San Francisco Bay University

CS360L - Programming in C and C++ Lab
Lab Assignment #6

Due day: 4/13/2024

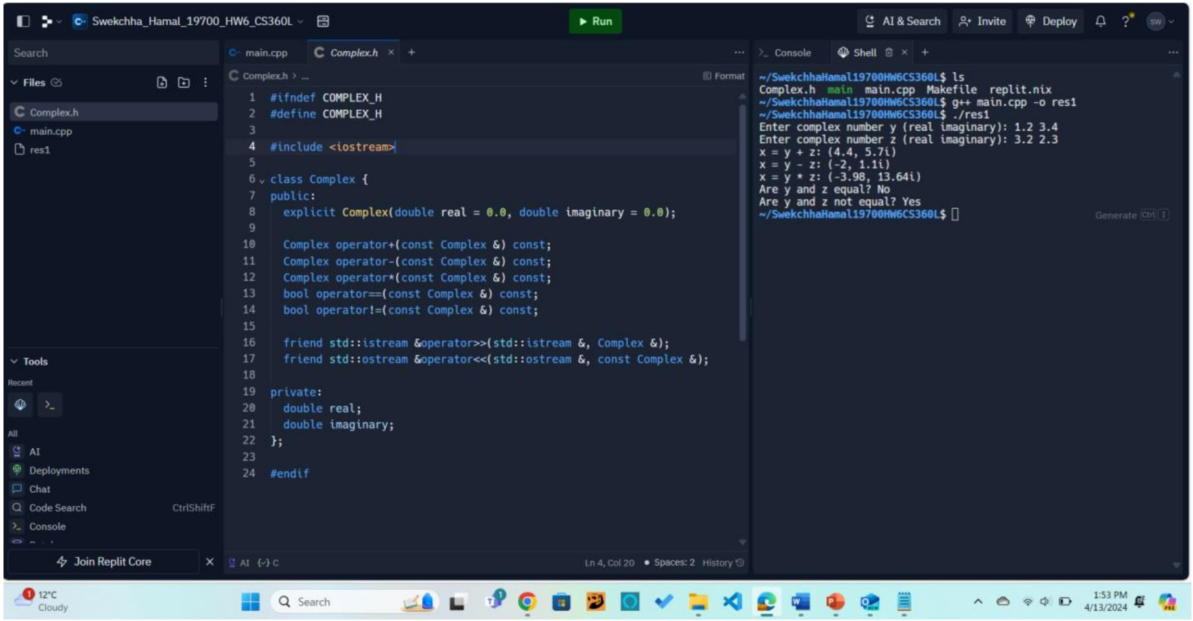
Instruction:

1. Push the answer sheets/source code to Github
2. Please follow the code style rule like programs on handout.
3. Overdue lab assignment submission can't be accepted.
4. Take academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)

1. Consider class *Complex* shown as follows. The class enables operations on so-called *complex numbers*. These are numbers of the form *realPart* + *imaginaryPart*i*, where *i* has the value $\sqrt{-1}$
   a. Modify the class to enable input and output of complex numbers via overloaded >> and << operators, respectively (you should remove the print function from the class).
   b. Overload the multiplication operator to enable multiplication of two complex numbers as in algebra.
   c. Overload the == and != operators to allow comparisons of complex numbers.

