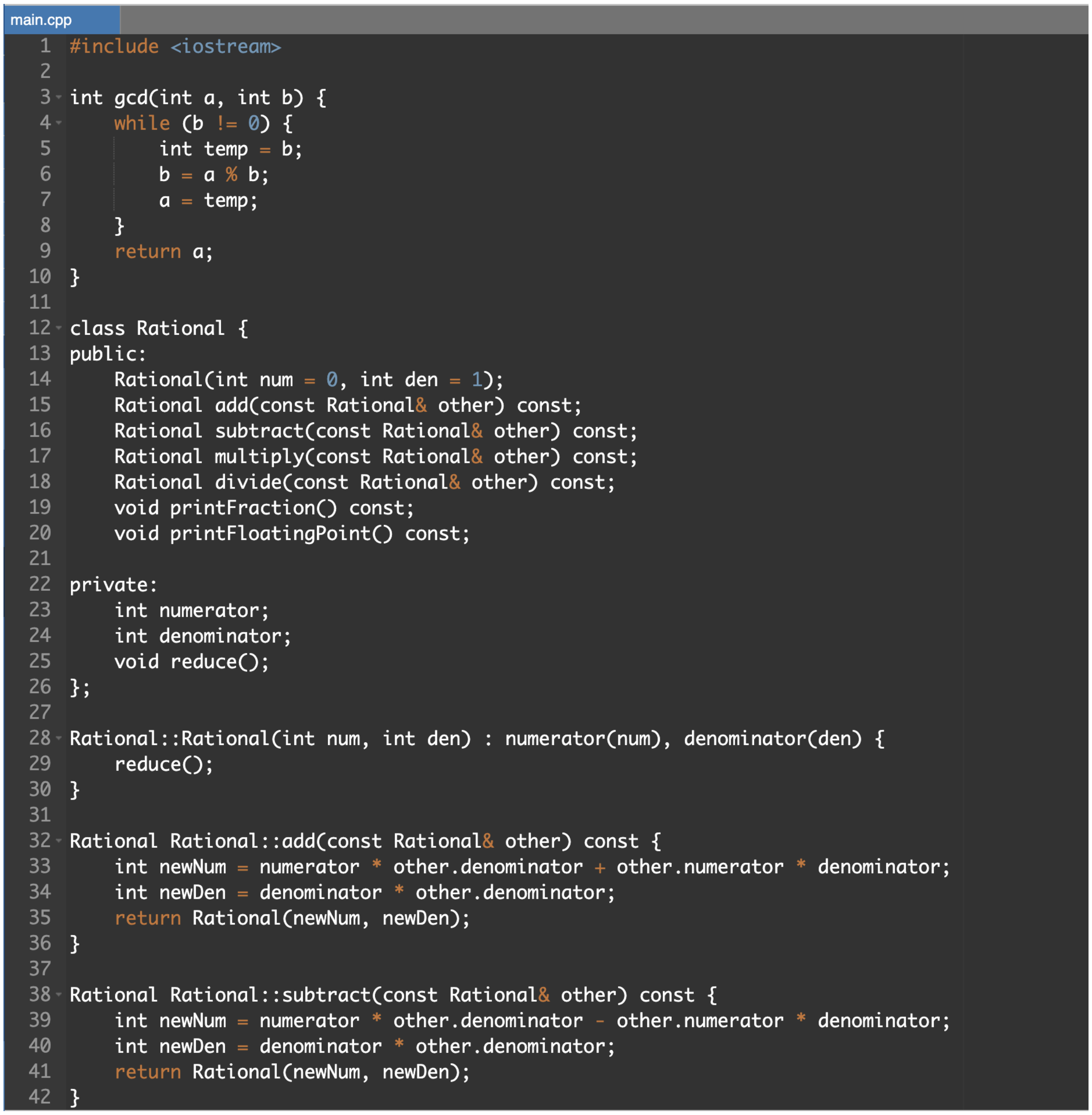
Samir Khadka
CS360L - Programming in C and C++
Lab Assignment #4

