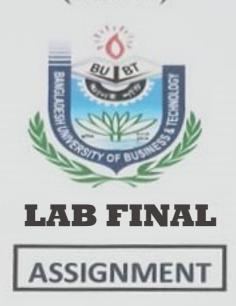
Bangladesh University of Business and Technology (BUBT)



Course code: CSE-121

Course Title: Object Oriented Programming Language

Assignment on:

Submitted By:

Name: SAMAPTY BISWAS

ID: 22235103692

Intake: 51

Section: 01

Submitted To:

Name: KHAN MD. HASIB

Dept.of CSE

BangladeshUniversity of Business

& Technology.

Submission Date: 14-11-2023

Answer to the question no-01

```
#include <iostream>
using namespace std;
class ManchesterUnited {
private:
  int coach;
  int player;
public:
  ManchesterUnited(): coach(0), player(0) {}
  ManchesterUnited(int c, int p) : coach(c), player(p) {}
  void getData() {
    cout << "Coach: " << coach << ", Player: " << player << endl;</pre>
  }
  ManchesterUnited operator++() {
    // Overloading pre-increment operator (++ronaldo)
    coach++;
    player++;
    return *this;
  }
  bool operator==(const ManchesterUnited& other) const {
    // Overloading equality operator (ronaldo == fernandes)
    return (coach == other.coach) && (player == other.player);
  }
```

```
};
int main() {
  ManchesterUnited ronaldo;
  ManchesterUnited fernandes;
  cout << "Before increment:" <<endl;</pre>
  cout << "ronaldo: ";</pre>
  ronaldo.getData();
  cout << "fernandes: ";</pre>
  fernandes.getData();
  // Incrementing ronaldo using overloaded '++' operator
  ++ronaldo;
  cout << "\nAfter increment:" << endl;</pre>
  cout << "ronaldo: ";</pre>
  ronaldo.getData();
  cout << "fernandes: ";</pre>
  fernandes.getData();
  // Comparing ronaldo and fernandes
  if (ronaldo == fernandes) {
    cout << "\nronaldo is equal to fernandes." << endl;</pre>
  } else {
    cout << "\nronaldo is not equal to fernandes." << endl;</pre>
  }
  return 0; }
```

Answer to the question no-02

```
#include <iostream>
#include <cassert>
using namespace std;
class Fraction {
private:
  int numerator;
  int denominator;
public:
 // Constructor
  Fraction(int num, int denom) {
    assert(denom != 0); // Check if denominator is not zero
    assert(typeid(num) == typeid(int) && typeid(denom) == typeid(int)); // Check if both numerator and
denominator are of type int
    numerator = num;
    denominator = denom;
    reduce(); // Reduce the fraction to lowest terms
  }
 // Method to reduce the fraction to lowest terms
 void reduce() {
    int gcd = calculateGCD(numerator, denominator);
    numerator /= gcd;
    denominator /= gcd;
  }
 // Helper method to calculate the greatest common divisor using Euclidean algorithm
```

```
int calculateGCD(int a, int b) {
    while (b != 0) {
      int temp = b;
      b = a \% b;
      a = temp;
    }
    return a;
  }
  // Overriding the + operator
  Fraction operator+(const Fraction& other) const {
    int newNumerator = numerator * other.denominator + other.numerator * denominator;
    int newDenominator = denominator * other.denominator;
    Fraction result(newNumerator, newDenominator);
    return result;
  }
  // Overriding the << operator for easy printing
  friend std::ostream& operator<<(std::ostream& os, const Fraction& frac) {
    os << frac.numerator << "/" << frac.denominator;
    return os;
  }
};
int main() {
  // Example usage
  Fraction frac1(1, 2);
  Fraction frac2(3, 4);
```

```
Fraction result = frac1 + frac2;

std::cout << "Fraction 1: " << frac1 << std::endl;

std::cout << "Fraction 2: " << frac2 << std::endl;

std::cout << "Sum: " << result << std::endl;

return 0;
}</pre>
```