Khoi Duong

Prof. Yang

CS360L

6/30/2022

LAB#6

1.

Source code:

```
// Complex.h
// Complex class definition.
#include <string>
#include <iostream>
using namespace std;
#ifndef COMPLEX_H
#define COMPLEX_H
class Complex{
  public:
    explicit Complex( double = 0.0, double = 0.0 ); // constructor
    Complex operator+( const Complex & ) const; // addition
    Complex operator-( const Complex & ) const; // subtraction
    Complex operator*( const Complex & ) const; // multiplication
    Complex operator/( const Complex & ) const; // division
    friend istream& operator>>(istream & input, Complex & cNumber); // input
    friend ostream& operator<<(ostream & output, const Complex & cNumber); // output
    bool operator==( const Complex & ) const; // equality
    bool operator!=( const Complex & ) const; // inequality
    void print() const; // output
  private:
    double real; // real part
    double imaginary; // imaginary part
}; // end class Complex
#endif
// Complex.cpp
// Complex class member-function definitions.
#include <iostream>
```

```cpp
#include "Complex.h" // Complex class definition
using namespace std;
// Constructor
Complex::Complex( double realPart, double imaginaryPart ):
real( realPart ),imaginary( imaginaryPart ){
 // empty body
} // end Complex constructor
// addition operator
Complex Complex::operator+( const Complex &operand2 ) const{
  return Complex( real + operand2.real,imaginary + operand2.imaginary );
} // end function operator+
// subtraction operator
Complex Complex::operator-( const Complex &operand2 ) const{
  return Complex( real - operand2.real,imaginary - operand2.imaginary );
} // end function operator-
// multiplication operator
Complex Complex::operator*( const Complex &operand2 ) const{
    Complex result;
    result.real = real * operand2.real - imaginary * operand2.imaginary;
    result.imaginary = real * operand2.imaginary + imaginary * operand2.real;
    return result;
} // end function operator*
// division operator
Complex Complex::operator/( const Complex &operand2 ) const{
    Complex result;
    result.real = (real * operand2.real + imaginary * operand2.imaginary) /
(operand2.real * operand2.real + operand2.imaginary * operand2.imaginary);
    result.imaginary = (imaginary * operand2.real - real * operand2.imaginary) /
(operand2.real * operand2.real + operand2.imaginary * operand2.imaginary);
    return result;
} // end function operator/
// input operator
istream & operator>>(istream & input, Complex & cNumber){
  double realPart,imaginaryPart;
  input >> realPart; // read real part
  input >> imaginaryPart; // read imaginary part
  if (input.good()){
    cNumber = Complex( realPart,imaginaryPart ); // construct Complex object
  }
  return input;
} // end function operator>>
// output operator
ostream & operator<<(ostream & output, const Complex & cNumber){
  output << "(" << cNumber.real << ", " << cNumber.imaginary << ")"; // output
Complex object
```

```
  return output;
} // end function operator<<
// display a Complex object in the form: (a, b)
void Complex::print() const{
  cout << '(' << real << ", " << imaginary << ')';
} // end function print
// equality operator
bool Complex::operator==( const Complex &operand2 ) const{
  return ( real == operand2.real && imaginary == operand2.imaginary );
} // end function operator==
// inequality operator
bool Complex::operator!=( const Complex &operand2 ) const{
  return ( real != operand2.real || imaginary != operand2.imaginary );
} // end function operator!=

// main.cpp
// Complex class test program.
#include <iostream>
#include "Complex.h"
using namespace std;
int main(void){
  Complex x;
  Complex y( 4.3, 8.2 );
  Complex z( 3.3, 1.1 );
  Complex w( 4.3, 8.2 );
  cout << "x: ";
  x.print();
  cout << "\ny: ";
  y.print();
  cout << "\nz: ";
  z.print();

  x = y + z;
  cout << "\n\nx = y + z:" << endl;
  x.print();
  cout << " = ";
  y.print();
  cout << " + ";
  z.print();

  x = y - z;
  cout << "\n\nx = y - z:" << endl;
  x.print();
  cout << " = ";
  y.print();
```