Question 1

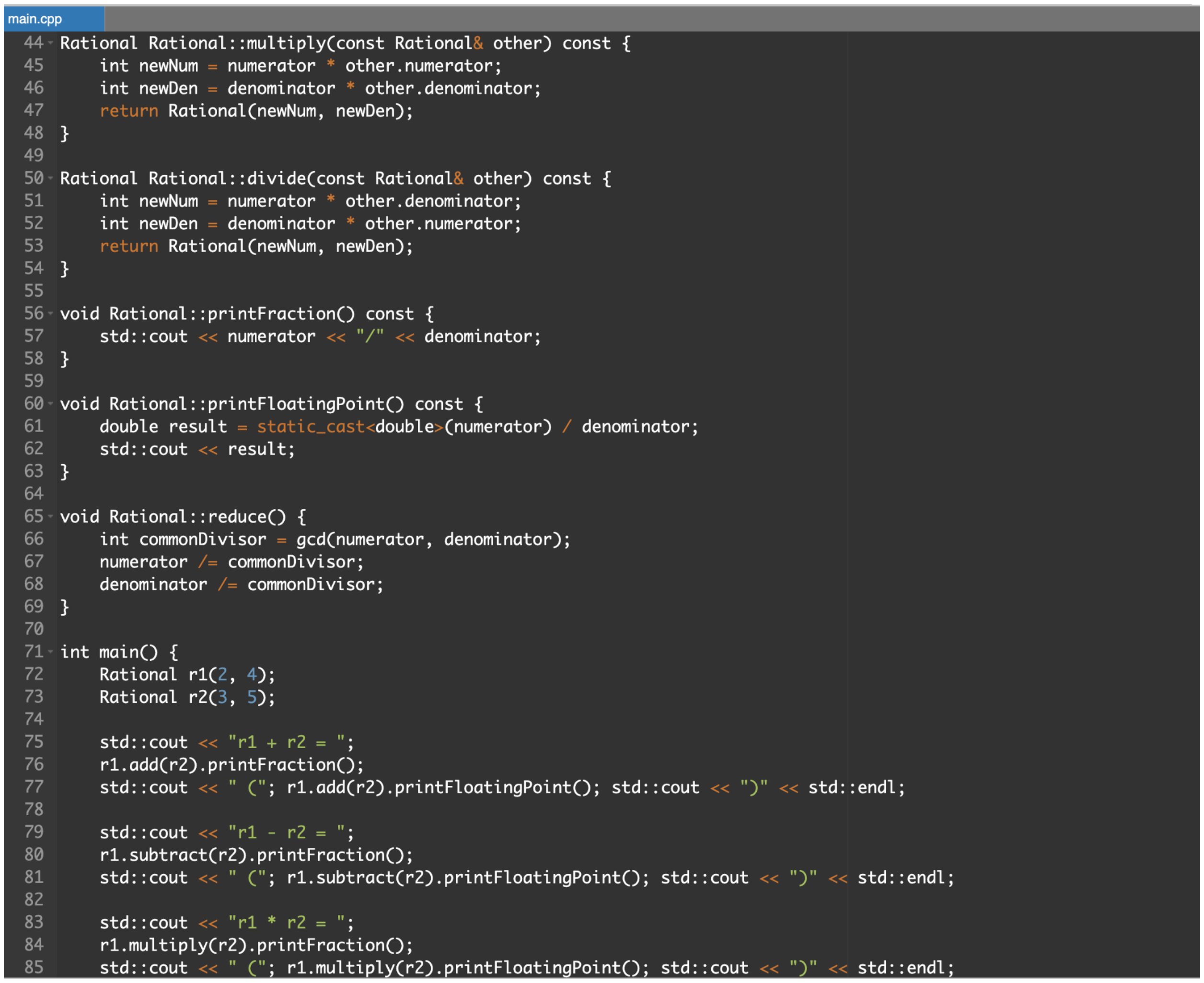
a. Corrected: ~Time();
b. Corrected: Employee(string, string);
c. Corrected:

```
class Example {
public:
    Example(int y = 10) : data(y) {
        // empty body
    } // end Example constructor

    int getIncrementedData() const {
        return ++data;
    } // end function getIncrementedData

    static int getCount() {
        // Note: We can't access 'data' here; it's an instance
variable.
        cout << "Count is " << count << endl;
        return count;
    } // end function getCount

private:
    int data;
    static int count; // Initialize count somewhere (e.g., in the
implementation file).
}; // end class Example
```

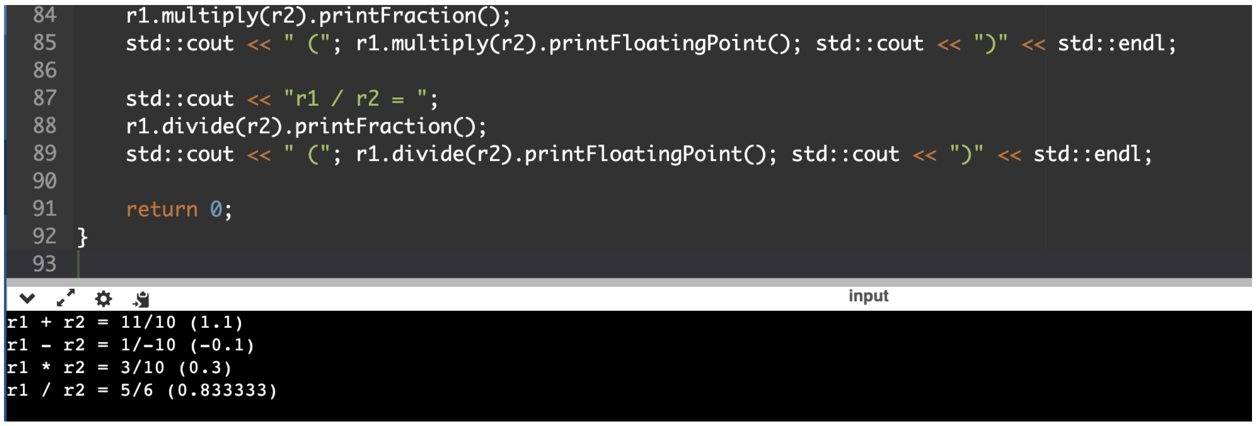
Question 2


main.cpp

```cpp
#include <iostream>

int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

class Rational {
public:
    Rational(int num = 0, int den = 1);
    Rational add(const Rational& other) const;
    Rational subtract(const Rational& other) const;
    Rational multiply(const Rational& other) const;
    Rational divide(const Rational& other) const;
    void printFraction() const;
    void printFloatingPoint() const;

private:
    int numerator;
    int denominator;
    void reduce();
};

Rational::Rational(int num, int den) : numerator(num), denominator(den) {
    reduce();
}

Rational Rational::add(const Rational& other) const {
    int newNum = numerator * other.denominator + other.numerator * denominator;
    int newDen = denominator * other.denominator;
    return Rational(newNum, newDen);
}

Rational Rational::subtract(const Rational& other) const {
    int newNum = numerator * other.denominator - other.numerator * denominator;
    int newDen = denominator * other.denominator;
    return Rational(newNum, newDen);
}
```


main.cpp

```cpp
Rational Rational::multiply(const Rational& other) const {
    int newNum = numerator * other.numerator;
    int newDen = denominator * other.denominator;
    return Rational(newNum, newDen);
}

Rational Rational::divide(const Rational& other) const {
    int newNum = numerator * other.denominator;
    int newDen = denominator * other.numerator;
    return Rational(newNum, newDen);
}

void Rational::printFraction() const {
    std::cout << numerator << "/" << denominator;
}

void Rational::printFloatingPoint() const {
    double result = static_cast<double>(numerator) / denominator;
    std::cout << result;
}

void Rational::reduce() {
    int commonDivisor = gcd(numerator, denominator);
    numerator /= commonDivisor;
    denominator /= commonDivisor;
}

int main() {
    Rational r1(2, 4);
    Rational r2(3, 5);

    std::cout << "r1 + r2 = ";
    r1.add(r2).printFraction();
    std::cout << " ("; r1.add(r2).printFloatingPoint(); std::cout << ")" << std::endl;

    std::cout << "r1 - r2 = ";
    r1.subtract(r2).printFraction();
    std::cout << " ("; r1.subtract(r2).printFloatingPoint(); std::cout << ")" << std::endl;

    std::cout << "r1 * r2 = ";
    r1.multiply(r2).printFraction();
    std::cout << " ("; r1.multiply(r2).printFloatingPoint(); std::cout << ")" << std::endl;
```



