San Francisco Bay University

CS360 - Programming in C and C++
Homework Assignment #1

Due day: 2/14/2024

Instruction:

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

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.

➔ Complex coordinate is the point where we know x-axis , real number but don't have specific y-axis . Therefore, it is difficult to have a specific position , arbitrary. But we can calculate the value or magnitude of complex number by the formula:

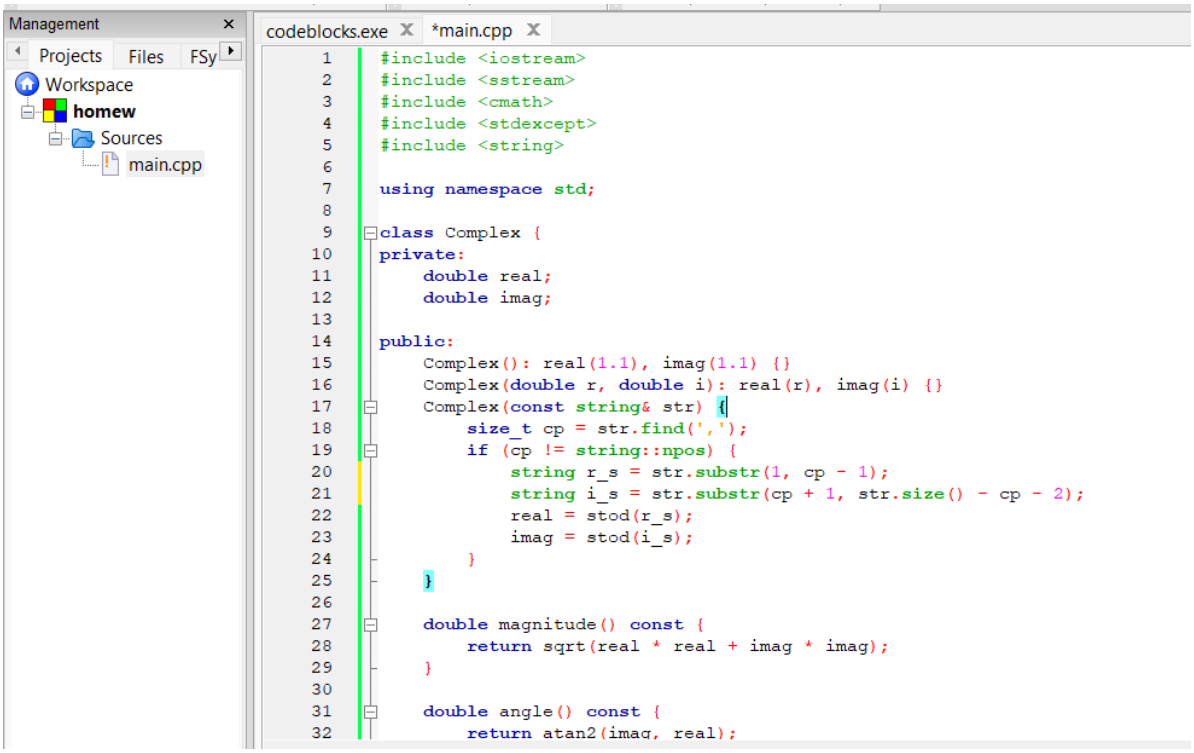
$$|z| = \sqrt{a^2 + b^2}$$

Where a = real part of complex num
b = imaginary part of complex num
Likewise, to identify the direction or which quadrant the complex number is facing we need the angle (counterclockwise) which can be calculated by:

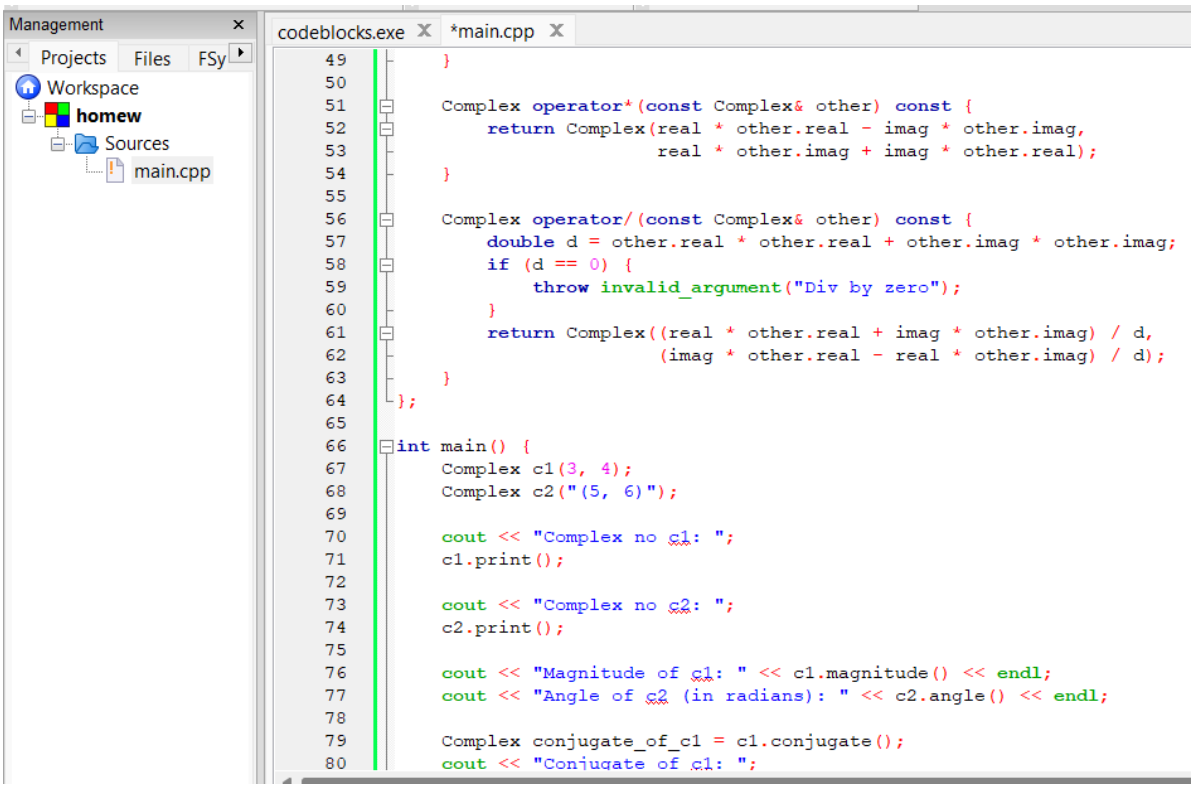
A = atan2(b,a)

Conjugate of complex number is obtained by changing the sign of imaginary part of complex number.

Operations like addition , subtraction , multiplication and division have the following formulas:
Add: (a+bi)+(c+di)=(a+c)+(b+d)i
Subtract: (a+bi)−(c+di)=(a−c)+(b−d)i
Mulitply: (a+bi)×(c+di)=(ac−bd)+(ad+bc)i
Divide: $\frac{a+bi}{c+di} = \frac{(ac+bd)+(bc-ad)i}{c^2+d^2}$