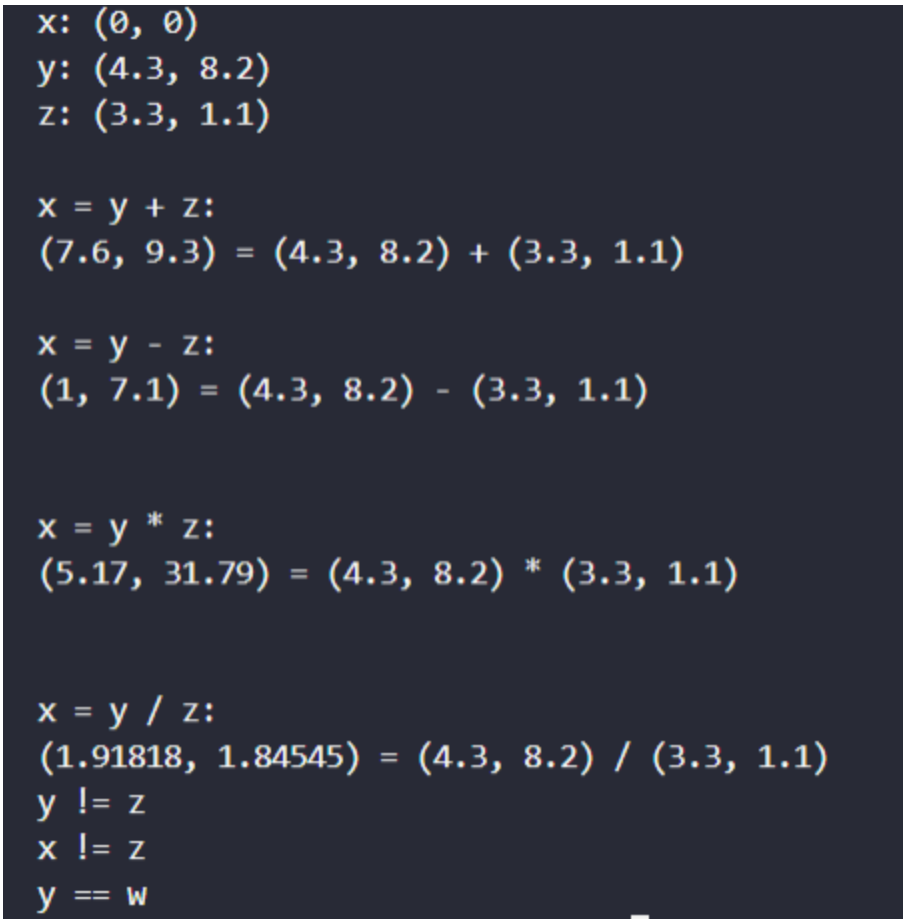
```
  cout << " - ";
  z.print();
  cout << endl;

  x = y * z;
  cout << "\n\nx = y * z:" << endl;
  x.print();
  cout << " = ";
  y.print();
  cout << " * ";
  z.print();
  cout << endl;

    x = y / z;
    cout << "\n\nx = y / z:" << endl;
    x.print();
    cout << " = ";
    y.print();
    cout << " / ";
    z.print();
    cout << endl;

    if (y == z)
        cout << "y == z" << endl;
    else
        cout << "y != z" << endl;
    if (x != z)
        cout << "x != z" << endl;
    else
        cout << "x == z" << endl;
    if (y != w)
        cout << "y != w" << endl;
    else
        cout << "y == w" << endl;
} // end main
```

Run program & result:



2.

a. Describe precisely how it operates.

The program will create support for new data types which will be able to present a larger integer. It has two constructors: a default constructor that converts a long integer into the object, and a constructor that converts the input string into a *HugeInt* object. Following the two constructors, the program will overload the addition operator in order to add the *HugeInt* object with an integer, another object of the same class, or a string that represents a very big number. It also overloads the output operator to print out the result of the object

b. What restrictions does the class have?

Following the private member of the class *HugeInt*, the number is restricted to the maximum number of the variable “digit”. In the program, the variable is limited to 30 digits, Furthermore, one of the restrictions the class may have is the complexity of the program, which may affect the running time of the calculations. For example, a complex program can handle a new data type, but it may run slower if the input number is a mega large number.

c. Overload the * multiplication operator.

d. Overload the / division operator.

e. Overload all the relational and equality operators.