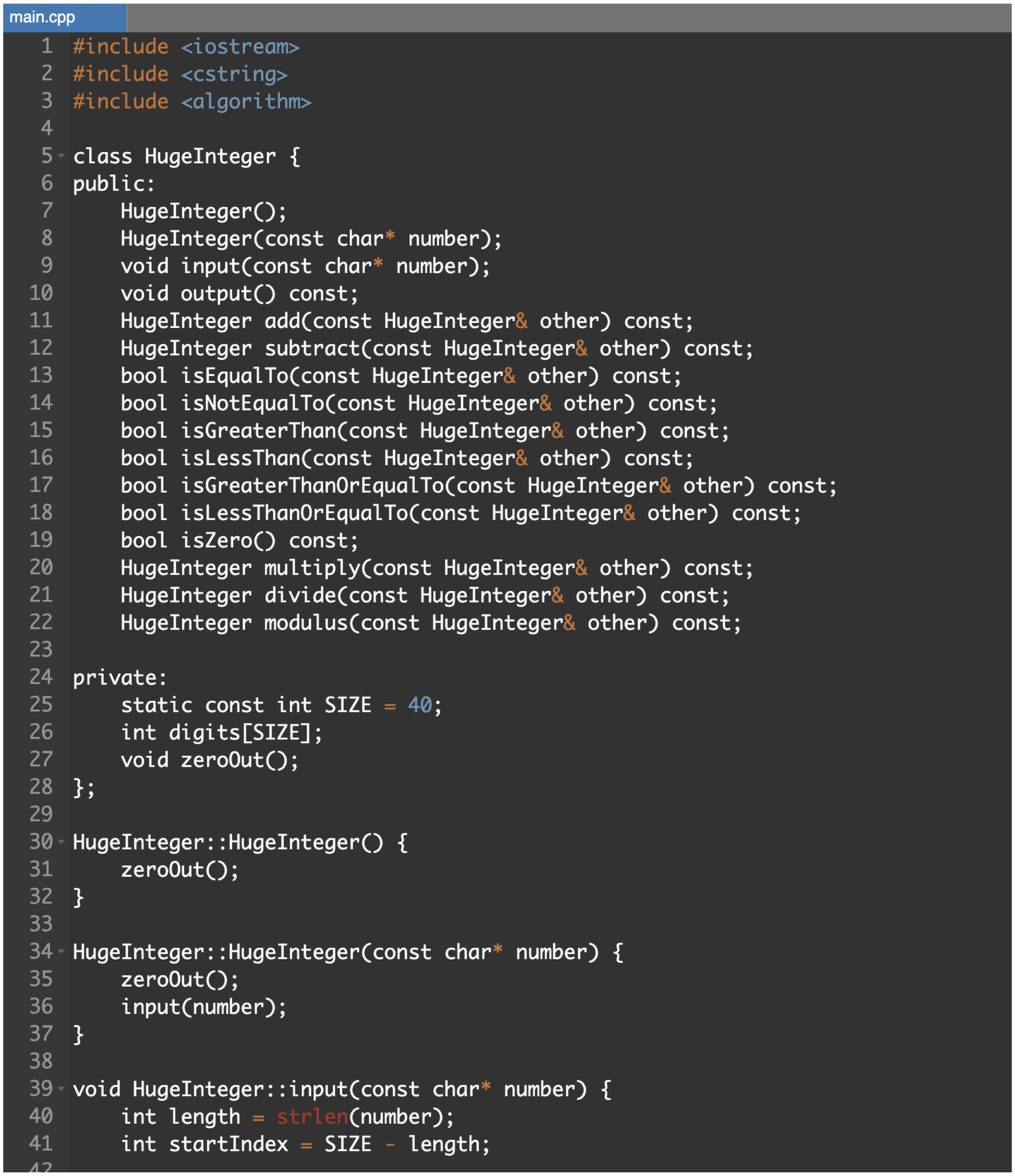
```cpp
    r1.multiply(r2).printFraction();
    std::cout << " ("; r1.multiply(r2).printFloatingPoint(); std::cout << ")" << std::endl;
```

```cpp

    std::cout << "r1 / r2 = ";
    r1.divide(r2).printFraction();
    std::cout << " ("; r1.divide(r2).printFloatingPoint(); std::cout << ")" << std::endl;

    return 0;
}
```