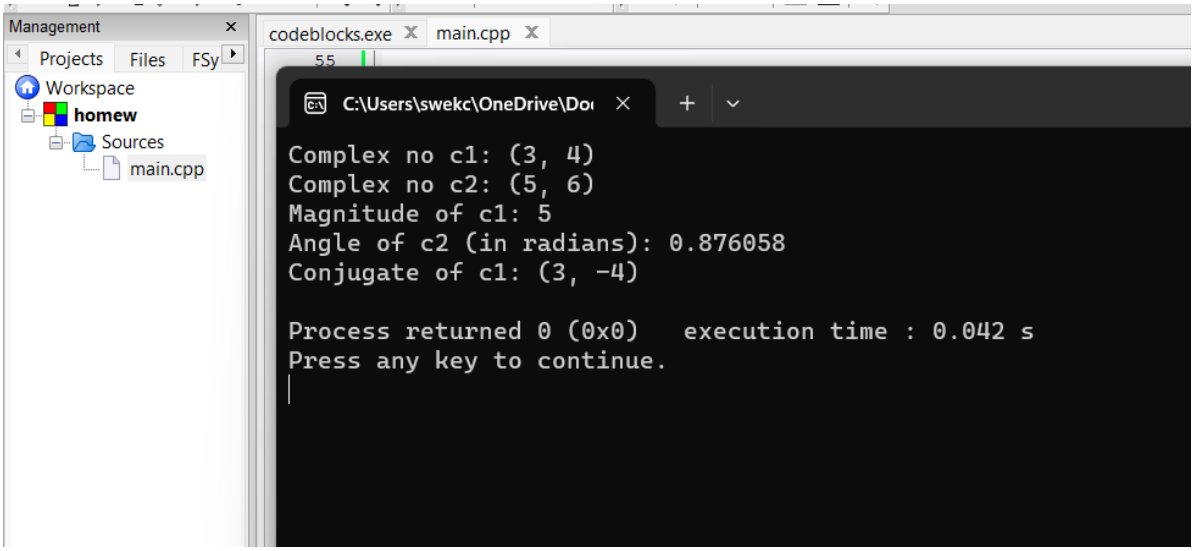



```
25      }
26
27      double magnitude() const {
28          return sqrt(real * real + imag * imag);
29      }
30
31      double angle() const {
32          return atan2(imag, real);
33      }
34
35      Complex conjugate() const {
36          return Complex(real, -imag);
37      }
38
39      void print() const {
40          cout << "(" << real << ", " << imag << ")" << endl;
41      }
42
43      Complex operator+(const Complex& other) const {
44          return Complex(real + other.real, imag + other.imag);
45      }
46
47      Complex operator-(const Complex& other) const {
48          return Complex(real - other.real, imag - other.imag);
49      }
50
51      Complex operator*(const Complex& other) const {
52          return Complex(real * other.real - imag * other.imag,
53                         real * other.imag + imag * other.real);
54      }
55
56      Complex operator/(const Complex& other) const {
```




```
49      }
50
51      Complex operator*(const Complex& other) const {
52          return Complex(real * other.real - imag * other.imag,
53                         real * other.imag + imag * other.real);
54      }
55
56      Complex operator/(const Complex& other) const {
57          double d = other.real * other.real + other.imag * other.imag;
58          if (d == 0) {
59              throw invalid_argument("Div by zero");
60          }
61          return Complex((real * other.real + imag * other.imag) / d,
62                         (imag * other.real - real * other.imag) / d);
63      }
64  };
65
66  int main() {
67      Complex c1(3, 4);
68      Complex c2("(5, 6)");
69
70      cout << "Complex no c1: ";
71      c1.print();
72
73      cout << "Complex no c2: ";
74      c2.print();
75
76      cout << "Magnitude of c1: " << c1.magnitude() << endl;
77      cout << "Angle of c2 (in radians): " << c2.angle() << endl;
78
79      Complex conjugate_of_c1 = c1.conjugate();
80      cout << "Conjugate of c1: ";
```

```cpp
int main() {
    Complex c1(3, 4);
    Complex c2("(5, 6)");

    cout << "Complex no c1: ";
    c1.print();

    cout << "Complex no c2: ";
    c2.print();

    cout << "Magnitude of c1: " << c1.magnitude() << endl;
    cout << "Angle of c2 (in radians): " << c2.angle() << endl;

    Complex conjugate_of_c1 = c1.conjugate();
    cout << "Conjugate of c1: ";
    conjugate_of_c1.print();

    return 0;
}
```



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.

