**Logo

Description automatically generated San Francisco Bay University**

**CS360L - Programming in C and C++ Lab**

**Lab Assignment #6**

**Due day: 06/29/2022**

**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
   1. 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).
   2. Overload the multiplication operator to enable multiplication of two complex numbers as in algebra.
   3. 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*

1. 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:
   1. Describe precisely how it operates.
   2. What restrictions does the class have?
   3. Overload the \* multiplication operator.
   4. Overload the / division operator.
   5. Overload all the relational and equality operators.

[*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+*

*// 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( "99999999999999999999999999999" );*

*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*