input

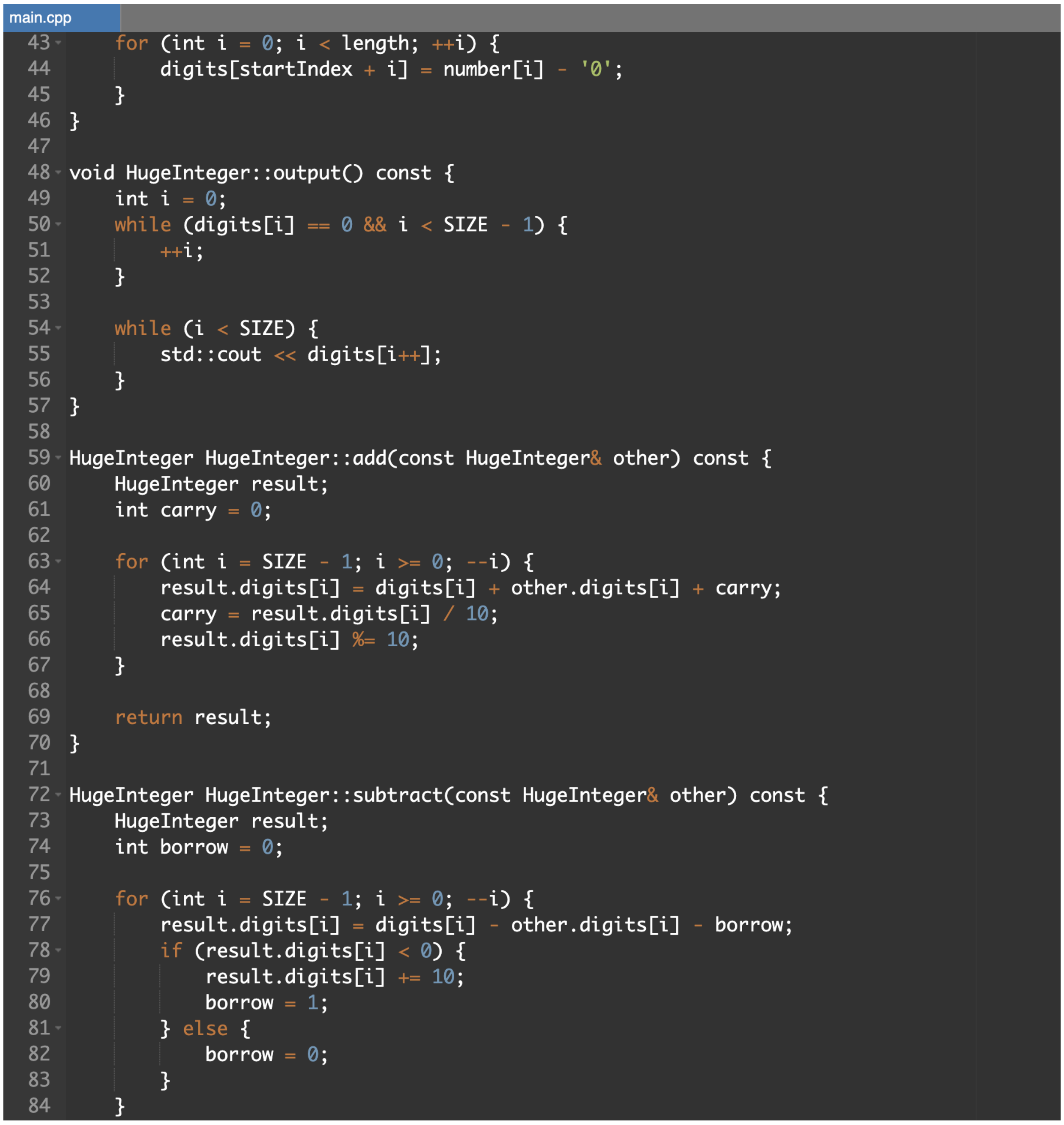
```
r1 + r2 = 11/10 (1.1)
r1 - r2 = 1/-10 (-0.1)
r1 * r2 = 3/10 (0.3)
r1 / r2 = 5/6 (0.833333)
```

