**Question no. 1**

a. The destructor in C++ does not call arguments and must be named just by adding tilde (~) to the starting of class name. So, the correct prototype for the destructor in class Time would be:

**~Time();**

b. In C++ the constructor is named according to the name of the class and it has no return type and also the constructor is not enumerated as a class method. So, the correct prototype for the constructor in class Employee would be:

**Employee(string, string);**

c. There are a couple of issues with the class Example:

→In the function getIncrementedData, the object will not be changeable due to const keyword's declaration to avoid this. However, to add more data,[it is trying] to change the modify one. Therefore, the const keyword there should not be used.

→Method getNum is declared static, indicating that it refers to class members but can not be accessed by the changeable class members. It however attempts to touch the variable, which is dynamic. Thus, data should be eliminated.

Here is the corrected version of the code:

**class Example{**

**public:**

**Example( int y = 10 ): data( y ){**

**// empty body**

**} // end Example constructor**

**int getIncrementedData(){**

**return ++data;**

**} // end function getIncrementedData**

**static int getCount(){**

**cout << "Count is " << count << endl;**

**return count;**

**} // end function getCount**

**private:**

**int data;**

**static int count;**

**}; // end class Example**

**Question no. 2**

→

**#include <iostream>**

**#include <numeric> // for std::gcd**

**class Rational {**

**private:**

**int numerator;**

**int denominator;**

**// Function to reduce the fraction**

**void reduce() {**

**int gcd = std::gcd(numerator, denominator);**

**numerator /= gcd;**

**denominator /= gcd;**

**}**

**public:**

**// Constructor with default values**

**Rational(int num = 0, int den = 1) : numerator(num), denominator(den) {**

**reduce();**

**}**

**// Function to add two Rational numbers**

**Rational add(const Rational& r) {**

**int num = numerator \* r.denominator + denominator \* r.numerator;**

**int den = denominator \* r.denominator;**

**return