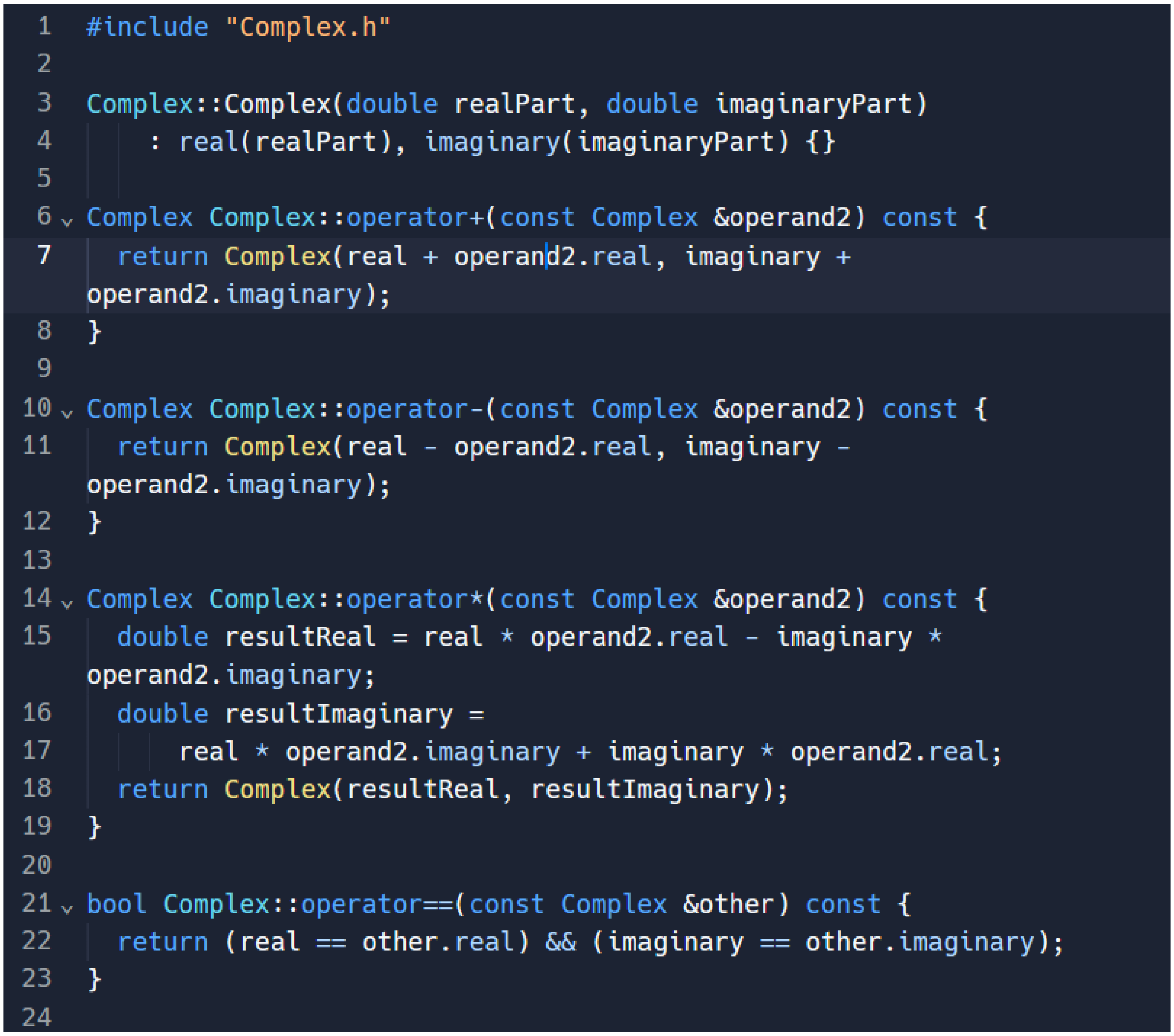
```
// Complex.h
// Complex class definition.
#ifndef COMPLEX_H
#define COMPLEX_H

class Complex{
 public:
   explicit Complex( double = 0.0, double = 0.0 ); // constructor
   Complex operator+( const Complex & ) const; // addition
   Complex operator-( const Complex & ) const; // subtraction
   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>
```

```
#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-

// display a Complex object in the form: (a, b)
void Complex::print() const{
  cout << '(' << real << ", " << imaginary << ')';
} // end function print

// 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 );

  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;
} // end main
```