Source code for questions c, d, and e:

HugeInt.h

```
// Hugeint.h
// HugeInt class definition.
#ifndef HUGEINT_H
#define HUGEINT_H

#include <array>
#include <iostream>
#include <string>
using namespace std;
class HugeInt{
  friend std::ostream &operator<<( std::ostream &, const HugeInt & );
  public:
    static const int digits = 30; // maximum digits in a HugeInt
    HugeInt( long = 0 ); // conversion/default constructor
    HugeInt( const std::string & ); // conversion constructor

  // addition operator; HugeInt + HugeInt
    HugeInt operator+( const HugeInt & ) const;
  // addition operator; HugeInt + int
    HugeInt operator+( int ) const;
  // addition operator;
  // HugeInt + string that represents large integer value
    HugeInt operator+( const std::string & ) const;
```

```
    // subtraction operator; HugeInt * HugeInt
    HugeInt operator-( const HugeInt & ) const;
    // subtraction operator; HugeInt * int
    HugeInt operator-( int ) const;
    // subtraction operator; HugeInt * string that represents large integer value
    HugeInt operator-( const std::string & ) const;

  // multiplication operator; HugeInt * HugeInt
    HugeInt operator*( const HugeInt & ) const;
    // multiplication operator; HugeInt * int
    HugeInt operator*( int ) const;
    // multiplication operator; HugeInt * string that represents large integer value
    HugeInt operator*( const std::string & ) const;

    // division operator; HugeInt / HugeInt
    HugeInt operator/( const HugeInt & ) const;
    // division operator; HugeInt / int
    HugeInt operator/( int ) const;
    // division operator; HugeInt / string that represents large integer value
    HugeInt operator/( const std::string & ) const;

    // greater than or equal to operator; HugeInt >= HugeInt
    bool operator>=( const HugeInt & ) const;
    // equality operator; HugeInt == HugeInt
    bool operator==( const HugeInt & ) const;
    // equality operator; HugeInt == int
    bool operator==( int ) const;
    // equality operator; HugeInt == string that represents large integer value
    bool operator==( const std::string & ) const;

    // inequality operator; HugeInt != HugeInt
    bool operator!=( const HugeInt & ) const;
    // inequality operator; HugeInt != int
    bool operator!=( int ) const;
    // inequality operator; HugeInt != string that represents large integer value
    bool operator!=( const std::string & ) const;

  private:
    std::array< short, digits > integer;
}; // end class HugeInt
#endif
```

