Q.no.1)

complex.h

```
#ifndef COMPLEX_H
#define COMPLEX_H

#include <iostream>

class Complex {
public:
    explicit Complex(double real = 0.0, double imaginary = 0.0); // constructor
    Complex operator+(const Complex& other) const; // addition
    Complex operator-(const Complex& other) const; // subtraction
    Complex operator*(const Complex& other) const; // multiplication
    bool operator==(const Complex& other) const; // equality
    bool operator!=(const Complex& other) const; // inequality

    // Overloaded stream insertion and extraction operators
  friend std::ostream& operator<<(std::ostream& out, const Complex& c);
  friend std::istream& operator>>(std::istream& in, Complex& c);

private:
```

complex.cpp

```cpp
// complex.cpp
#include <iostream>
#include "complex.h"

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

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

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

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

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

main:

```cpp
#include <iostream>
#include "complex.h"

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

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

    // Subtraction
    x = y - z;
    std::cout << "\n";
    std::cout << "x = y - z: " << x << std::endl;
    std::cout << "= " << y << " - " << z << std::endl;

    // Multiplication
    Complex product = y * z;
    std::cout << "\n";
    std::cout << "Product of y and z: " << product <<
std::endl;

    // Comparison
    std::cout << "\n";
```

```
~/cs360$ g++ complex.cpp main.cpp -o complex_program
~/cs360$ ./complex_program
x = y + z: (7.6, 9.3i)
= (4.3, 8.2i) + (3.3, 1.1i)

x = y - z: (1, 7.1i)
= (4.3, 8.2i) - (3.3, 1.1i)

Product of y and z: (5.17, 31.79i)

y is not equal to z
```

Q.no.2)

a. Describe precisely how it operates:

- The program defines a class `HugeInt` in C++ that allows working with large integers that exceed the range of typical 32-bit integers.
- The class provides constructors to initialize `HugeInt` objects with either a long integer value or a string representing a large integer.
- It overloads operators such as + for addition with another `HugeInt` object, addition with an integer, or addition with a string.

- Additionally, the program overloads the output stream operator << to display the `HugeInt` object.

b. What restrictions does the class have:

- The `HugeInt` class has a restriction on the maximum number of digits it can handle, which is set to 30 digits (`static const int digits = 30`

HugeInt.h

HugeInt.h > ... Format

```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;
```

HugeInt.cpp

HugeInt.cpp > ... Format

```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 HugeInt default/conversion constructor

// conversion constructor that converts a character string
// representing a large integer into a HugeInt object
```

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;
    return 0;
}
```

Output:

```
~/cs360labr$  g++ -o program main.cpp HugeInt.cpp
~/cs360labr$ ./program
n1 is 7654321
n2 is 7891234
n3 is 999999999999999999999999999999
n4 is 1
n5 is 0

7654321 + 7891234 = 15545555

999999999999999999999999999999 + 1
= 1000000000000000000000000000000

7654321 + 9 = 7654330

7891234 + 10000 = 7901234
```