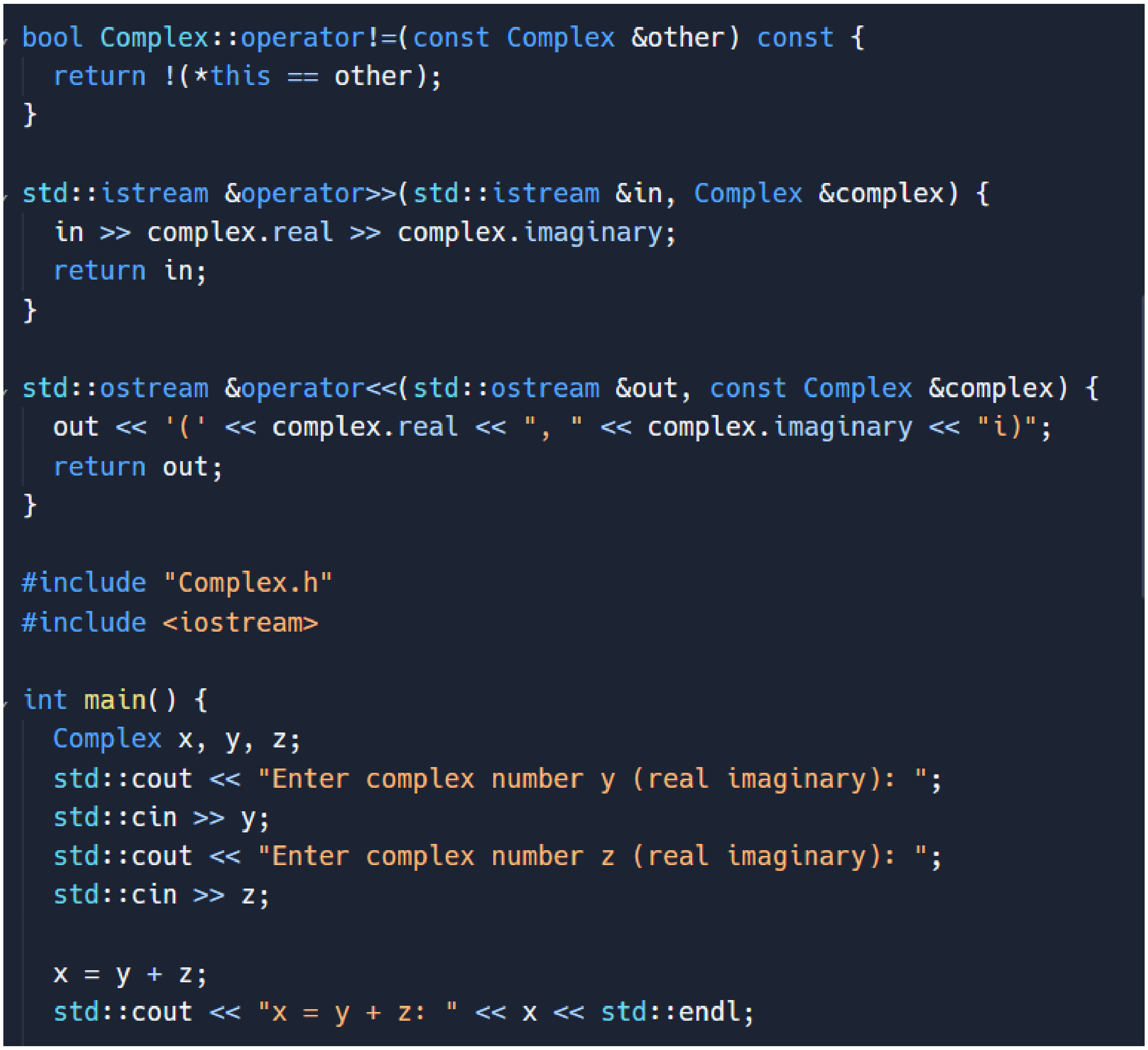


```cpp
#ifndef COMPLEX_H
#define COMPLEX_H

#include <iostream>

class Complex {
public:
  explicit Complex(double real = 0.0, double imaginary = 0.0);

  Complex operator+(const Complex &) const;
  Complex operator-(const Complex &) const;
  Complex operator*(const Complex &) const;
  bool operator==(const Complex &) const;
  bool operator!=(const Complex &) const;

  friend std::istream &operator>>(std::istream &, Complex &);
  friend std::ostream &operator<<(std::ostream &, const Complex &);

private:
  double real;
  double imaginary;
};

#endif
```