HugeInt.cpp

```
// Hugeint.cpp
// HugeInt member-function and friend-function definitions.
```

```
#include <cctype> // isdigit function prototype
#include "Hugeint.h" // HugeInt class definition
using namespace std;
// default constructor; conversion constructor that converts
// a long integer into a HugeInt object
HugeInt::HugeInt( long value ){
 // initialize array to zero
  for ( short &element : integer )
    element = 0;
 // place digits of argument into array
  for ( size_t j = digits - 1; value != 0 && j >= 0; j-- ){
    integer[ j ] = value % 10;
    value /= 10;
  } // end for
} // end HugeInt default/conversion constructor
//conversion constructor that converts a character string
//representing a large integer into a HugeInt object
HugeInt::HugeInt( const string &number ){
 //initialize array to zero
  for ( short &element : integer )
    element = 0;
 //place digits of argument into array
  size_t length = number.size();
  for ( size_t j = digits - length, k = 0; j < digits; ++j, ++k )
    if( isdigit( number[ k ] ) ) // ensure that character is a digit
      integer[ j ] = number[ k ] - '0';
}// end HugeInt conversion constructor
 //addition operator; HugeInt + HugeInt
HugeInt HugeInt::operator+( const HugeInt &op2 ) const{
  HugeInt temp; // temporary result
  int carry = 0;
  for ( int i = digits - 1; i >= 0; i-- ){
    temp.integer[ i ] = integer[ i ] + op2.integer[ i ] + carry;
    // determine whether to carry a 1
    if ( temp.integer[ i ] > 9 ){
      temp.integer[ i ] %= 10; // reduce to 0-9
      carry = 1;
    } // end if
    else // no carry
      carry = 0;
  } // end for
  return temp; // return copy of temporary object
} // end function operator+
// addition operator; HugeInt + int
HugeInt HugeInt::operator+( int op2 ) const
```

```
{
// convert op2 to a HugeInt, then invoke
// operator+ for two HugeInt objects
return *this + HugeInt( op2 );
} // end function operator+
// addition operator;
// HugeInt + string that represents large integer value
HugeInt HugeInt::operator+( const string &op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator+ for two HugeInt objects
  return *this + HugeInt( op2 );
} // end operator+

// subtraction operator; HugeInt - HugeInt
HugeInt HugeInt::operator-( const HugeInt &op2 ) const{
  HugeInt temp; // temporary result
  int carry = 0;
  for ( int i = digits - 1; i >= 0; i-- ){
    temp.integer[ i ] = integer[ i ] - op2.integer[ i ] - carry;
    // determine whether to carry a 1
    if ( temp.integer[ i ] < 0 ){
      temp.integer[ i ] += 10; // reduce to 0-9
      carry = 1;
    } // end if
    else // no carry
      carry = 0;
  } // end for
  return temp; // return copy of temporary object
} // end function operator+
// subtraction operator; HugeInt - int
HugeInt HugeInt::operator-( int op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator- for two HugeInt objects
return *this - HugeInt( op2 );
} // end function operator-
// subtraction operator; HugeInt - string that represents large integer value
HugeInt HugeInt::operator-( const string &op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator- for two HugeInt objects
return *this - HugeInt( op2 );
} // end function operator-


// multiplication operator; HugeInt * HugeInt
HugeInt HugeInt::operator*(const HugeInt &op2) const{
```

```
    int arr[100];
    // initialize array to zero
    for ( int i = 0; i < 100; i++ )
        arr[ i ] = 0;
    HugeInt temp; // temporary result
    HugeInt temp1, temp2;
    // reverse the two HugeInt
    for (int i = 0; i < digits; i++) {
      temp1.integer[i] = integer[digits - i - 1];
      temp2.integer[i] = op2.integer[digits - i - 1];
    }

    for(int i = 0; i < digits; i++){
      for(int j = 0; j < digits; j++){
        arr[i + j] += temp1.integer[i] * temp2.integer[j];
        if(arr[i + j] >= 10){
            int carry = arr[i + j] / 10;
            arr[i + j] %= 10;
            arr[i + j + 1] += carry;
        }
      }
    }

    // reverse back to have the result in the correct order
    for(int i = 0; i < digits; i++){
        temp.integer[i] = arr[digits - i - 1];
    }
    return temp;
}
// multiplication operator; HugeInt * int
HugeInt HugeInt::operator*( int op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator* for two HugeInt objects
return *this * HugeInt( op2 );
} // end function operator*
// multiplication operator; HugeInt * string that represents large integer value
HugeInt HugeInt::operator*( const string &op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator* for two HugeInt objects
  return *this * HugeInt( op2 );
} // end function operator*

// greater than or equal operator; HugeInt >= HugeInt
bool HugeInt::operator>=( const HugeInt &op2 ) const{
  for (int i = 0; i < 30 ; i++){
```

```
    if (integer[i] > op2.integer[i]){
        return true;
        }
    else if (integer[i] < op2.integer[i]) {return false;}
    }
    return true;
} // end function operator>=

// division operator; HugeInt / HugeInt
HugeInt HugeInt::operator/(const HugeInt &op2) const{
    HugeInt result;
    HugeInt temp = *this;
    HugeInt temp2 = op2;
    int i = 0;
    while(temp >= temp2){
      temp = temp - temp2;
      i++;
    }
    string number = to_string(i);
    result = HugeInt(number);

    return result;
