Khoi Duong

Prof. Yang

CS360L

6/13/2022

LAB #4

1.

a.

Source code: void ~Time(int);

Code change: void ~Time(~~int~~);

Explanation: the destructor does not take any parameters

b.

Source code: int Employee(string, string);

Code change: int Employee(string a, string b);

Explanation: the parameters in this function should be named (e.g. ‘a’ and ‘b’) in order to be correctly assigned to the right variables.

c.

Source code:

```
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(){
      cout << "Data is " << data << endl;
```

```
      return count;
    } // end function getCount
  private:
    int data;
    static int count;
}; // end class Example
```

Code change:

```
class Example{
  public:
    Example( int y = 10 ): data( y ){
    // empty body
    } // end Example constructor
    int getIncrementedData()const{
      return ++data;
    } // end function getIncrementedData
// Explanation: the function should not have 'const' because
// it has to modify the value of the variable 'data'

    static int getCount(){
// Explanation: the function should be a non-static function instead of a static
function
// Static functions can only access static variables, which cannot access the
variable 'data'
// Non-static functions can access static variables and non-static variables
      cout << "Data is " << data << endl;
      return count;
    } // end function getCount
  private:
    int data;
    static int count;
}; // end class Example
```

2.

Source code:

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

//create a class Rational to perform arithmetic with fractions
```

```
class Rational{
    private:
        int numerator;
        int denominator;

    public:
        // default constructor
        Rational(){
            numerator = 0;
            denominator = 1;
        }
        // constructor with 2 parameters
        Rational(int n, int d): numerator(n), denominator(d){
            if(d == 0){
                throw "Denominator cannot be zero";
            }
        }
        // constructor taking another Rational object as parameter
        Rational(const        Rational&        r):        numerator(r.numerator),
denominator(r.denominator){}

        // store the fraction in reduced form
        void reduce(){
            int div = gcd( numerator, denominator );
            numerator /= div;
            denominator /= div;
        }

        // add two fractions
        Rational operator+( const Rational& rhs ) const{
            int n = numerator * rhs.denominator + rhs.numerator * denominator;
            int d = denominator * rhs.denominator;
            // void reduce is called to store the fraction in reduced form
            Rational result( n, d );
            result.reduce();
            return result;
        }

        // subtract two fractions
        Rational operator-( const Rational& rhs ) const{
            int n = numerator * rhs.denominator - rhs.numerator * denominator;
            int d = denominator * rhs.denominator;
            // void reduce is called to store the fraction in reduced form
            Rational result( n, d );
            result.reduce();
```

```
            return result;
        }

        // multiply two fractions
        Rational operator*( const Rational& rhs ) const{
            int n = numerator * rhs.numerator;
            int d = denominator * rhs.denominator;
            // void reduce is called to store the fraction in reduced form
            Rational result( n, d );
            result.reduce();
            return result;
        }

        // divide two fractions
        Rational operator/( const Rational& rhs ) const{
            int n = numerator * rhs.denominator;
            int d = denominator * rhs.numerator;
            // void reduce is called to store the fraction in reduced form
            Rational result( n, d );
            result.reduce();
            return result;
        }

        // print Rational numbers in float format
        void floatPrint(){
            cout << (float)numerator / denominator << endl;
        }

        // print Rational numbers in fraction format
        void print() const{
            // reduce the fraction to its reduced form before printing
            Rational result( numerator, denominator );
            result.reduce();
            cout << result.numerator << "/" << result.denominator << endl;
        }
};

int main(){
    Rational r1( 15, 25 );
    cout << "r1 fraction format: "; r1.print();
    cout << "r1 float format: "; r1.floatPrint();
    Rational r2( 6, 18 );
    cout << "r2 fraction format: "; r2.print();
    cout << "r2 float format: "; r2.floatPrint();
    Rational r3 = r1 + r2;
```

