

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

```
1  #include <iostream>
2
3  int gcd(int a, int b) {
4      while (b != 0) {
5          int temp = b;
6          b = a % b;
7          a = temp;
8      }
9      return a;
10 }
11
12 class Rational {
13 public:
14     Rational(int num = 0, int den = 1);
15     Rational add(const Rational& other) const;
16     Rational subtract(const Rational& other) const;
17     Rational multiply(const Rational& other) const;
18     Rational divide(const Rational& other) const;
19     void printFraction() const;
20     void printFloatingPoint() const;
21
22 private:
23     int numerator;
24     int denominator;
25     void reduce();
26 };
27
28 Rational::Rational(int num, int den) : numerator(num), denominator(den) {
29     reduce();
30 }
31
32 Rational Rational::add(const Rational& other) const {
33     int newNum = numerator * other.denominator + other.numerator * denominator;
34     int newDen = denominator * other.denominator;
35     return Rational(newNum, newDen);
36 }
37
38 Rational Rational::subtract(const Rational& other) const {
39     int newNum = numerator * other.denominator - other.numerator * denominator;
40     int newDen = denominator * other.denominator;
41     return Rational(newNum, newDen);
42 }
```

```
main.cpp
44 Rational Rational::multiply(const Rational& other) const {
45     int newNum = numerator * other.numerator;
46     int newDen = denominator * other.denominator;
47     return Rational(newNum, newDen);
48 }
49
50 Rational Rational::divide(const Rational& other) const {
51     int newNum = numerator * other.denominator;
52     int newDen = denominator * other.numerator;
53     return Rational(newNum, newDen);
54 }
55
56 void Rational::printFraction() const {
57     std::cout << numerator << "/" << denominator;
58 }
59
60 void Rational::printFloatingPoint() const {
61     double result = static_cast<double>(numerator) / denominator;
62     std::cout << result;
63 }
64
65 void Rational::reduce() {
66     int commonDivisor = gcd(numerator, denominator);
67     numerator /= commonDivisor;
68     denominator /= commonDivisor;
69 }
70
71 int main() {
72     Rational r1(2, 4);
73     Rational r2(3, 5);
74
75     std::cout << "r1 + r2 = ";
76     r1.add(r2).printFraction();
77     std::cout << " ("; r1.add(r2).printFloatingPoint(); std::cout << ")" << std::endl;
78
79     std::cout << "r1 - r2 = ";
80     r1.subtract(r2).printFraction();
81     std::cout << " ("; r1.subtract(r2).printFloatingPoint(); std::cout << ")" << std::endl;
82
83     std::cout << "r1 * r2 = ";
84     r1.multiply(r2).printFraction();
85     std::cout << " ("; r1.multiply(r2).printFloatingPoint(); std::cout << ")" << std::endl;
86
87     std::cout << "r1 / r2 = ";
88     r1.divide(r2).printFraction();
89     std::cout << " ("; r1.divide(r2).printFloatingPoint(); std::cout << ")" << std::endl;
90
91     return 0;
92 }
93
```

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

```
1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4
5  class HugeInteger {
6  public:
7      HugeInteger();
8      HugeInteger(const char* number);
9      void input(const char* number);
10     void output() const;
11     HugeInteger add(const HugeInteger& other) const;
12     HugeInteger subtract(const HugeInteger& other) const;
13     bool isEqualTo(const HugeInteger& other) const;
14     bool isNotEqualTo(const HugeInteger& other) const;
15     bool isGreaterThan(const HugeInteger& other) const;
16     bool isLessThan(const HugeInteger& other) const;
17     bool isGreaterThanOrEqualTo(const HugeInteger& other) const;
18     bool isLessThanOrEqualTo(const HugeInteger& other) const;
19     bool isZero() const;
20     HugeInteger multiply(const HugeInteger& other) const;
21     HugeInteger divide(const HugeInteger& other) const;
22     HugeInteger modulus(const HugeInteger& other) const;
23
24 private:
25     static const int SIZE = 40;
26     int digits[SIZE];
27     void zeroOut();
28 };
29
30 HugeInteger::HugeInteger() {
31     zeroOut();
32 }
33
34 HugeInteger::HugeInteger(const char* number) {
35     zeroOut();
36     input(number);
37 }
38
39 void HugeInteger::input(const char* number) {
40     int length = strlen(number);
41     int startIndex = SIZE - length;
42 }
```

main.cpp

```
43 for (int i = 0; i < length; ++i) {
44     digits[startIndex + i] = number[i] - '0';
45 }
46 }
47
48 void HugeInteger::output() const {
49     int i = 0;
50     while (digits[i] == 0 && i < SIZE - 1) {
51         ++i;
52     }
53
54     while (i < SIZE) {
55         std::cout << digits[i++];
56     }
57 }
58
59 HugeInteger HugeInteger::add(const HugeInteger& other) const {
60     HugeInteger result;
61     int carry = 0;
62
63     for (int i = SIZE - 1; i >= 0; --i) {
64         result.digits[i] = digits[i] + other.digits[i] + carry;
65         carry = result.digits[i] / 10;
66         result.digits[i] %= 10;
67     }
68
69     return result;
70 }
71
72 HugeInteger HugeInteger::subtract(const HugeInteger& other) const {
73     HugeInteger result;
74     int borrow = 0;
75
76     for (int i = SIZE - 1; i >= 0; --i) {
77         result.digits[i] = digits[i] - other.digits[i] - borrow;
78         if (result.digits[i] < 0) {
79             result.digits[i] += 10;
80             borrow = 1;
81         } else {
82             borrow = 0;
83         }
84     }
```