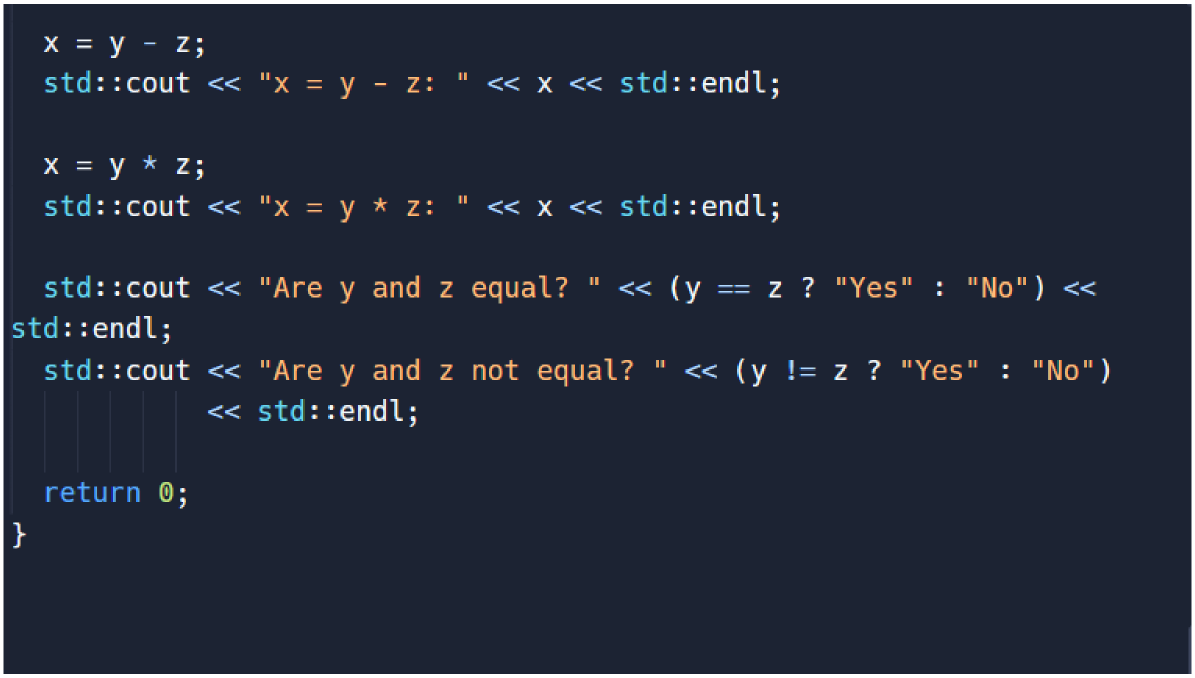
```
~/SwekchhaHamal19700HW6CS360L$ ls
Complex.h  main  main.cpp  Makefile  replit.nix
~/SwekchhaHamal19700HW6CS360L$ g++ main.cpp -o res1
~/SwekchhaHamal19700HW6CS360L$ ./res1
Enter complex number y (real imaginary): 1.2 3.4
Enter complex number z (real imaginary): 3.2 2.3
x = y + z: (4.4, 5.7i)
x = y - z: (-2, 1.1i)
x = y * z: (-3.98, 13.64i)
Are y and z equal? No
Are y and z not equal? Yes
~/SwekchhaHamal19700HW6CS360L$
```



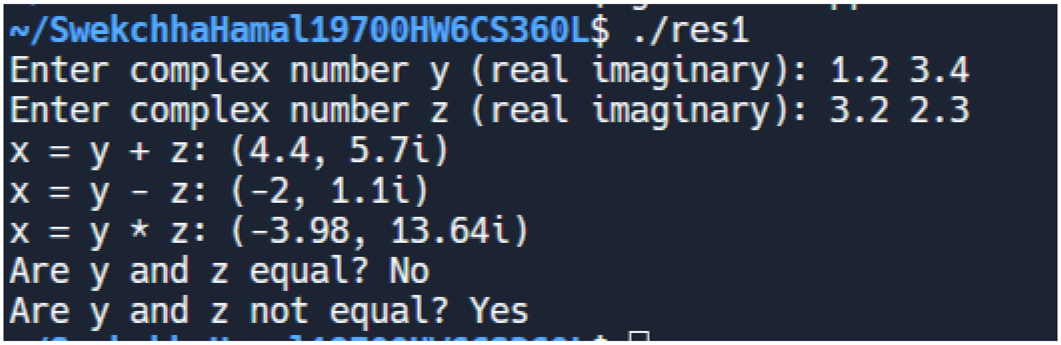
```cpp
#include "Complex.h"

Complex::Complex(double realPart, double imaginaryPart)
    : real(realPart), imaginary(imaginaryPart) {}

Complex Complex::operator+(const Complex &operand2) const {
  return Complex(real + operand2.real, imaginary + operand2.imaginary);
}

Complex Complex::operator-(const Complex &operand2) const {
  return Complex(real - operand2.real, imaginary - operand2.imaginary);
}

Complex Complex::operator*(const Complex &operand2) const {
  double resultReal = real * operand2.real - imaginary * operand2.imaginary;
  double resultImaginary =
      real * operand2.imaginary + imaginary * operand2.real;
  return Complex(resultReal, resultImaginary);
}

bool Complex::operator==(const Complex &other) const {
  return (real == other.real) && (imaginary == other.imaginary);
}