```
    cout << "r1 + r2 fraction format: "; r3.print();
    cout << "r1 + r2 float format: "; r3.floatPrint();
    Rational r4 = r1 - r2;
    cout << "r1 - r2 fraction format: "; r4.print();
    cout << "r1 - r2 float format: "; r4.floatPrint();
    Rational r5 = r1 * r2;
    cout << "r1 * r2 fraction format: "; r5.print();
    cout << "r1 * r2 float format: "; r5.floatPrint();
    Rational r6 = r1 / r2;
    cout << "r1 / r2 fraction format: "; r6.print();
    cout << "r1 / r2 float format: "; r6.floatPrint();

    return 0;
}
```

Run program & result:

We choose r1 = ⅗ = 0.6 and r2 = ⅓ = 0.3333

```
g++ 2.cpp -o 2 } ; if ($?) { .\2 }
r1 fraction format: 3/5
r1 float format: 0.6
r2 fraction format: 1/3
r2 float format: 0.333333
r1 + r2 fraction format: 14/15
r1 + r2 float format: 0.933333
r1 - r2 fraction format: 4/15
r1 - r2 float format: 0.266667
r1 * r2 fraction format: 1/5
r1 * r2 float format: 0.2
r1 / r2 fraction format: 9/5
r1 / r2 float format: 1.8
PS D:\VS CODE\C C++\CS360L\Lab4>
```

3.

Source code:

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

// class HugeInteger that uses a 40-element array of digits to store integers as
large as 40 digits each
class HugeInteger{
    private:
        int digits[40];

    public:
        // default constructor
        HugeInteger(){
            for(int i = 0; i < 40; i++){
                digits[i] = 0;
            }
        }
        // input function
        void input(){
            string n;
            cin >> n;
            for(int i = 0; i < n.length() ; i++){
                int a = stoi(n.substr((n.length()-1-i), 1));
                digits[i] = a;
            }
        }

        // output function
        void output(){
            for(int i = 39; i >= 0; i--){
                // pass zero located to the left of the first non-zero digit
                if(digits[i] != 0){
                    for(int j = i; j >= 0; j--){
                        cout << digits[j];
                    }
                    break;
                }
                if(digits[i] == 0 && i == 0){
                    cout << 0;
                }
            }
            cout << endl;
        }

        // add two HugeIntegers
        HugeInteger operator+( const HugeInteger& rhs ) const{
```

```
            HugeInteger result;
            int carry = 0;
            for(int i = 0; i < 40; i++){
                result.digits[i] = digits[i] + rhs.digits[i] + carry;
                if(result.digits[i] >= 10){
                    result.digits[i] -= 10;
                    carry = 1;
                }
                else{
                    carry = 0;
                }
            }
            return result;
        }

        // subtract two HugeIntegers
        HugeInteger operator-( const HugeInteger& rhs ) const{
            HugeInteger result;
            if ((*this).isLessThan(rhs)){
                cout << "Error: cannot subtract a larger number from a smaller
number" << endl;
                cout << "The function will take the absolute value of the
difference" << endl;
                result = rhs - *this;
                return result;
            }
            int carry = 0;
            for(int i = 0; i < 40; i++){
                result.digits[i] = digits[i] - rhs.digits[i] - carry;
                if(result.digits[i] < 0){
                    result.digits[i] += 10;
                    carry = 1;
                }
                else{
                    carry = 0;
                }
            }
            return result;
        }

        // multiply two HugeIntegers
        HugeInteger operator*( const HugeInteger& rhs ) const{
            HugeInteger result;
            for(int i = 0; i < 40; i++){
                for(int j = 0; j < 40; j++){
```

```
                result.digits[i + j] += digits[i] * rhs.digits[j];
                if(result.digits[i + j] >= 10){
                    int carry = result.digits[i + j] / 10;
                    result.digits[i + j] %= 10;
                    result.digits[i + j + 1] += carry;
                }
            }
        }
        return result;
    }

    // divide two HugeIntegers
    HugeInteger operator/( const HugeInteger& rhs ) const{
        HugeInteger result;
        if((*this).isLessThan(rhs)){
            result.digits[0] = 0;
        }
        if((*this).isEqualTo(rhs)){
            result.digits[0] = 1;
        }
        HugeInteger temp = *this;
        HugeInteger temp2 = rhs;
        int i = 0;
        while(temp.isGreaterThanOrEqualTo(temp2)){
            temp = temp - temp2;
            i++;
        }
        result.digits[0] = i;
        for (int j = 0; j < 40; j++){