} // end division operator
// division operator; HugeInt / int
HugeInt HugeInt::operator/( int op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator/ for two HugeInt objects
return *this / HugeInt( op2 );
} // end function operator/
// division operator; HugeInt / string that represents large integer value
HugeInt HugeInt::operator/( const string &op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator/ for two HugeInt objects
  return *this / HugeInt( op2 );
} // end function operator/

// equality operator; HugeInt == HugeInt
bool HugeInt::operator==(const HugeInt &op2) const{
    for (int i = 0; i < 30; i++){
        if (integer[i] != op2.integer[i]){
            return false;
        }
    }
    return true;
} // end function operator==
```

```
// equality operator; HugeInt == int
bool HugeInt::operator==( int op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator== for two HugeInt objects
return *this == HugeInt( op2 );
} // end function operator==
// equality operator; HugeInt == string that represents large integer value
bool HugeInt::operator==( const string &op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator== for two HugeInt objects
  return *this == HugeInt( op2 );
} // end function operator==

// inequality operator; HugeInt != HugeInt
bool HugeInt::operator!=(const HugeInt &op2) const{
    for (int i = 0; i < 30; i++){
        if (integer[i] != op2.integer[i]){
            return true;
        }
    }
    return false;
} // end function operator!=
// inequality operator; HugeInt != int
bool HugeInt::operator!=( int op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator!= for two HugeInt objects
return *this != HugeInt( op2 );
} // end function operator!=
// inequality operator; HugeInt != string that represents large integer value
bool HugeInt::operator!=( const string &op2 ) const{
// convert op2 to a HugeInt, then invoke
// operator!= for two HugeInt objects
  return *this != HugeInt( op2 );
} // end function operator!=


// overloaded output operator
ostream& operator<<( ostream &output, const HugeInt &num ){
  int i;
  for ( i = 0; ( i < HugeInt::digits ) && ( 0 == num.integer[ i ] ); ++i )
    ; // skip leading zeros
  if ( i == HugeInt::digits )
    output << 0;
  else
    for ( ; i < HugeInt::digits; ++i )
```

```
      output << num.integer[ i ];
  return output;
} // end function operator<<
// main.cpp
// HugeInt test program.
#include <iostream>
#include "Hugeint.h"
using namespace std;
int main(void){
  HugeInt n1( 7654321 );
  HugeInt n2( 7891234 );
  HugeInt n3( "99999999999999999999999999999" );
  HugeInt n4( "1" );
  HugeInt n5, n6, n7, n8;
  HugeInt n9( "1234" );

  cout << "n1 is " << n1 << "\nn2 is " << n2
  << "\nn3 is " << n3 << "\nn4 is " << n4
  << "\nn5 is " << n5 << "\n\n";

  n5 = n1 + n2;
  cout << n1 << " + " << n2 << " = " << n5 << "\n\n";

  cout << n3 << " + " << n4 << "\n= " << ( n3 + n4 ) << "\n\n";

  n5 = n1 + 9;
  cout << n1 << " + " << 9 << " = " << n5 << "\n\n";

  n5 = n2 + "10000";
  cout << n2 << " + " << "10000" << " = " << n5 << endl;

  n6 = n2 - n1;
  cout << n2 << " - " << n1 << " = " << n6 << endl;

  cout << n3 << " - " << n4 << "\n= " << ( n3 - n4 ) << endl;

  n6 = n1 - 9;
  cout << n1 << " - " << 9 << " = " << n6 << endl;

  n6 = n2 - "10000";
  cout << n2 << " - " << "10000" << " = " << n6 << endl;

  n7 = n1 * n2;
  cout << n1 << " * " << n2 << " = " << n7 << endl;
```

```
cout << n2 << " * " << n9 << " = " << (n2 * n9) << endl;

n7 = n1 * 9;
cout << n1 << " * " << 9 << " = " << n7 << endl;

n7 = n2 * "10000";
cout << n2 << " * " << "10000" << " = " << n7 << endl;

n8 = n2 / n1;
cout << n2 << " / " << n1 << " = " << n8 << endl;

n8 = n9 / 99;
cout << n9 << " / " << 99 << " = " << n8 << endl;

cout << n1 << " == " << n1 << " ? ";
if ( n1 == n1 )
  cout << "true";
else
  cout << "false";
cout << endl;

cout << n1 << " == " << 9 << " ? ";
if ( n1 == 9 )
  cout << "true";
else
  cout << "false";
cout << endl;

cout << n1 << " == " << "10000" << " ? ";
if ( n1 == "10000" )
  cout << "true";
else
  cout << "false";
cout << endl;

cout << n2 << " != " << n3 << " ? ";
if ( n2 != n3 )
  cout << "true";
else
  cout << "false";
cout << endl;

cout << n2 << " != " << 9 << " ? ";
if ( n2 != 9 )
  cout << "true";
```

```
  else
    cout << "false";
  cout << endl;

  cout << n2 << " != " << "10000" << " ? ";
  if ( n2 != "10000" )
    cout << "true";
  else
    cout << "false";
  cout << endl;

} // end main
```

Run program & result:

```
PS D:\VS CODE\C C++\CS360L\Lab6> cd "d:\VS CODE\C C++\CS360L\Lab6\" ;
n1 is 7654321
n2 is 7891234
n3 is 99999999999999999999999999999
n4 is 1
n5 is 0

7654321 + 7891234 = 15545555

99999999999999999999999999999 + 1
= 100000000000000000000000000000

7654321 + 9 = 7654330

7891234 + 10000 = 7901234
7891234 - 7654321 = 236913
99999999999999999999999999999 - 1
= 99999999999999999999999999998
7654321 - 9 = 7654312
7891234 - 10000 = 7881234
7654321 * 7891234 = 60402038122114
7891234 * 1234 = 9737782756
7654321 * 9 = 68888889
7891234 * 10000 = 78912340000
7891234 / 7654321 = 1
1234 / 99 = 12
7654321 == 7654321 ? true
7654321 == 9 ? false
7654321 == 10000 ? false
7891234 != 99999999999999999999999999999 ? true
7891234 != 9 ? true
7891234 != 10000 ? true
PS D:\VS CODE\C C++\CS360L\Lab6>
```