main.cpp

```
86     return result;
87 }
88
89 bool HugeInteger::isEqualTo(const HugeInteger& other) const {
90     for (int i = 0; i < SIZE; ++i) {
91         if (digits[i] != other.digits[i]) {
92             return false;
93         }
94     }
95     return true;
96 }
97
98 bool HugeInteger::isNotEqualTo(const HugeInteger& other) const {
99     return !isEqualTo(other);
100 }
101
102 bool HugeInteger::isGreaterThan(const HugeInteger& other) const {
103     for (int i = 0; i < SIZE; ++i) {
104         if (digits[i] > other.digits[i]) {
105             return true;
106         } else if (digits[i] < other.digits[i]) {
107             return false;
108         }
109     }
110     return false;
111 }
112
113 bool HugeInteger::isLessThan(const HugeInteger& other) const {
114     return !isGreaterThan(other) && !isEqualTo(other);
115 }
116
117 bool HugeInteger::isGreaterThanOrEqualTo(const HugeInteger& other) const {
118     return isGreaterThan(other) || isEqualTo(other);
119 }
120
121 bool HugeInteger::isLessThanOrEqualTo(const HugeInteger& other) const {
122     return !isGreaterThan(other);
123 }
124
125 bool HugeInteger::isZero() const {
126     for (int i = 0; i < SIZE; ++i) {
127         if (digits[i] != 0) {
```

```

128         return false;
129     }
130 }
131 return true;
132 }
133
134 HugeInteger HugeInteger::multiply(const HugeInteger& other) const {
135     HugeInteger result;
136     HugeInteger temp;
137     int carry = 0;
138
139     for (int i = SIZE - 1; i >= SIZE / 2; --i) {
140         temp.zeroOut();
141         for (int j = SIZE - 1; j >= SIZE / 2; --j) {
142             temp.digits[i + j - SIZE] = digits[i] * other.digits[j] + carry;
143             carry = temp.digits[i + j - SIZE] / 10;
144             temp.digits[i + j - SIZE] %= 10;
145         }
146         result = result.add(temp);
147     }
148
149     return result;
150 }
151
152 HugeInteger HugeInteger::divide(const HugeInteger& other) const {
153     HugeInteger result;
154     HugeInteger remainder(*this);
155
156     while (remainder.isGreaterThanOrEqualTo(other)) {
157         result.digits[0]++;
158         remainder = remainder.subtract(other);
159     }
160
161     return result;
162 }
163
164 HugeInteger HugeInteger::modulus(const HugeInteger& other) const {
165     HugeInteger result(*this);
166     while (result.isGreaterThanOrEqualTo(other)) {
167         result = result.subtract(other);
168     }
169     return result;

```

```

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: 11111111011111111101111111101111111100
  
Difference: 1358024679135802467913580246791358024680
  
Product: 958695313578722742498094791124980947000
  
Modulus: 1234567890123456789012345678901234567890

## Question 4



```
1  #include <iostream>
2
3  class SavingsAccount {
4  private:
5      static double annualInterestRate;
6      double savingsBalance;
7
8  public:
9      SavingsAccount(double balance = 0.0) : savingsBalance(balance) {}
10
11     void calculateMonthlyInterest() {
12         double monthlyInterest = (savingsBalance * annualInterestRate) / 12.0;
13         savingsBalance += monthlyInterest;
14     }
15
16     static void modifyInterestRate(double newRate) {
17         annualInterestRate = newRate;
18     }
19
20     double getBalance() const {
21         return savingsBalance;
22     }
23 };
24
25 // Initializing static member outside the class definition
26 double SavingsAccount::annualInterestRate = 0.03; // 3% initially
27
28 int main() {
29     // Instantiate saver1 and saver2 with initial balances
30     SavingsAccount saver1(2000.0);
31     SavingsAccount saver2(3000.0);
32
33     // Set annual interest rate to 3% and calculate monthly interest
34     SavingsAccount::modifyInterestRate(0.03);
35     saver1.calculateMonthlyInterest();
36     saver2.calculateMonthlyInterest();
37
38     // Print balances after first month's interest
39     std::cout << "Balances after one month at 3% interest rate:\n";
40     std::cout << "Saver1 balance: $" << saver1.getBalance() << std::endl;
41     std::cout << "Saver2 balance: $" << saver2.getBalance() << std::endl;
```

```

43      // Set annual interest rate to 4% and calculate next month's interest
44      SavingsAccount::modifyInterestRate(0.04);
45      saver1.calculateMonthlyInterest();
46      saver2.calculateMonthlyInterest();
47
48      // Print balances after second month's interest
49      std::cout << "\nBalances after one more month at 4% interest rate:\n";
50      std::cout << "Saver1 balance: $" << saver1.getBalance() << std::endl;
51      std::cout << "Saver2 balance: $" << saver2.getBalance() << std::endl;
52
53      return 0;
54  }
55

```



input

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