Question 3



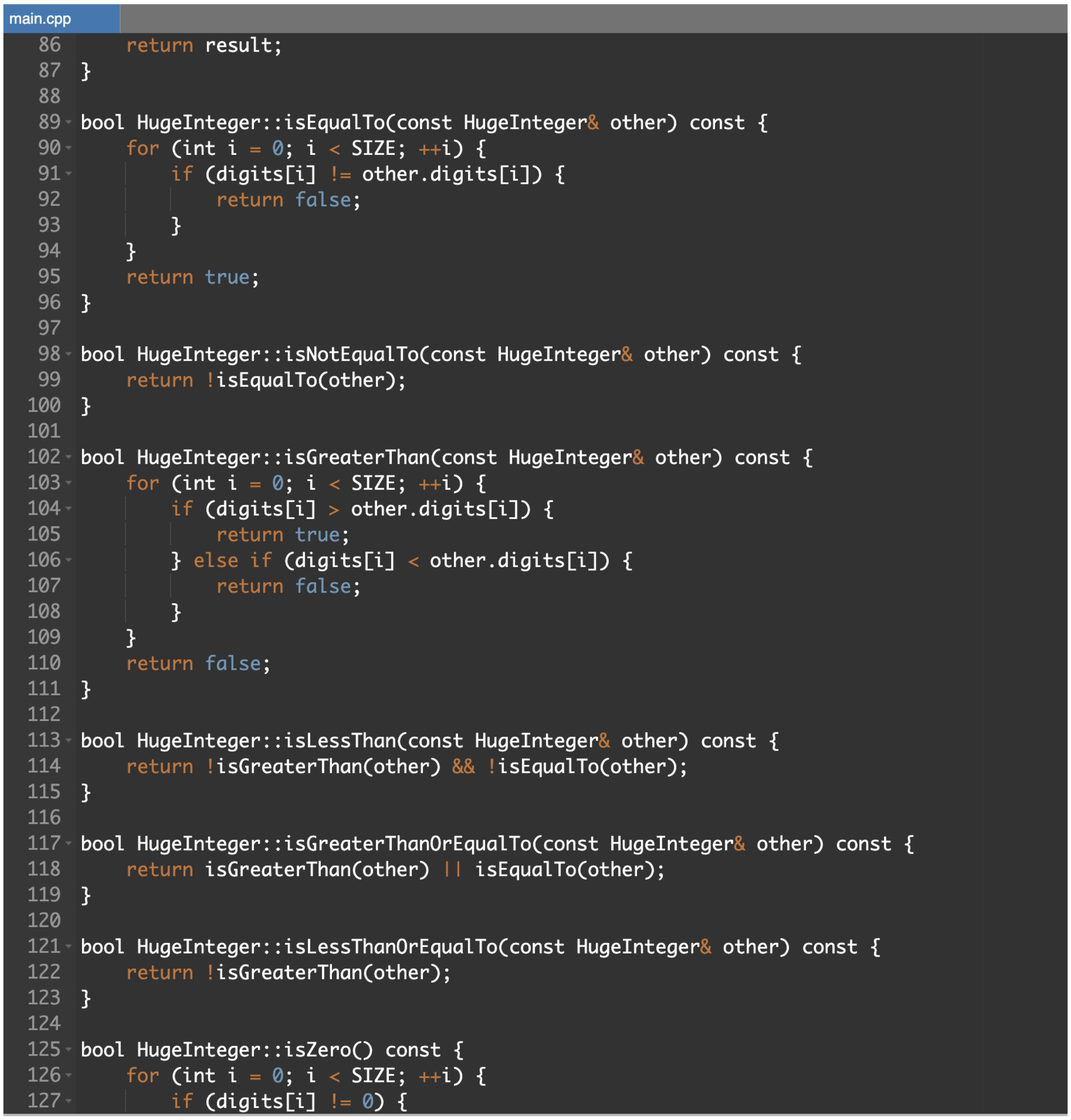
main.cpp

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>

class HugeInteger {
public:
    HugeInteger();
    HugeInteger(const char* number);
    void input(const char* number);
    void output() const;
    HugeInteger add(const HugeInteger& other) const;
    HugeInteger subtract(const HugeInteger& other) const;
    bool isEqualTo(const HugeInteger& other) const;
    bool isNotEqualTo(const HugeInteger& other) const;
    bool isGreaterThan(const HugeInteger& other) const;
    bool isLessThan(const HugeInteger& other) const;
    bool isGreaterThanOrEqualTo(const HugeInteger& other) const;
    bool isLessThanOrEqualTo(const HugeInteger& other) const;
    bool isZero() const;
    HugeInteger multiply(const HugeInteger& other) const;
    HugeInteger divide(const HugeInteger& other) const;
    HugeInteger modulus(const HugeInteger& other) const;

private:
    static const int SIZE = 40;
    int digits[SIZE];
    void zeroOut();
};

HugeInteger::HugeInteger() {
    zeroOut();
}

HugeInteger::HugeInteger(const char* number) {
    zeroOut();
    input(number);
}

void HugeInteger::input(const char* number) {
    int length = strlen(number);
    int startIndex = SIZE - length;
```

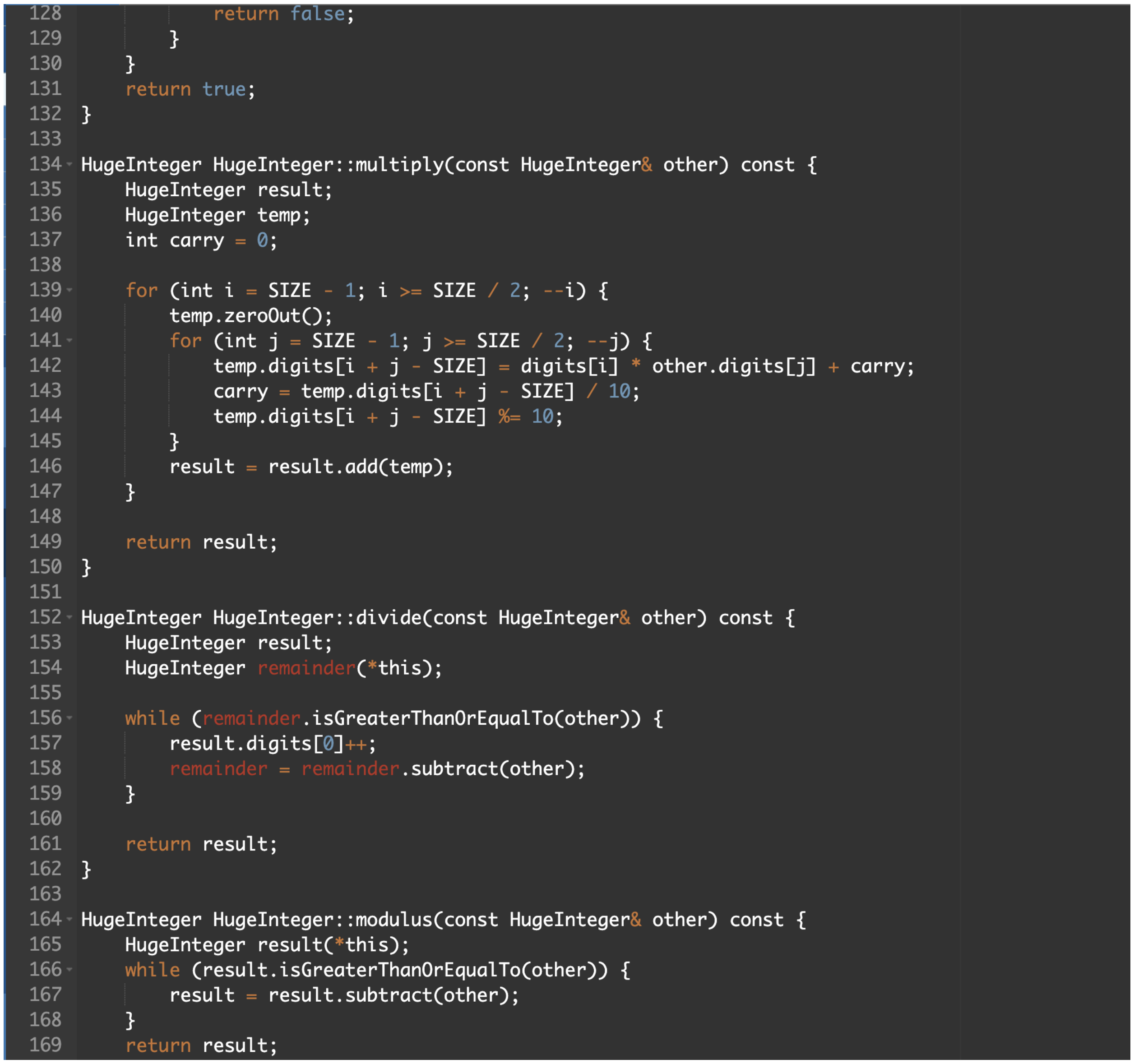


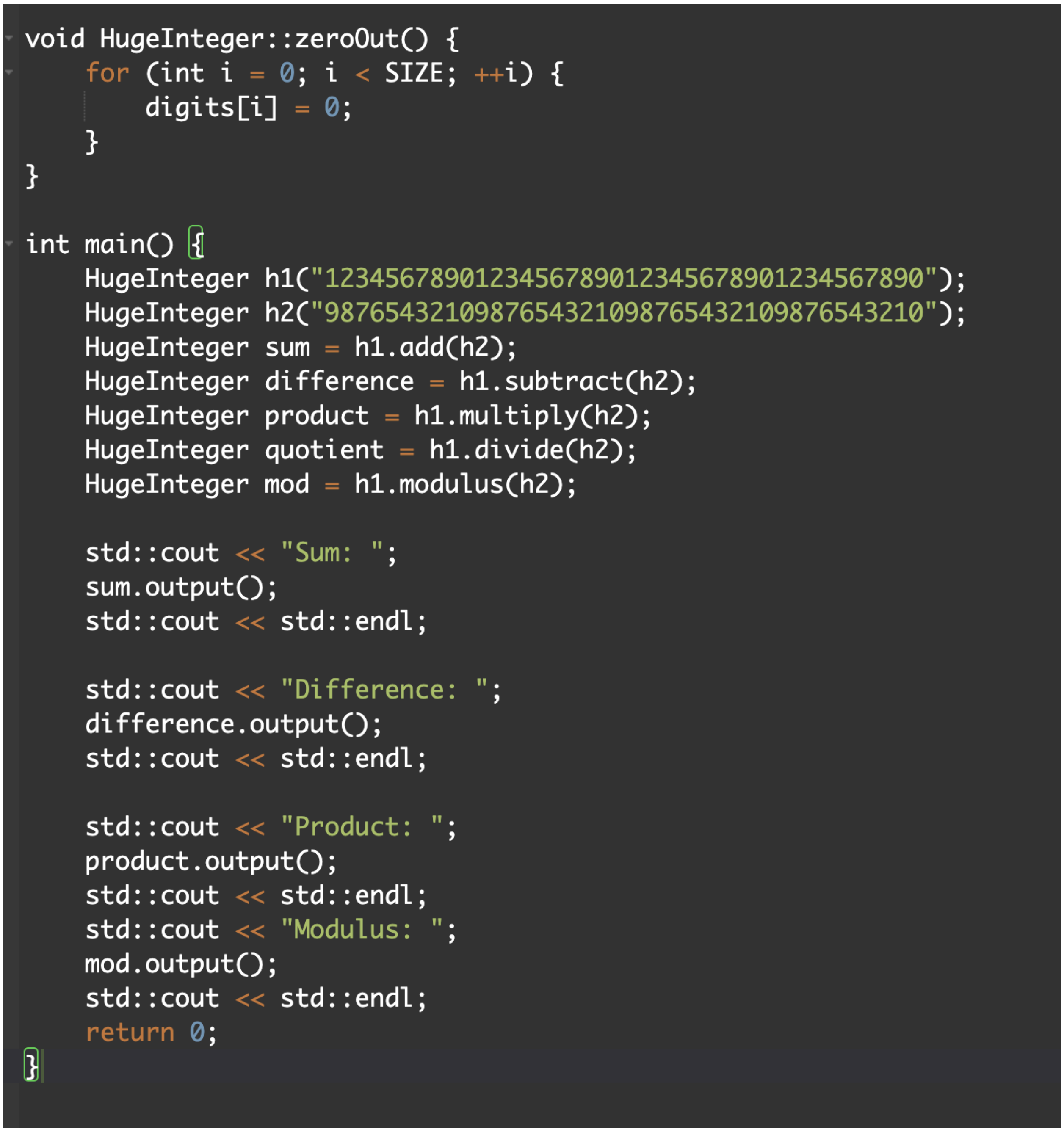
main.cpp

