**Samir Khadka**
**CS360L - Programming in C and C++ Lab**
**Lab Assignment #6**

## Question 1:

main.cpp

main.cpp

Format

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

#include <iostream>

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
    bool operator==(const Complex&) const; // equality
    bool operator!=(const Complex&) const; // inequality

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

private:
    double real; // real part
    double imaginary; // imaginary part
}; // end class Complex

#endif

// Complex.cpp
// Complex class member-function definitions.
#include "Complex.h" // Complex class definition

// 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+
```

AI {} C++ Ln 114, Col 1 Spaces: 2 History

main.cpp

main.cpp

Format

```cpp
// 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 {
    return Complex(
        (real * operand2.real) - (imaginary * operand2.imaginary),
        (real * operand2.imaginary) + (imaginary * operand2.real)
    );
} // end function operator*

// 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 !(*this == operand2);
} // end function operator!=

// output operator
std::ostream& operator<<(std::ostream& os, const Complex& complex) {
    os << '(' << complex.real << ", " << complex.imaginary << "i)";
    return os;
} // end function operator<<

// input operator
std::istream& operator>>(std::istream& is, Complex& complex) {
    char dummy;
    is >> complex.real >> dummy >> complex.imaginary >> dummy;
    return is;
} // end function operator>>

// main.cpp
// Complex class test program.
#include <iostream>
#include "Complex.h"
```

main.cpp

main.cpp

Format

```cpp
using namespace std;

int main() {
    Complex x;
    Complex y(4.3, 8.2);
    Complex z(3.3, 1.1);

    cout << "Enter complex number x in the form (a, b): ";
    cin >> x;

    cout << "x: " << x << endl;
    cout << "y: " << y << endl;
    cout << "z: " << z << endl;

    Complex sum = y + z;
    cout << "x = y + z: " << sum << endl;

    Complex difference = y - z;
    cout << "x = y - z: " << difference << endl;

    Complex product = y * z;
    cout << "x = y * z: " << product << endl;

    cout << "Comparison: ";
    if (x == y)
        cout << "x is equal to y";
    else
        cout << "x is not equal to y";
    cout << endl;

    return 0;
}
```

Run Ask AI 14s on 09:42:26, 04/11

```
Enter complex number x in the form (a, b): (3,2)
x: (0, 0i)
y: (4.3, 8.2i)
z: (3.3, 1.1i)
x = y + z: (7.6, 9.3i)
x = y - z: (1, 7.1i)
x = y * z: (5.17, 31.79i)
Comparison: x is not equal to y
```

## Question 2:

Question 2.cpp

Question 2.cpp > ...

Format

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

#include <array>
#include <iostream>
#include <string>
#include <cmath>

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

    // arithmetic operators
    HugeInt operator+(const HugeInt&) const;
    HugeInt operator-(const HugeInt&) const;
    HugeInt operator*(const HugeInt&) const;
    HugeInt operator/(const HugeInt&) const;

    // relational and equality operators
    bool operator<(const HugeInt&) const;
    bool operator>(const HugeInt&) const;
    bool operator<=(const HugeInt&) const;
    bool operator>=(const HugeInt&) const;
    bool operator==(const HugeInt&) const;
    bool operator!=(const HugeInt&) const;

private:
    std::array<short, digits> integer;
};

// constructor; conversion/default constructor
HugeInt::HugeInt(long value) {
    // initialize array to zero
```

Question 2.cpp

Question 2.cpp > ...

Format

```
40      for (short& element : integer)
41          element = 0;
42      // place digits of argument into array
43      for (size_t j = digits - 1; value != 0 && j >= 0; j--) {
44          integer[j] = value % 10;
45          value /= 10;
46      }
47  }
48
49  // conversion constructor; converts a character string representing a large integer
    into a HugeInt object
50  HugeInt::HugeInt(const std::string& number) {
51      // initialize array to zero
52      for (short& element : integer)
53          element = 0;
54      // place digits of argument into array
55      size_t length = number.size();
56      for (size_t j = digits - length, k = 0; j < digits; ++j, ++k)
57          if (isdigit(number[k])) // ensure that character is a digit
58              integer[j] = number[k] - '0';
59  }
60
61  // addition operator; HugeInt + HugeInt
62  HugeInt HugeInt::operator+(const HugeInt& op2) const {
63      HugeInt temp; // temporary result
64      int carry = 0;
65      for (int i = digits - 1; i >= 0; i--) {
66          temp.integer[i] = integer[i] + op2.integer[i] + carry;
67          // determine whether to carry a 1
68          if (temp.integer[i] > 9) {
69              temp.integer[i] %= 10; // reduce to 0-9
70              carry = 1;
71          } else // no carry
72              carry = 0;
73      }
74      return temp; // return copy of temporary object
75  }
76
77  // subtraction operator; HugeInt - HugeInt
78  HugeInt HugeInt::operator-(const HugeInt& op2) const {
```

Question 2.cpp

Question 2.cpp > ...