            if (result.digits[j] > 10) {
                result.digits[j + 1] += result.digits[j] / 10;
                result.digits[j] %= 10;
            }
        }
        return result;
    }

    // modulus function
    HugeInteger operator%( const HugeInteger& rhs) const{
        HugeInteger temp = *this;
        HugeInteger temp2 = rhs;
        if (temp.isLessThan(temp2)){
            return temp;
        }
        if (temp.isEqualTo(temp2)){
```

```
            HugeInteger result;
            result.digits[0] = 0;
            return result;
        }
        else {
            while(temp.isGreaterThanOrEqualTo(temp2)){
                temp = temp - temp2;
                }
            return temp;
            }
    }

    // isEqualTo function
    bool isEqualTo( const HugeInteger& rhs ) const{
        for(int i = 0; i < 40; i++){
            if(digits[i] != rhs.digits[i]){
                return false;
            }
        }
        return true;
    }

    // isNotEqualTo function
    bool isNotEqualTo( const HugeInteger& rhs ) const{
        for(int i = 0; i < 40; i++){
            if(digits[i] != rhs.digits[i]){
                return true;
            }
        }
        return false;
    }

    // isGreaterThan function
    bool isGreaterThan( const HugeInteger& rhs ) const{
        for(int i = 39; i >= 0; i--){
            if(digits[i] > rhs.digits[i]){
                return true;
            }
            else if(digits[i] < rhs.digits[i]){
                return false;
            }
        }
        return false;
    }
```

```
        // isLessThan function
        bool isLessThan( const HugeInteger& rhs ) const{
            for(int i = 39; i >= 0; i--){
                if(digits[i] < rhs.digits[i]){
                    return true;
                }
                else if(digits[i] > rhs.digits[i]){
                    return false;
                }
            }
            return false;
        }

        // isGreaterThanOrEqualTo function
        bool isGreaterThanOrEqualTo( const HugeInteger& rhs ) const{
            for(int i = 39; i >= 0; i--){
                if(digits[i] > rhs.digits[i]){
                    return true;
                }
                else if(digits[i] < rhs.digits[i]){
                    return false;
                }
            }
            return true;
        }

        // isLessThanOrEqualTo function
        bool isLessThanOrEqualTo( const HugeInteger& rhs ) const{
            for(int i = 39; i >= 0; i--){
                if(digits[i] < rhs.digits[i]){
                    return true;
                }
                else if(digits[i] > rhs.digits[i]){
                    return false;
                }
            }
            return true;
        }

        // isZero function
        bool isZero() const{
            for(int i = 0; i < 40; i++){
                if(digits[i] != 0){
                    return false;
                }
```

```
            }
            return true;
        }
};

int main(){
    HugeInteger a, b, c;
    cout << "Enter first number: "; a.input();
    cout << "Enter second number: "; b.input();
    cout << "Addition: "; c = a + b; c.output();
    cout << "Subtraction: "; c = a - b; c.output();
    cout << "Multiplication: "; c = a * b; c.output();
    cout << "Division: "; c = a / b; c.output();
    cout << "Modulus: "; c = a % b; c.output();
    cout << "isEqualTo: ";
    if(a.isEqualTo(b)){
        cout << "a is equal to b" << endl;
    }
    else{
        cout << "a is not equal to b" << endl;
    }

    cout << "isNotEqualTo: ";
    if(a.isNotEqualTo(b)){
        cout << "a is not equal to b" << endl;
    }
    else{
        cout << "a is equal to b" << endl;
    }

    cout << "isGreaterThan: ";
    if(a.isGreaterThan(b)){
        cout << "a is greater than b" << endl;
    }
    else{
        cout << "a is not greater than b" << endl;
    }

    cout << "isLessThan: ";
    if(a.isLessThan(b)){
        cout << "a is less than b" << endl;
    }
    else{
        cout << "a is not less than b" << endl;
    }
```

```
    cout << "isGreaterThanOrEqualTo: ";
    if(a.isGreaterThanOrEqualTo(b)){
        cout << "a is greater than or equal to b" << endl;
    }
    else{
        cout << "a is not greater than or equal to b" << endl;
    }

    cout << "isLessThanOrEqualTo: ";
    if(a.isLessThanOrEqualTo(b)){
        cout << "a is less than or equal to b" << endl;
    }
    else{
        cout << "a is not less than or equal to b" << endl;
    }

    cout << "isZero: ";
    if(a.isZero()){
        cout << "a is zero" << endl;
    }
    else{
        cout << "a is not zero" << endl;
    }

    return 0;
}
```