```



```cpp
bool Complex::operator!=(const Complex &other) const {
  return !(*this == other);
}

std::istream &operator>>(std::istream &in, Complex &complex) {
  in >> complex.real >> complex.imaginary;
  return in;
}

std::ostream &operator<<(std::ostream &out, const Complex &complex) {
  out << '(' << complex.real << ", " << complex.imaginary << "i)";
  return out;
}

#include "Complex.h"
#include <iostream>

int main() {
  Complex x, y, z;
  std::cout << "Enter complex number y (real imaginary): ";
  std::cin >> y;
  std::cout << "Enter complex number z (real imaginary): ";
  std::cin >> z;

  x = y + z;
  std::cout << "x = y + z: " << x << std::endl;
```



```
  x = y - z;
  std::cout << "x = y - z: " << x << std::endl;

  x = y * z;
  std::cout << "x = y * z: " << x << std::endl;

  std::cout << "Are y and z equal? " << (y == z ? "Yes" : "No") <<
std::endl;
  std::cout << "Are y and z not equal? " << (y != z ? "Yes" : "No")
            << std::endl;

  return 0;
}
```



```
~/SwekchhaHamal19700HW6CS360L$ ./res1
Enter complex number y (real imaginary): 1.2 3.4
Enter complex number z (real imaginary): 3.2 2.3
x = y + z: (4.4, 5.7i)
x = y - z: (-2, 1.1i)
x = y * z: (-3.98, 13.64i)
Are y and z equal? No
Are y and z not equal? Yes
```

2. A machine with *32-bit* integers can represent integers in the range of approximately *-2* billion to *+2* billion. This fixed-size restriction is rarely troublesome, but there are applications in which we would like to be able to use a much wider range of integers. This is what C++ was built to do, namely, create powerful new data types. Consider class *HugeInt* in the following program. Study the class carefully, then answer the following:

   a. Describe precisely how it operates.

➔ HugeInt class is made to handle very large integers that exceed the built-in range of data type int or long. It uses array of short integers to store each digits of large one internally.

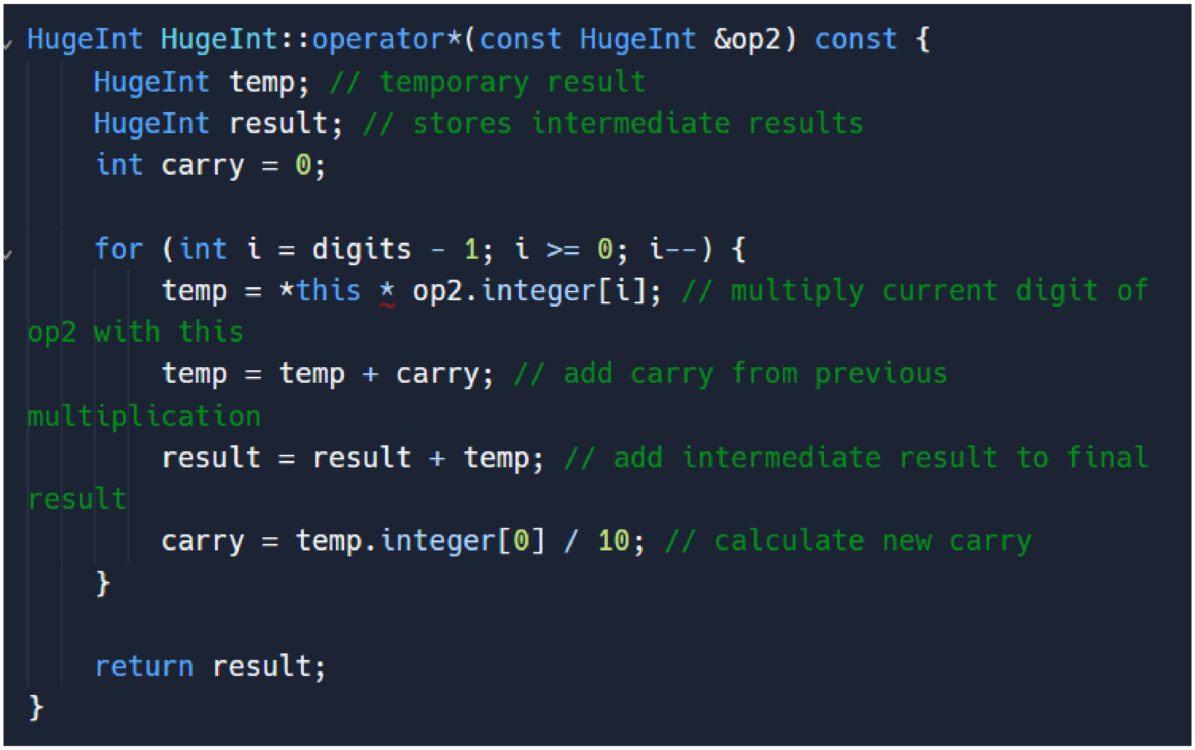
Constructor for that converts long integer to HugeInt and another constructor takes string of large integer and converts it into HugeInt.

