Question no.1

This program design is to calculate complex number. Complex values are denoted by a parenthesized pair of values separated by a comma representing the real and imaginary part of the variable. For example, (1, 2) indicates that the real part is 1 and the imaginary part is 2. A complex number can also be represented by the magnitude and angle format like this (1 > 45) indicating a complex value with a magnitude of 1 and an angle of 45 degrees.

You will need to implement the Complex class, and provide operations for the plus, minus, multiply, and divide calculations. You will NOT need an exponentiation operator for this assignment. The Complex class will need a constructor with no arguments (default constructor), one with two arguments with initial values of both the real and imaginary part, and a third constructor that builds a complex number from a const string&, such as Complex("123, 456"). You will likely need the length() and empty() methods that give the length of a string and a Boolean true value if the string is empty. You will also need a member function to calculate the magnitude of the complex value, the angle of the value, and the complex conjugate of the value. Finally, you will create a Print() method in your Complex class to print the value of the complex number.

- ⇨ Answer:
  Code:

Q.1.cpp > main

```cpp
#include <iostream>
#include <cmath>
#include <string>
#include <sstream>

class Complex {
private:
    double real;
    double imag;

public:
    // Default constructor
    Complex() : real(0), imag(0) {}

    // Constructor with real and imaginary parts
    Complex(double r, double i) : real(r), imag(i) {}

    // Constructor from string
    Complex(const std::string& str) {
        std::istringstream iss(str);
        char comma;
        iss >> real >> comma >> imag;
    }

    // Getter methods
    double getReal() const { return real; }
    double getImag() const { return imag; }

    // Calculate magnitude of complex number
    double magnitude() const {
```

Q.1.cpp > main

```cpp
    double magnitude() const {
        return std::sqrt(real * real + imag * imag);
    }

    // Calculate angle of complex number in radians
    double angle() const {
        return std::atan2(imag, real);
    }

    // Calculate complex conjugate
    Complex conjugate() const {
        return Complex(real, -imag);
    }

    // Addition operator
    Complex operator+(const Complex& other) const {
        return Complex(real + other.real, imag + other.imag);
    }

    // Subtraction operator
    Complex operator-(const Complex& other) const {
        return Complex(real - other.real, imag - other.imag);
    }

    // Multiplication operator
    Complex operator*(const Complex& other) const {
        return Complex(real * other.real - imag * other.imag,
                       real * other.imag + imag * other.real);
    }

```

⇨ Output:

```
~/cs360-hw1$ touch Q.1.cpp
~/cs360-hw1$ g++ Q.1.cpp -o Q.1_output
~/cs360-hw1$ ./ Q.1_output
bash: ./: Is a directory
~/cs360-hw1$ ./Q.1_output
c1: (1, 2)
c2: (3, 4)
Magnitude of c1: 2.23607
Angle of c1: 1.10715 radians
Conjugate of c1: (1, -2)
c1 + c2 = (4, 6)
c1 - c2 = (-2, -2)
c1 * c2 = (-5, 10)
c1 / c2 = (0.44, 0.08)
~/cs360-hw1$
```

Question no.2

Design a program to implement matrix operations, such as add, subtract and multiply (we won't do divide). In order to do this, we will create a class called Matrix that processes a two-dimensional matrix. This class contains a constructor that builds the matrix with data from a character string. To describe a matrix with a string, we use parenthesis to delineate the rows of the matrix. For example: (1,2,3),(4,5,6),(7,8,9) would represent the matrix:

$$: \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

The three types of matrix operations should be covered in the method(s). We will also use a Not A Matrix flag in our matrix class to indicate that the matrix is invalid. This would be set when the size of the matrices being added or multiplied are not compatible.

Specific Program Requirements:

a) You must define and implement a Matrix class, with a constructor with a string argument, to construct a matrix with initial contents. In this case, the size of the matrix is apparent from the input string.
b) Since your Matrix class allocates memory in the constructor, you MUST implement a destructor that frees the memory.
c) You must implement a IsNaM function that returns a Boolean true/false indicating whether the matrix is Not a Matrix.
d) The Matrix class must implement indexing operator (operator[]) to access individual elements in the matrix.
e) All matrix operations must be implemented as member functions.

