in the main function. Overload the operator '++' in this class to perform the increment of both member variables through the following instruction ronaldo++ from the main function. After that create another instance of the Manchester United class named ‘fernandes’. Now, you set the values of coach and player for ronaldo to 4 and 5, and for fernandes, it is 5 and 6. Overloading only one relational operator, compare the result of ronaldo with fernandes before and after incrementing ronaldo by one. Which operator will be appropriate for both cases? Support your explanation by implementing that operator.

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

class ManchesterUnited
{
private:
    int coach = 0;
    int player = 0;

public:
    void getData()
    {
        cout << "Coach: " << coach << ", Player: " << player << endl;
    }

    ManchesterUnited(int a, int b)
    {
        coach = a;
        player = b;
    }

    void operator++(int)
    {
        // Overload the postfix increment operator
        coach++;
        player++;
    }

    bool operator==(ManchesterUnited &obj)
    {
        // Compare the current object with another object
        return (this->coach + this->player == obj.coach + obj.player);
    }
};

int main()
{
    ManchesterUnited ronaldo(4, 5);
    ManchesterUnited fernandes(5, 6);

    cout<<"Crew Member for Ronaldo  : ";
    ronaldo.getData();

    cout<<"Crew Member for Fernandes: ";
    fernandes.getData();
    if (ronaldo == fernandes)
    {
        cout << "Crew Member is equal"<<endl;
    }
    else
    {
        cout << "Crew Member is not equal"<<endl;
    }

```

```
55      cout<<endl;
56      cout<<endl;
57
58      ronaldo++; // Increment coach and player for Ronaldo
59
60      cout<<"Crew Member for Ronaldo  : ";
61      ronaldo.getData();
62
63      cout<<"Crew Member for Fernandes: ";
64      fernandes.getData();
65
66      if (ronaldo == fernandes)
67      {
68          cout << "Crew Member is equal"<<endl;
69      }
70      else
71      {
72          cout << "Crew Member is not equal"<<endl;
73      }
74
75      return 0;
76  }
77
```

```
"C:\Users\atalh\Desktop\Lab final.exe"
Crew Member for Ronaldo  : Coach: 4, Player: 5
Crew Member for Fernandes: Coach: 5, Player: 6
Crew Member is not equal


Crew Member for Ronaldo  : Coach: 5, Player: 6
Crew Member for Fernandes: Coach: 5, Player: 6
Crew Member is equal

Process returned 0 (0x0)   execution time : 0.447 s
Press any key to continue.
```

Problem 2

Demonstrate a C++ code that creates a class called Fraction. The class Fraction has two attributes: numerator and denominator.

• In your constructor (inyour__init__ method), verify(assert?) that the numerator and denominator passed in durin initiation are both of type int. If you want to be thorough, also check to make sure that the denominator is not zero

• Write a .reduce() method that will reduce a fraction to lowest terms.

• Override the Object class's __str__ and __repl__ methods so that your objects will print out nicely. Remember that__str__ is more for humans; __repl__ is more for programmers. Ideally ,the__repl__ method will produce a string that you can run through the eval() function to clone the original fraction object.

• Override the + operator. In your code, this means that you will implement the special method __add__. The signature of the __add__ function will be def __dd__(self, other): , and you'll return a new Fraction with the result of the addition. Run your new Fraction through the reduce() function before returning.

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

class Fraction
{
public:
    int numerator;
    int denominator;
    // Constructor
    Fraction(int num, int denom)
    {
        assert(typeid(num) == typeid(int));
        assert(typeid(denom) == typeid(int));
        assert(denom != 0);

        numerator = num;
        denominator = denom;
    }

    // Method to reduce the fraction to lowest terms
    int calculateGCD(int a, int b)
    {
        while (b != 0)
        {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }


    void reduce()
    {
        int gcd = calculateGCD(numerator, denominator);
        numerator = numerator/ gcd;
        denominator =denominator/gcd;
    }

```

```cpp
    // Human-readable representation
    string __str__()
    {
        return to_string(numerator) + "/" + to_string(denominator);
    }

    // Representation for Programmer
    string __repl__()
    {
        return "Fraction(" + to_string(numerator) + ", " + to_string(denominator) + ")";
    }


};

int main()
{
    Fraction obj(10,20);

    cout<<"Before using the reduce function:"<<endl;
    cout<<endl;
    cout<<"Human-Readable Representation: "<<obj.__str__()<<endl;
    cout<<"Programmer Readable Representation: "<<obj.__repl__()<<endl;



    cout<<endl;
    cout<<endl;
    cout<<endl;

    cout<<"After using the reduce function:"<<endl;
    cout<<endl;
    obj.reduce();
    cout<<"Human-Readable Representation: "<<obj.__str__()<<endl;
    cout<<"Programmer Readable Representation: "<<obj.__repl__()<<endl;

    return 0;
}
```

```
"C:\Users\atalh\Desktop\Lab Final 2.exe"
Before using the reduce function:

Human-Readable Representation: 10/20
Programmer Readable Representation: Fraction(10, 20)



After using the reduce function:

Human-Readable Representation: 1/2
Programmer Readable Representation: Fraction(1, 2)

Process returned 0 (0x0)   execution time : 0.467 s
Press any key to continue.
```