Operator+ overloads the addition of HugeInt objects using carry-based algorithm
Operator << overloads the output of HugeInt object to standard output stream.

b. What restrictions does the class have?
   i. It has a fixed maximum no of digits .
   ii. It only deals with integers represented as strings of digits .

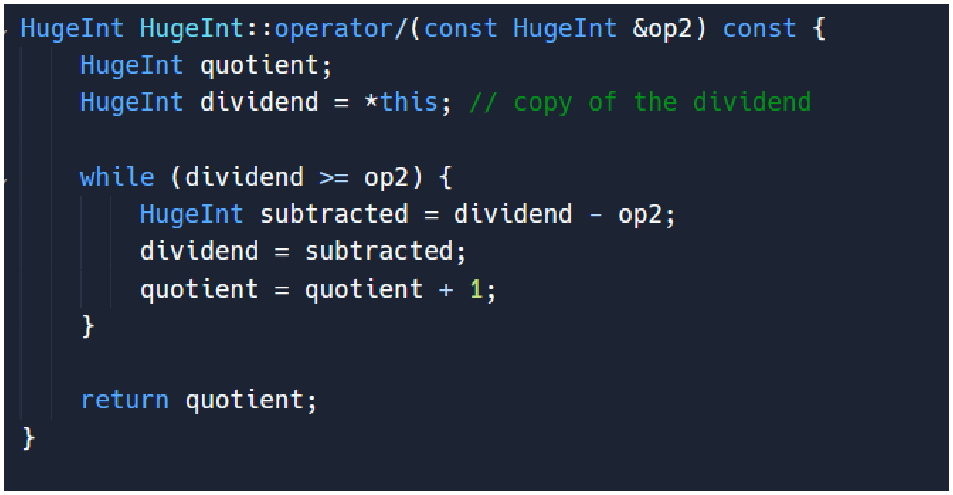
c. Overload the * multiplication operator.