```cpp
    for (int i = 0; i < length; ++i) {
        digits[startIndex + i] = number[i] - '0';
    }
}

void HugeInteger::output() const {
    int i = 0;
    while (digits[i] == 0 && i < SIZE - 1) {
        ++i;
    }

    while (i < SIZE) {
        std::cout << digits[i++];
    }
}

HugeInteger HugeInteger::add(const HugeInteger& other) const {
    HugeInteger result;
    int carry = 0;

    for (int i = SIZE - 1; i >= 0; --i) {
        result.digits[i] = digits[i] + other.digits[i] + carry;
        carry = result.digits[i] / 10;
        result.digits[i] %= 10;
    }

    return result;
}

HugeInteger HugeInteger::subtract(const HugeInteger& other) const {
    HugeInteger result;
    int borrow = 0;

    for (int i = SIZE - 1; i >= 0; --i) {
        result.digits[i] = digits[i] - other.digits[i] - borrow;
        if (result.digits[i] < 0) {
            result.digits[i] += 10;
            borrow = 1;
        } else {
            borrow = 0;
        }
    }
```

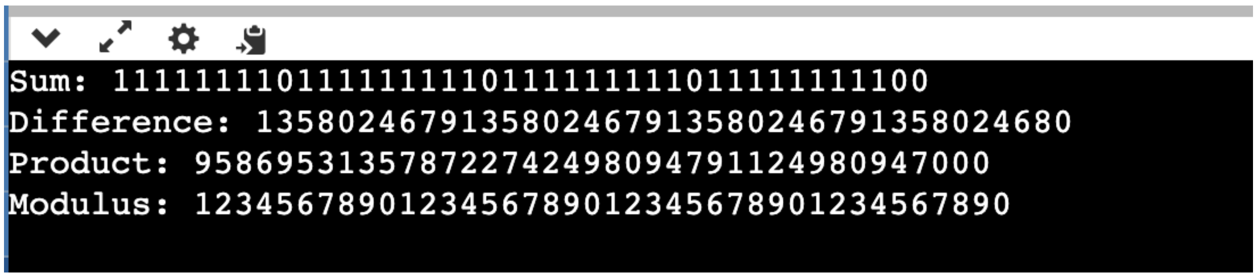

main.cpp