Format

```
79      HugeInt result;
80      int borrow = 0;
81      for (int i = digits - 1; i >= 0; i--) {
82          int diff = integer[i] - op2.integer[i] - borrow;
83          if (diff < 0) {
84              diff += 10;
85              borrow = 1;
86          } else {
87              borrow = 0;
88          }
89          result.integer[i] = diff;
90      }
91      return result;
92  }
93
94  // multiplication operator; HugeInt * HugeInt
95  HugeInt HugeInt::operator*(const HugeInt& op2) const {
96      HugeInt result; // Initialize result
97      HugeInt temp;
98      int carry = 0;
99
100     for (int i = digits - 1; i >= 0; i--) {
101         for (int j = digits - 1; j >= 0; j--) {
102             int product = (integer[i] * op2.integer[j]) + carry;
103             temp.integer[j] = product % 10; // Store the result
104             carry = product / 10; // Carry
105         }
106         result = result + temp; // Add temp to result
107     }
108     return result;
109 }
110
111 // division operator; HugeInt / HugeInt
112 HugeInt HugeInt::operator/(const HugeInt& op2) const {
113     HugeInt quotient;
114     HugeInt temp(*this);
115     HugeInt remainder("0");
116     HugeInt one("1");
117     while (temp >= op2) {
118         temp = temp - op2;
```

Question 2.cpp

Question 2.cpp > ...

Format

```
119             quotient = quotient + one;
120         }
121         return quotient;
122     }
123
124     // less than operator
125     bool HugeInt::operator<(const HugeInt& op2) const {
126         for (int i = 0; i < digits; ++i) {
127             if (integer[i] < op2.integer[i])
128                 return true;
129             else if (integer[i] > op2.integer[i])
130                 return false;
131         }
132         return false;
133     }
134
135     // greater than operator
136     bool HugeInt::operator>(const HugeInt& op2) const {
137         return op2 < *this;
138     }
139
140     // less than or equal to operator
141     bool HugeInt::operator<=(const HugeInt& op2) const {
142         return !(op2 < *this);
143     }
144
145     // greater than or equal to operator
146     bool HugeInt::operator>=(const HugeInt& op2) const {
147         return !(*this < op2);
148     }
149
150     // equality operator
151     bool HugeInt::operator==(const HugeInt& op2) const {
152         for (int i = 0; i < digits; ++i) {
153             if (integer[i] != op2.integer[i])
154                 return false;
155         }
156         return true;
157     }
158
```

Question 2.cpp

Question 2.cpp > ...

Format

```cpp
// inequality operator
bool HugeInt::operator!=(const HugeInt& op2) const {
    return !(*this == op2);
}

// overloaded output operator
std::ostream& operator<<(std::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;
}

#endif

// main.cpp
// HugeInt test program.
#include <iostream>
#include "Hugeint.h"
using namespace std;

int main() {
    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";

```

Question 2.cpp

```
    cout << n3<< " + " << n4 << "\n= " << (n3 + n4) << "\n\n";
    n5 = n1 + 9;cout << n1 << " + " << 9 << " = " << n5 << "\n\n";

n5 = n2 + HugeInt("10000");
cout << n2 << " + " << "10000" << " = " << n5 << endl;

// Testing multiplication and division
HugeInt n6("12345678901234567890123456789O");
HugeInt n7("98765432109876543210987654321O");
HugeInt n8;

cout << "\nMultiplication:\n";
n8 = n6 * n7;
cout << n6 << " * " << n7 << " = " << n8 << endl;

cout << "\nDivision:\n";
n8 = n7 / n6;
cout << n7 << " / " << n6 << " = " << n8 << endl;

return 0;


}
```

Run

Ask AI

5s on 10:10:51, 04/11

```
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

Multiplication:
12345678901234567890123456789O * 98765432109876543210987654321O = 33333334833
3333334833333333458

Division:
98765432109876543210987654321O / 12345678901234567890123456789O = 8
```