➔



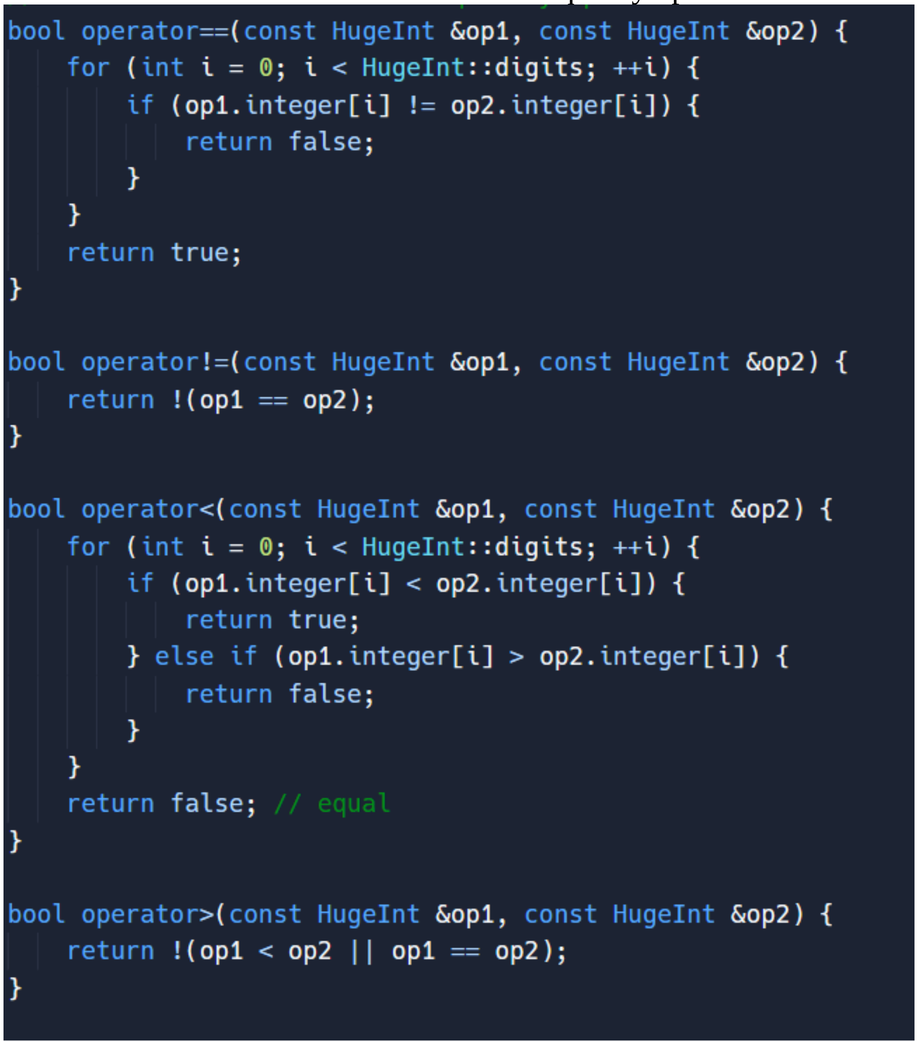
```
HugeInt HugeInt::operator*(const HugeInt &op2) const {
    HugeInt temp; // temporary result
    HugeInt result; // stores intermediate results
    int carry = 0;

    for (int i = digits - 1; i >= 0; i--) {
        temp = *this * op2.integer[i]; // multiply current digit of
op2 with this
        temp = temp + carry; // add carry from previous
multiplication
        result = result + temp; // add intermediate result to final
result
        carry = temp.integer[0] / 10; // calculate new carry
    }

    return result;
}
```

d. Overload the / division operator.

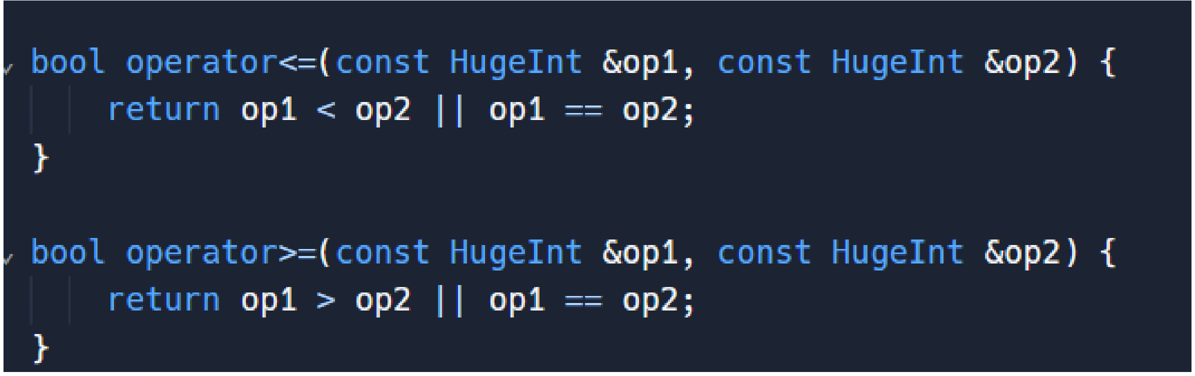


```
HugeInt HugeInt::operator/(const HugeInt &op2) const {
    HugeInt quotient;
    HugeInt dividend = *this; // copy of the dividend

    while (dividend >= op2) {
        HugeInt subtracted = dividend - op2;
        dividend = subtracted;
        quotient = quotient + 1;
    }

    return quotient;
}
```