```cpp
    return result;
}

bool HugeInteger::isEqualTo(const HugeInteger& other) const {
    for (int i = 0; i < SIZE; ++i) {
        if (digits[i] != other.digits[i]) {
            return false;
        }
    }
    return true;
}

bool HugeInteger::isNotEqualTo(const HugeInteger& other) const {
    return !isEqualTo(other);
}

bool HugeInteger::isGreaterThan(const HugeInteger& other) const {
    for (int i = 0; i < SIZE; ++i) {
        if (digits[i] > other.digits[i]) {
            return true;
        } else if (digits[i] < other.digits[i]) {
            return false;
        }
    }
    return false;
}

bool HugeInteger::isLessThan(const HugeInteger& other) const {
    return !isGreaterThan(other) && !isEqualTo(other);
}

bool HugeInteger::isGreaterThanOrEqualTo(const HugeInteger& other) const {
    return isGreaterThan(other) || isEqualTo(other);
}

bool HugeInteger::isLessThanOrEqualTo(const HugeInteger& other) const {
    return !isGreaterThan(other);
}

bool HugeInteger::isZero() const {
    for (int i = 0; i < SIZE; ++i) {
        if (digits[i] != 0) {
```


```cpp
            return false;
        }
    }
    return true;
}

HugeInteger HugeInteger::multiply(const HugeInteger& other) const {
    HugeInteger result;
    HugeInteger temp;
    int carry = 0;

    for (int i = SIZE - 1; i >= SIZE / 2; --i) {
        temp.zeroOut();
        for (int j = SIZE - 1; j >= SIZE / 2; --j) {
            temp.digits[i + j - SIZE] = digits[i] * other.digits[j] + carry;
            carry = temp.digits[i + j - SIZE] / 10;
            temp.digits[i + j - SIZE] %= 10;
        }
        result = result.add(temp);
    }

    return result;
}

HugeInteger HugeInteger::divide(const HugeInteger& other) const {
    HugeInteger result;
    HugeInteger remainder(*this);

    while (remainder.isGreaterThanOrEqualTo(other)) {
        result.digits[0]++;
        remainder = remainder.subtract(other);
    }

    return result;
}

HugeInteger HugeInteger::modulus(const HugeInteger& other) const {
    HugeInteger result(*this);
    while (result.isGreaterThanOrEqualTo(other)) {
        result = result.subtract(other);
    }
    return result;
```

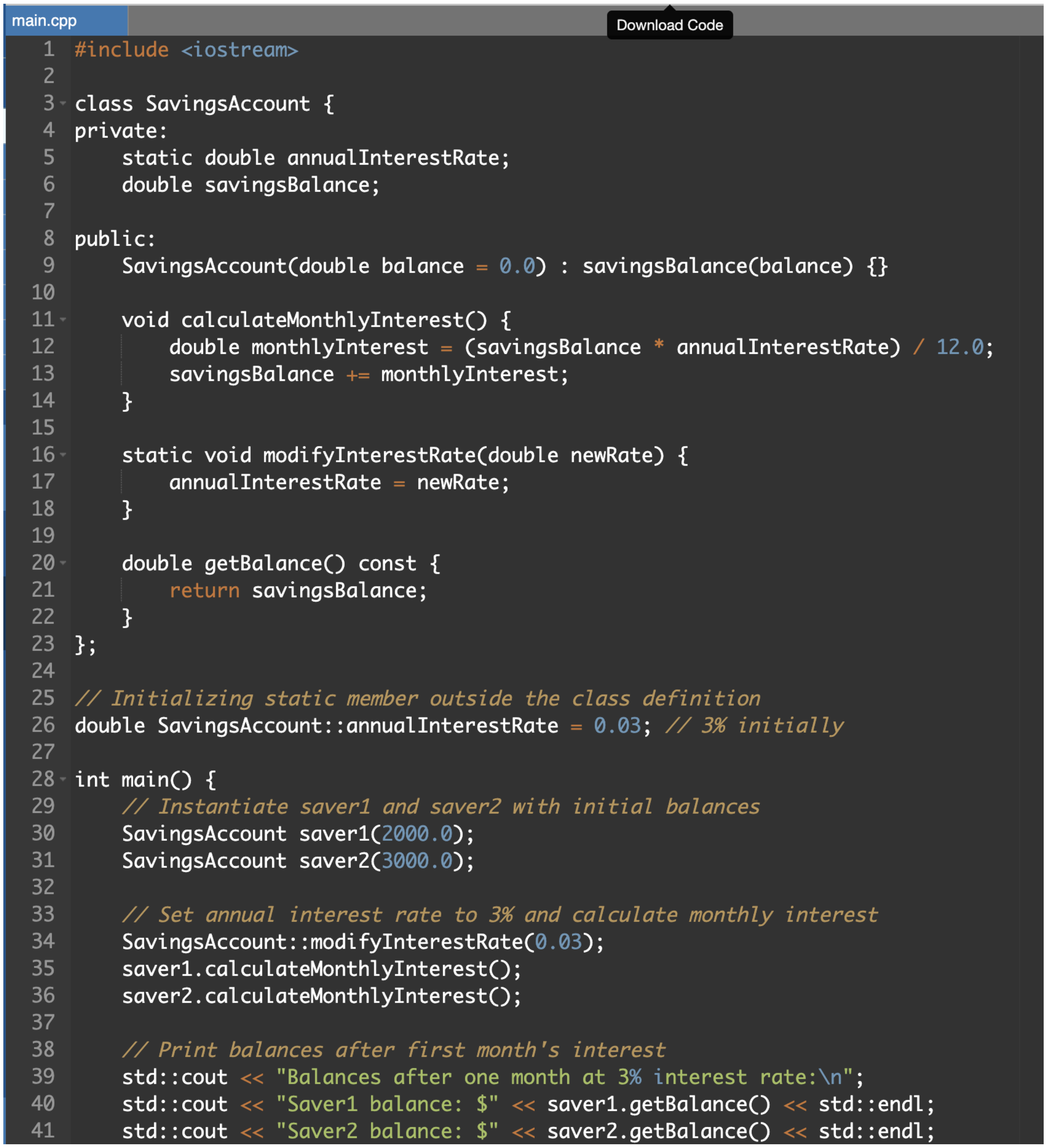