Rational(num, den);**

**}**

**// Function to subtract two Rational numbers**

**Rational subtract(const Rational& r) {**

**int num = numerator \* r.denominator - denominator \* r.numerator;**

**int den = denominator \* r.denominator;**

**return Rational(num, den);**

**}**

**// Function to multiply two Rational numbers**

**Rational multiply(const Rational& r) {**

**int num = numerator \* r.numerator;**

**int den = denominator \* r.denominator;**

**return Rational(num, den);**

**}**

**// Function to divide two Rational numbers**

**Rational divide(const Rational& r) {**

**int num = numerator \* r.denominator;**

**int den = denominator \* r.numerator;**

**return Rational(num, den);**

**}**

**// Function to print Rational number in the form a/b**

**void print() const {**

**std::cout << numerator << "/" << denominator << std::endl;**

**}**

**// Function to print Rational number in floating-point format**

**void printFloat() const {**

**std::cout << static\_cast<double>(numerator) / denominator << std::endl;**

**}**

**};**

**int main() {**

**Rational r1(2, 4);**

**Rational r2(3, 4);**

**std::cout << "r1 = ";**

**r1.print();**

**std::cout << "r2 = ";**

**r2.print();**

**std::cout << "r1 + r2 = ";**

**r1.add(r2).print();**

**std::cout << "r1 - r2 = ";**

**r1.subtract(r2).print();**

**std::cout << "r1 \* r2 = ";**

**r1.multiply(r2).print();**

**std::cout << "r1 / r2 = ";**

**r1.divide(r2).print();**

**std::cout << "r1 in float = ";**

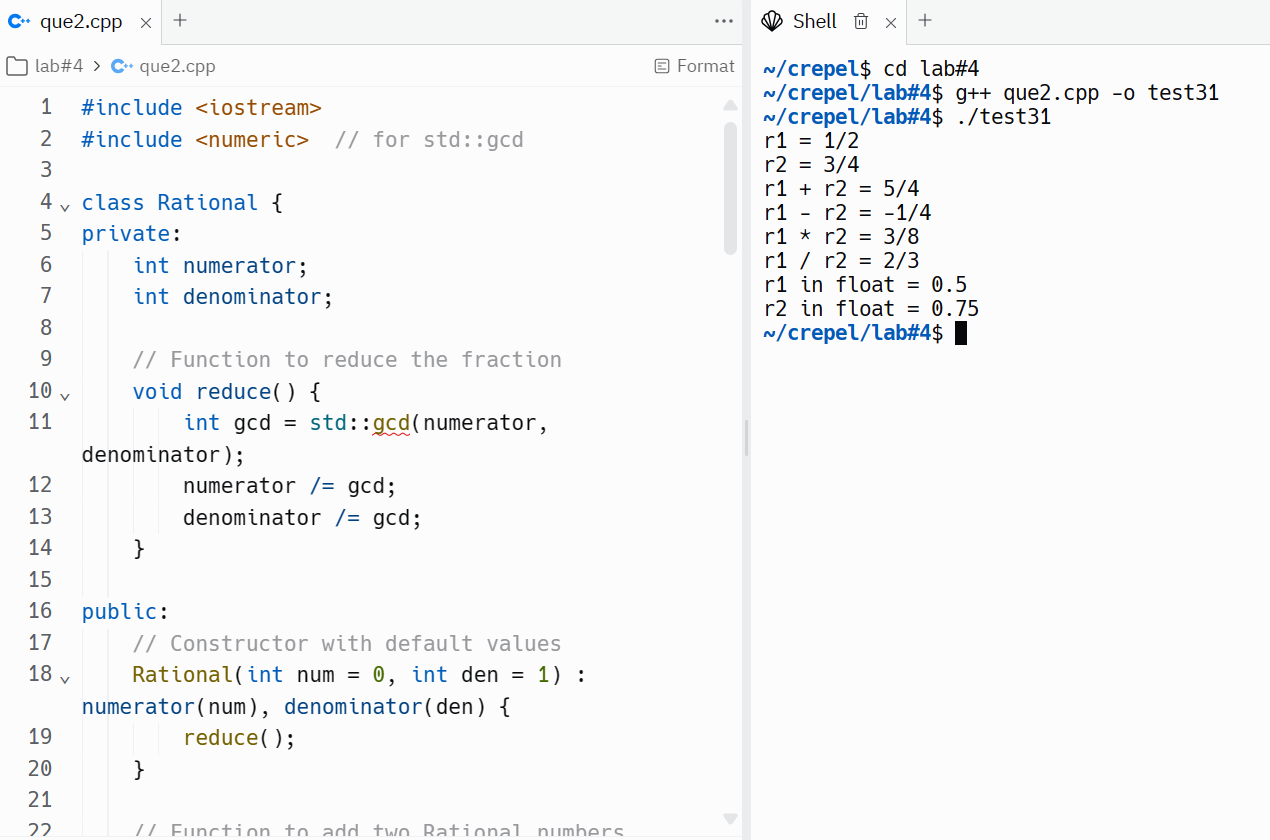
**r1.printFloat();**

**std::cout << "r2 in float = ";**

**r2.printFloat();**

**return 0;**

**}**



**Question no.3**

**#include <array>**

**#include <iostream>**

**class HugeInteger {**

**private:**

**std::array<int, 40> digits;**

**public:**

**// Constructor**

**HugeInteger() : digits({0}) {}**

**// Input function**

**void input(const std::string &num) {**

**std::fill(digits.begin(), digits.end(), 0);**

**int len = num.size();**

**for (int i = 0; i < len; i++) {**

**digits[40 - len + i] = num[i] - '0';**

**}**

**}**

**// Output function**

**void output() const {**

**int i = 0;**

**while (i < 40 && digits[i] == 0)**