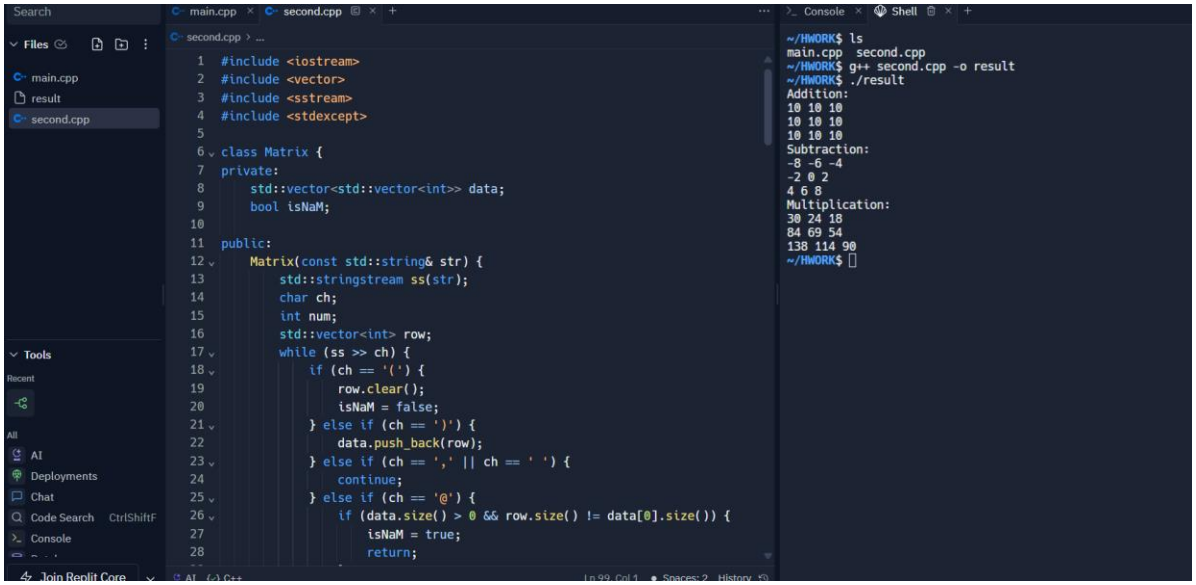
b. Since your Matrix class allocates memory in the constructor, you MUST implement a destructor that frees the memory.

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

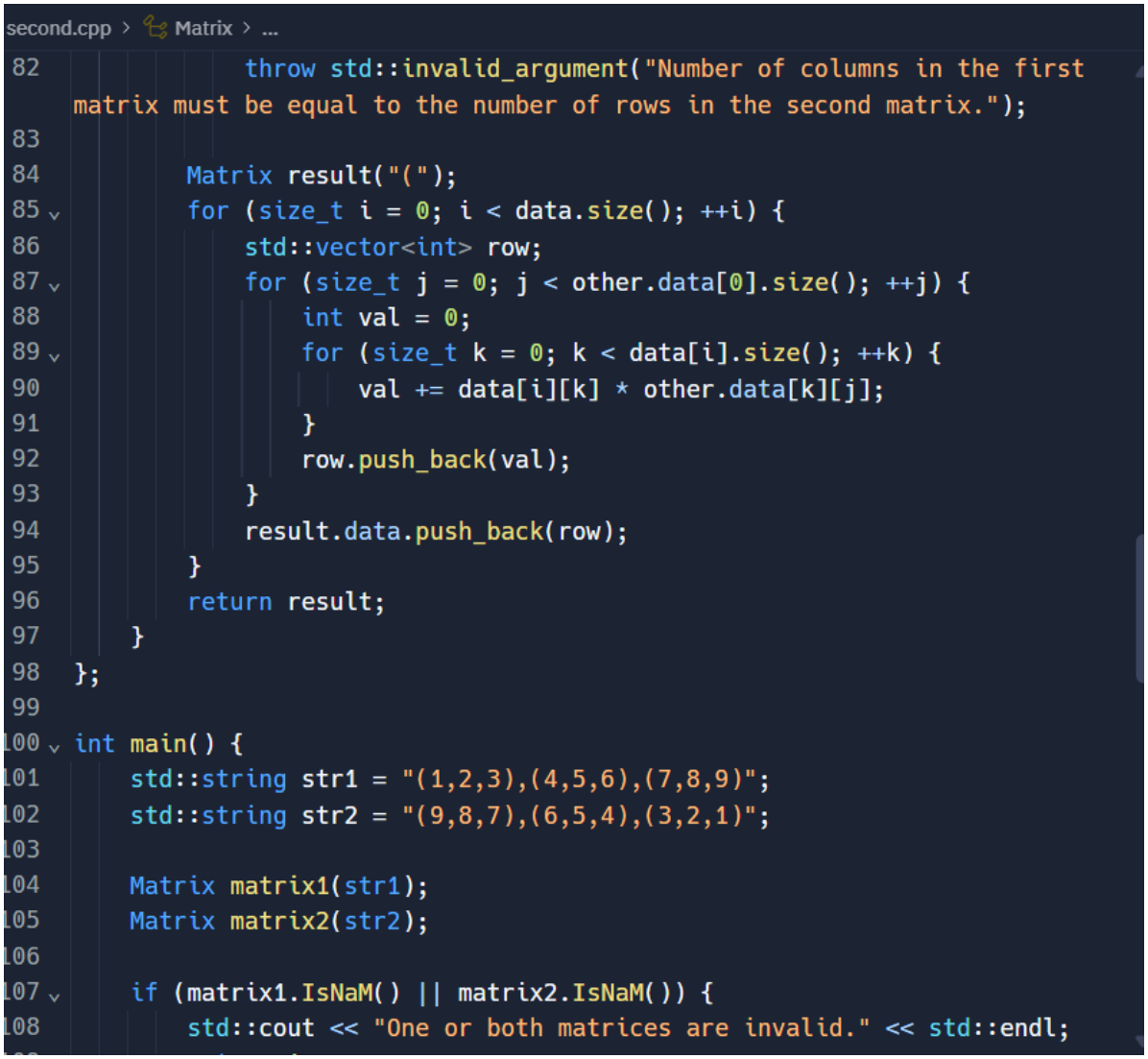
➔ In order to perform matrix calculations, the required condition are as follows:
Add/Subtract : same no of rows and columns
Multiplication: no of cols in first matrix = no of rows of second matrix.