```cpp
void HugeInteger::zeroOut() {
    for (int i = 0; i < SIZE; ++i) {
        digits[i] = 0;
    }
}

int main() {
    HugeInteger h1("1234567890123456789012345678901234567890");
    HugeInteger h2("9876543210987654321098765432109876543210");
    HugeInteger sum = h1.add(h2);
    HugeInteger difference = h1.subtract(h2);
    HugeInteger product = h1.multiply(h2);
    HugeInteger quotient = h1.divide(h2);
    HugeInteger mod = h1.modulus(h2);

    std::cout << "Sum: ";
    sum.output();
    std::cout << std::endl;

    std::cout << "Difference: ";
    difference.output();
    std::cout << std::endl;

    std::cout << "Product: ";
    product.output();
    std::cout << std::endl;
    std::cout << "Modulus: ";
    mod.output();
    std::cout << std::endl;
    return 0;
}
```

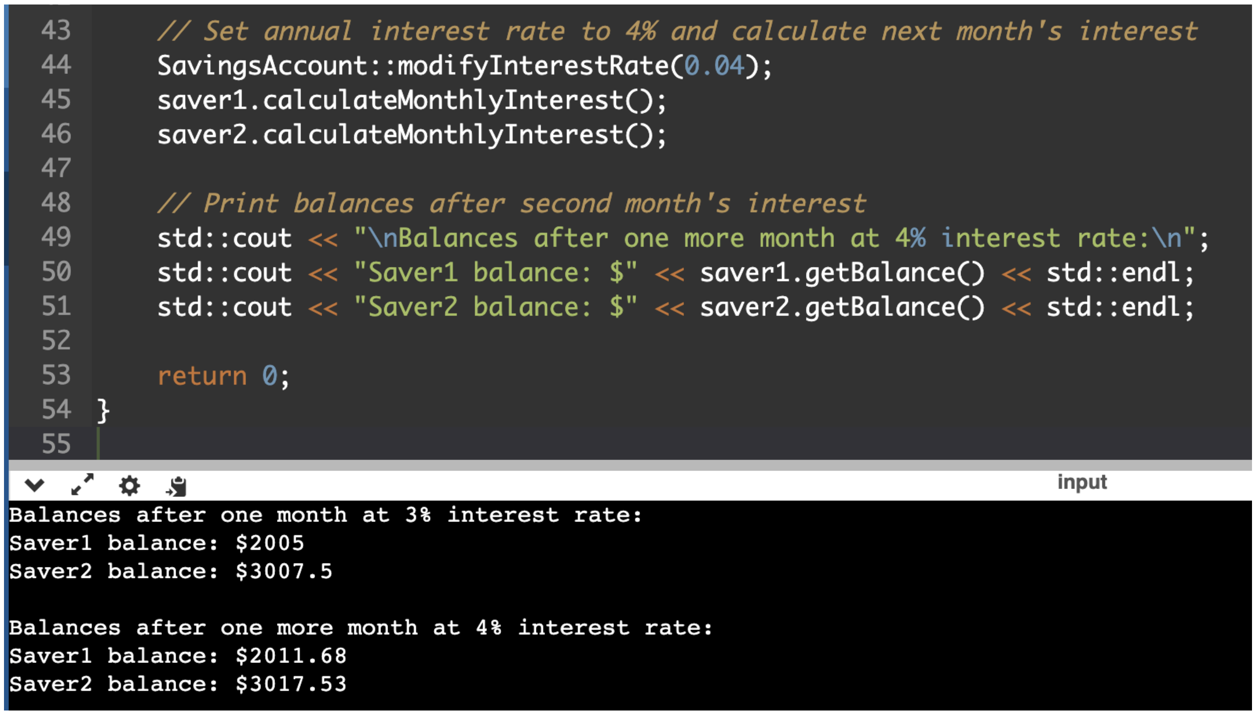


```
Sum: 1111111101111111110111111111011111111100
Difference: 1358024679135802467913580246791358024680
Product: 958695313578722742498094791124980947000
Modulus: 1234567890123456789012345678901234567890
```

Question 4



```cpp
#include <iostream>

class SavingsAccount {
private:
    static double annualInterestRate;
    double savingsBalance;

public:
    SavingsAccount(double balance = 0.0) : savingsBalance(balance) {}

    void calculateMonthlyInterest() {
        double monthlyInterest = (savingsBalance * annualInterestRate) / 12.0;
        savingsBalance += monthlyInterest;
    }

    static void modifyInterestRate(double newRate) {
        annualInterestRate = newRate;
    }

    double getBalance() const {
        return savingsBalance;
    }
};

// Initializing static member outside the class definition
double SavingsAccount::annualInterestRate = 0.03; // 3% initially

int main() {
    // Instantiate saver1 and saver2 with initial balances
    SavingsAccount saver1(2000.0);
    SavingsAccount saver2(3000.0);

    // Set annual interest rate to 3% and calculate monthly interest
    SavingsAccount::modifyInterestRate(0.03);
    saver1.calculateMonthlyInterest();
    saver2.calculateMonthlyInterest();

    // Print balances after first month's interest
    std::cout << "Balances after one month at 3% interest rate:\n";
    std::cout << "Saver1 balance: $" << saver1.getBalance() << std::endl;
    std::cout << "Saver2 balance: $" << saver2.getBalance() << std::endl;
```



```cpp
    // Set annual interest rate to 4% and calculate next month's interest
    SavingsAccount::modifyInterestRate(0.04);
    saver1.calculateMonthlyInterest();
    saver2.calculateMonthlyInterest();

    // Print balances after second month's interest
    std::cout << "\nBalances after one more month at 4% interest rate:\n";
    std::cout << "Saver1 balance: $" << saver1.getBalance() << std::endl;
    std::cout << "Saver2 balance: $" << saver2.getBalance() << std::endl;

    return 0;
}

```

```
Balances after one month at 3% interest rate:
Saver1 balance: $2005
Saver2 balance: $3007.5

Balances after one more month at 4% interest rate:
Saver1 balance: $2011.68
Saver2 balance: $3017.53
```