**i++;**

**if (i == 40)**

**std::cout << 0;**

**while (i < 40)**

**std::cout << digits[i++];**

**std::cout << std::endl;**

**}**

**// Add function**

**HugeInteger add(const HugeInteger &num) const {**

**HugeInteger result;**

**int carry = 0;**

**for (int i = 39; i >= 0; i--) {**

**int sum = carry + digits[i] + num.digits[i];**

**result.digits[i] = sum % 10;**

**carry = sum / 10;**

**}**

**return result;**

**}**

**// Subtract function**

**HugeInteger subtract(const HugeInteger &num) const {**

**HugeInteger result;**

**int borrow = 0;**

**for (int i = 39; i >= 0; i--) {**

**int diff = digits[i] - num.digits[i] - borrow;**

**if (diff < 0) {**

**diff += 10;**

**borrow = 1;**

**} else {**

**borrow = 0;**

**}**

**result.digits[i] = diff;**

**}**

**return result;**

**}**

**// Comparison functions**

**bool isEqualTo(const HugeInteger &num) const { return digits == num.digits; }**

**bool isNotEqualTo(const HugeInteger &num) const { return !isEqualTo(num); }**

**bool isGreaterThan(const HugeInteger &num) const {**

**return digits > num.digits;**

**}**

**bool isLessThan(const HugeInteger &num) const { return digits < num.digits; }**

**bool isGreaterThanOrEqualTo(const HugeInteger &num) const {**

**return !isLessThan(num);**

**}**

**bool isLessThanOrEqualTo(const HugeInteger &num) const {**

**return !isGreaterThan(num);**

**}**

**// Predicate function**

**bool isZero() const { return digits == std::array<int, 40>({0}); }**

**// Multiply function**

**HugeInteger multiply(const HugeInteger &num) const {**

**// TODO: Implement this function**

**return HugeInteger();**

**}**

**// Divide function**

**HugeInteger divide(const HugeInteger &num) const {**

**// TODO: Implement this function**

**return HugeInteger();**

**}**

**// Modulus function**

**HugeInteger modulus(const HugeInteger &num) const {**

**// TODO: Implement this function**

**return HugeInteger();**

**}**

**};**

**int main() {**

**HugeInteger num1, num2;**

**// Input numbers**

**num1.input("1234567890123456789012345678901234567890");**

**num2.input("9876543210987654321098765432109876543210");**

**// Output numbers**

**std::cout << "Number 1: ";**

**num1.output();**

**std::cout << "Number 2: ";**

**num2.output();**

**// Test addition**

**HugeInteger sum = num1.add(num2);**

**std::cout << "Sum: ";**

**sum.output();**

**// Test subtraction**

**if (num1.isGreaterThan(num2)) {**

**HugeInteger diff = num1.subtract(num2);**

**std::cout << "Difference: ";**

**diff.output();**

**} else {**

**std::cout << "Difference: Not computed, num1 is less than num2."**

**<< std::endl;**

**}**

**// Test comparison functions**

**std::cout << "Are num1 and num2 equal? "**

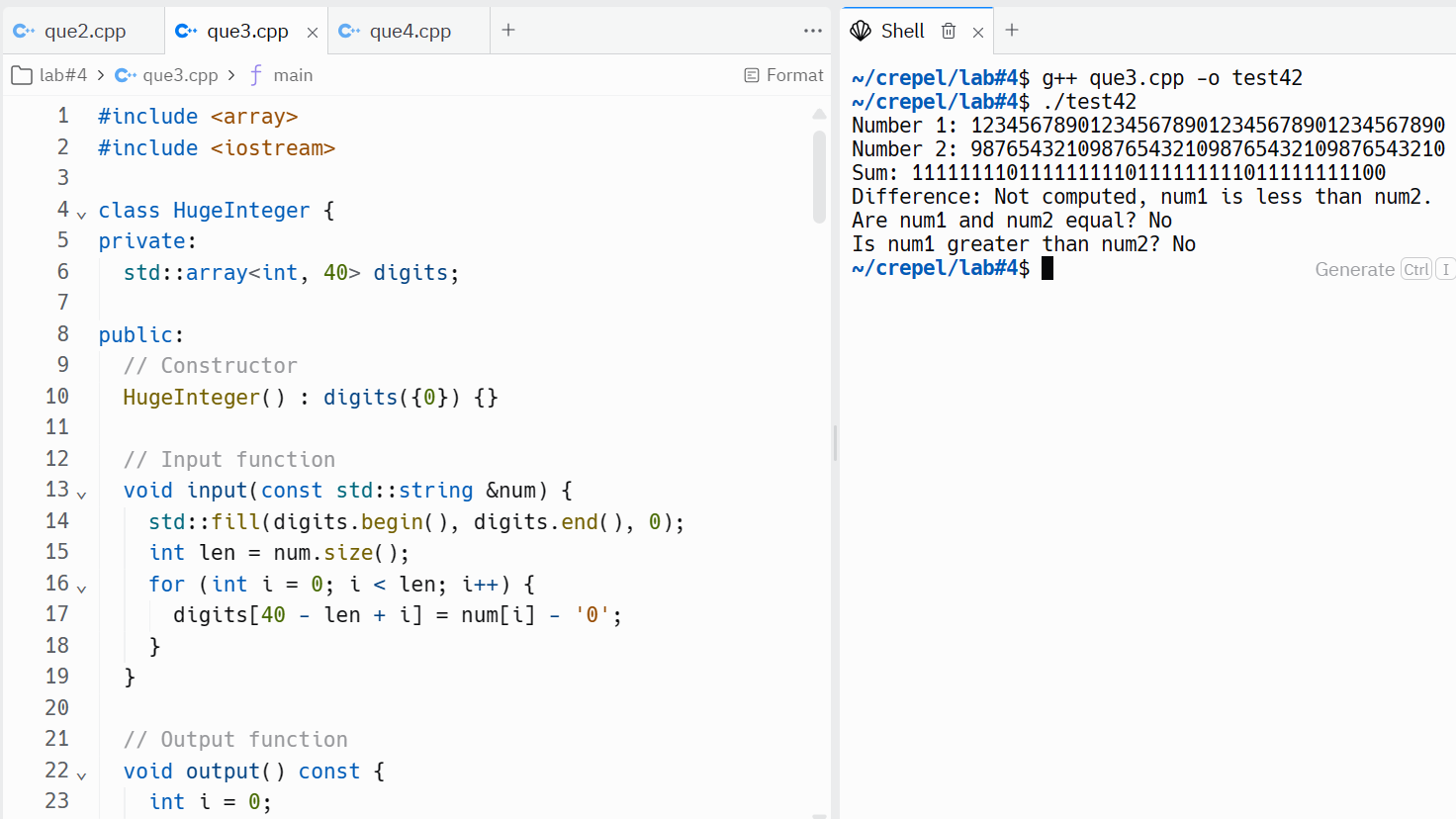
**<< (num1.isEqualTo(num2) ? "Yes" : "No") << std::endl;**

**std::cout << "Is num1 greater than num2? "**

**<< (num1.isGreaterThan(num2) ? "Yes" : "No") << std::endl;**

**return 0;**

**}**



**Question no. 4**

**#include <iostream>**

**class SavingsAccount {**

**private:**

**double savingsBalance;**

**public:**

**static double annualInterestRate;**

**SavingsAccount(double balance) : savingsBalance(balance) {}**

**void calculateMonthlyInterest() {**

**savingsBalance += (savingsBalance \* annualInterestRate / 12.0);**

**}**

**static void modifyInterestRate(double newRate) {**

**annualInterestRate = newRate;**

**}**

**void printBalance() const {**

**std::cout << "Balance: $" << savingsBalance << std::endl;**

**}**

**};**

**double SavingsAccount::annualInterestRate = 0.0;**

**int main() {**

**SavingsAccount saver1(2000.00), saver2(3000.00);**

**SavingsAccount::modifyInterestRate(0.03); // Set annual interest rate to 3%**

**saver1.calculateMonthlyInterest();**

**saver2.calculateMonthlyInterest();**

**std::cout << "Balance after 1 month's (with 3% interest rate):\n";**

**std::cout << "Saver1 ";**

**saver1.printBalance();**

**std::cout << "Saver2 ";**

**saver2.printBalance();**

**SavingsAccount::modifyInterestRate(0.04); // Set annual interest rate to 4%**

**saver1.calculateMonthlyInterest();**

**saver2.calculateMonthlyInterest();**

**std::cout << "Balance after 2 month's (with 4% interest rate):\n";**

**std::cout << "Saver1 ";**

**saver1.printBalance();**

**std::cout << "Saver2 ";**

**saver2.printBalance();**

**return 0;**

**}**