e. Overload all the relational and equality operators.



```
bool operator==(const HugeInt &op1, const HugeInt &op2) {
    for (int i = 0; i < HugeInt::digits; ++i) {
        if (op1.integer[i] != op2.integer[i]) {
            return false;
        }
    }
    return true;
}

bool operator!=(const HugeInt &op1, const HugeInt &op2) {
    return !(op1 == op2);
}

bool operator<(const HugeInt &op1, const HugeInt &op2) {
    for (int i = 0; i < HugeInt::digits; ++i) {
        if (op1.integer[i] < op2.integer[i]) {
            return true;
        } else if (op1.integer[i] > op2.integer[i]) {
            return false;
        }
    }
    return false; // equal
}

bool operator>(const HugeInt &op1, const HugeInt &op2) {
    return !(op1 < op2 || op1 == op2);
}
```



```
bool operator<=(const HugeInt &op1, const HugeInt &op2) {
    return op1 < op2 || op1 == op2;
}

bool operator>=(const HugeInt &op1, const HugeInt &op2) {
    return op1 > op2 || op1 == op2;
}
```

[*Note:* We do not show an assignment operator or copy constructor for class *HugeInt*, because the assignment operator and copy constructor provided by the compiler are capable of copying the entire array data member properly.]

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

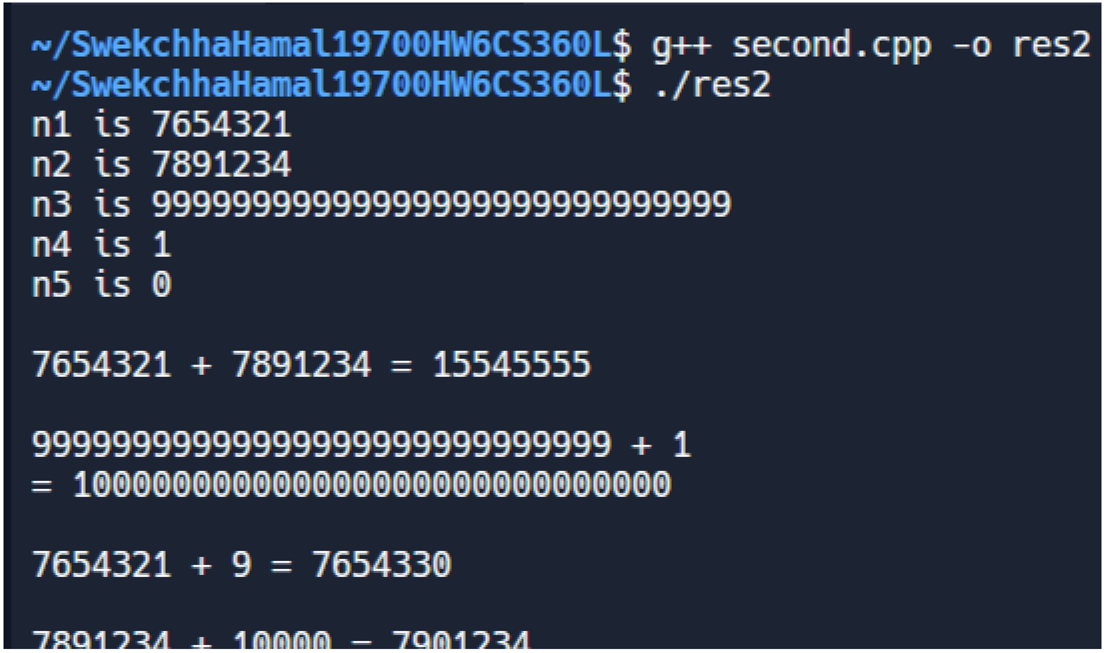
```
#include <array>
#include <iostream>
#include <string>

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;
 private:
   std::array< short, digits > integer;
}; // end class HugetInt

#endif
```

```
// 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+
```

```cpp
// 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+

// 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( "999999999999999999999999999999" );
  HugeInt n4( "1" );
  HugeInt n5;

  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;
} // end main
```



```
~/SwekchhaHamal19700HW6CS360L$ g++ second.cpp -o res2
~/SwekchhaHamal19700HW6CS360L$ ./res2
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
```