Run program & result:

We choose the first number = 6514681516584 and the second number = 321654

```
PS D:\VS CODE\C C++\CS360L\Lab4> cd "d:\VS CODE\C C++\CS360L\Lab4\"
Enter first number: 6514681516584
Enter second number: 321654
Addition: 6514681838238
Subtraction: 6514681194930
Multiplication: 20954733685353099936
Division: 20253693
Modulus: 148362
isEqualTo: a is not equal to b
isNotEqualTo: a is not equal to b
isGreaterThan: a is greater than b
isLessThan: a is not less than b
isGreaterThanOrEqualTo: a is greater than or equal to b
isLessThanOrEqualTo: a is not less than or equal to b
isZero: a is not zero
PS D:\VS CODE\C C++\CS360L\Lab4>
```

4.

Source code:

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

//create a SavingsAccount class
class SavingsAccount{
    private:
        double savingsBalance;
        static double annual_InterestRate;

    public:
    //default constructor
    SavingsAccount(double a){
        savingsBalance = a;
    }

    // function calculateMonthlyInterest()
    float calculateMonthlyInterest(){
        double monthlyInterest = savingsBalance * (annual_InterestRate / 12);
```

```
        return savingsBalance += monthlyInterest;
    }

    // static function modifyInterestRate()
    static void modifyInterestRate(double new_interest_rate){
        annual_InterestRate = new_interest_rate;
    }

    // function print()
    void print(){
        cout << fixed << setprecision(3) << savingsBalance << endl;
    }
};

//Initialize static variable annual_InterestRate
double SavingsAccount::annual_InterestRate = 0.0;

int main(){
    //create a SavingsAccount object
    SavingsAccount saver1(2000);
    SavingsAccount saver2(3000);

    //set annual interest rate to 3%
    cout << "Annual interest rate is modified to 3%." << endl;
    saver1.modifyInterestRate(0.03);
    saver2.modifyInterestRate(0.03);

    //calculate the monthly interest and print the new balance
    saver1.calculateMonthlyInterest();
    saver2.calculateMonthlyInterest();
    cout << "Saver 1 new balance: "; saver1.print();
    cout << "Saver 2 new balance: "; saver2.print();

    //set the annual interest rate to 4%
    cout << "Annual interest rate is modified to 4%." << endl;
    saver1.modifyInterestRate(0.04);
    saver2.modifyInterestRate(0.04);

    //calculate the monthly interest and print the new balance
    saver1.calculateMonthlyInterest();
    saver2.calculateMonthlyInterest();
    cout << "Saver 1 new balance: "; saver1.print();
    cout << "Saver 2 new balance: "; saver2.print();

    return 0;
```

```
}
```

Run program & result:

```
PS D:\VS CODE\C C++\CS360L\Lab4> cd "d:\VS CODE\C C++\CS360L\Lab4\" ;
 4 } ; if ($?) { .\4 }
Annual interest rate is modified to 3%.
Saver 1 new balance: 2005.000
Saver 2 new balance: 3007.500
Annual interest rate is modified to 4%.
Saver 1 new balance: 2011.683
Saver 2 new balance: 3017.525
PS D:\VS CODE\C C++\CS360L\Lab4>
```