⇨ Answer:

Code:

Q.2.cpp

```cpp
#include <iostream>
#include <vector>
#include <sstream>
#include <stdexcept>

class Matrix {
private:
    std::vector<std::vector<int>> data;
    bool isNaM;

public:
    // Constructor with string argument to build the matrix
    Matrix(const std::string& str) : isNaM(false) {
        std::istringstream iss(str);
        char delimiter;
        int num;
        std::vector<int> row;

        while (iss >> delimiter) {
            if (delimiter == '(') {
                while (iss >> num) {
                    row.push_back(num);
                    if (iss.peek() == ',')
                        iss.ignore();
                    else if (iss.peek() == ')') {
                        iss.ignore();
                        break;
                    }
                }
                data.push_back(row);
```

Q.2.cpp

```cpp
                data.push_back(row);
                row.clear();
            }
        }

        // Check if the matrix is not valid
        if (data.empty() || data[0].empty())
            isNaM = true;
        else {
            int cols = data[0].size();
            for (const auto& r : data) {
                if (r.size() != cols) {
                    isNaM = true;
                    break;
                }
            }
        }
    }

    // Constructor with number of rows and columns
    Matrix(size_t rows, size_t cols) : data(rows, std::vector<int>
(cols, 0)), isNaM(false) {}

    // Destructor to free memory
    ~Matrix() {}

    // Function to check if matrix is Not a Matrix
    bool IsNaM() const {
        return isNaM;
    }
```

Q.2.cpp

```cpp
    }

    // Indexing operator to access individual elements in the matrix
    int operator()(size_t row, size_t col) const {
        return data[row][col];
    }

    // Addition of two matrices
    Matrix add(const Matrix& other) const {
        if (data.size() != other.data.size() || data[0].size() != other.data[0].size()) {
            throw std::invalid_argument("Matrices are not compatible for addition");
        }

        Matrix result = *this;
        for (size_t i = 0; i < data.size(); ++i) {
            for (size_t j = 0; j < data[0].size(); ++j) {
                result.data[i][j] += other.data[i][j];
            }
        }
        return result;
    }

    // Subtraction of two matrices
    Matrix subtract(const Matrix& other) const {
        if (data.size() != other.data.size() || data[0].size() != other.data[0].size()) {
            throw std::invalid_argument("Matrices are not compatible for subtraction");
```

Q.2.cpp

```cpp

        Matrix result = *this;
        for (size_t i = 0; i < data.size(); ++i) {
            for (size_t j = 0; j < data[0].size(); ++j) {
                result.data[i][j] -= other.data[i][j];
            }
        }
        return result;
    }

    // Multiplication of two matrices
    Matrix multiply(const Matrix& other) const {
        if (data[0].size() != other.data.size()) {
            throw std::invalid_argument("Matrices are not compatible for multiplication");
        }

        Matrix result(data.size(), other.data[0].size());
        for (size_t i = 0; i < data.size(); ++i) {
            for (size_t j = 0; j < other.data[0].size(); ++j) {
                for (size_t k = 0; k < data[0].size(); ++k) {
                    result.data[i][j] += data[i][k] * other.data[k][j];
                }
            }
        }
        return result;
    }
};

int main() {
```

Q.2.cpp

```cpp
    std::string input1 = "(1,2,3),(4,5,6),(7,8,9)";
    std::string input2 = "(9,8,7),(6,5,4),(3,2,1)";

    Matrix matrix1(input1);
    Matrix matrix2(input2);

    if (matrix1.IsNaM() || matrix2.IsNaM()) {
        std::cout << "One or both of the matrices are not valid." << std::endl;
        return 1;
    }

    try {
        Matrix result_add = matrix1.add(matrix2);
        std::cout << "Addition result:" << std::endl;
        for (size_t i = 0; i < 3; ++i) {
            for (size_t j = 0; j < 3; ++j) {
                std::cout << result_add(i, j) << " ";
            }
            std::cout << std::endl;
        }
    } catch (const std::invalid_argument& e) {
        std::cout << e.what() << std::endl;
    }

    try {
        Matrix result_sub = matrix1.subtract(matrix2);
        std::cout << "Subtraction result:" << std::endl;
        for (size_t i = 0; i < 3; ++i) {
            for (size_t j = 0; j < 3; ++j) {
```

Q.2.cpp

```cpp
        std::cout << "Subtraction result:" << std::endl;
        for (size_t i = 0; i < 3; ++i) {
            for (size_t j = 0; j < 3; ++j) {
                std::cout << result_sub(i, j) << " ";
            }
            std::cout << std::endl;
        }
    } catch (const std::invalid_argument& e) {
        std::cout << e.what() << std::endl;
    }

    try {
        Matrix result_mul = matrix1.multiply(matrix2);
        std::cout << "Multiplication result:" << std::endl;
        for (size_t i = 0; i < 3; ++i) {
            for (size_t j = 0; j < 3; ++j) {
                std::cout << result_mul(i, j) << " ";
            }
            std::cout << std::endl;
        }
    } catch (const std::invalid_argument& e) {
        std::cout << e.what() << std::endl;
    }

    return 0;
}

```

Output:

```
~/cs360-hw1$ touch Q.2.cpp
~/cs360-hw1$ g++ Q.2.cpp -o Q.2_output
~/cs360-hw1$ ./Q.2_output
Addition result:
10 10 10
10 10 10
10 10 10
Subtraction result:
-8 -6 -4
-2 0 2
4 6 8
Multiplication result:
30 24 18
84 69 54
138 114 90
~/cs360-hw1$
```