second.cpp › ...

```
30              } else if (isdigit(ch)) {
31                  ss.putback(ch);
32                  ss >> num;
33                  row.push_back(num);
34              }
35          }
36      }
37
38      ~Matrix() {
39
40      }
41
42      bool IsNaM() const {
43          return isNaM;
44      }
45
46      std::vector<int>& operator[](int index) {
47          return data[index];
48      }
49
50      Matrix add(const Matrix& other) const {
51          if (data.size() != other.data.size() || data[0].size() !=
    other.data[0].size())
52              throw std::invalid_argument("Matrices must have the same
    dimensions .");
53
54          Matrix result("(");
55          for (size_t i = 0; i < data.size(); ++i) {
```


second.cpp › Matrix › ...
56 std::vector<int> row;
57 for (size_t j = 0; j < data[i].size(); ++j) {
58 row.push_back(data[i][j] + other.data[i][j]);
59 }
60 result.data.push_back(row);
61 }
62 return result;
63 }
64
65 Matrix subtract(const Matrix& other) const {
66 if (data.size() != other.data.size() || data[0].size() != other.data[0].size())
67 throw std::invalid_argument("Matrices must have the same dimensions.");
68
69 Matrix result("(");
70 for (size_t i = 0; i < data.size(); ++i) {
71 std::vector<int> row;
72 for (size_t j = 0; j < data[i].size(); ++j) {
73 row.push_back(data[i][j] - other.data[i][j]);
74 }
75 result.data.push_back(row);
76 }
77 return result;
78 }
79
80 Matrix multiply(const Matrix& other) const {
81 if (data[0].size() != other.data.size())


second.cpp > Matrix > ...
82 throw std::invalid_argument("Number of columns in the first matrix must be equal to the number of rows in the second matrix.");
83
84 Matrix result("(");
85 for (size_t i = 0; i < data.size(); ++i) {
86 std::vector<int> row;
87 for (size_t j = 0; j < other.data[0].size(); ++j) {
88 int val = 0;
89 for (size_t k = 0; k < data[i].size(); ++k) {
90 val += data[i][k] * other.data[k][j];
91 }
92 row.push_back(val);
93 }
94 result.data.push_back(row);
95 }
96 return result;
97 }
98 };
99
100 int main() {
101 std::string str1 = "(1,2,3),(4,5,6),(7,8,9)";
102 std::string str2 = "(9,8,7),(6,5,4),(3,2,1)";
103
104 Matrix matrix1(str1);
105 Matrix matrix2(str2);
106
107 if (matrix1.IsNaM() || matrix2.IsNaM()) {
108 std::cout << "One or both matrices are invalid." << std::endl;

```cpp
try {
    Matrix result_add = matrix1.add(matrix2);
    Matrix result_subtract = matrix1.subtract(matrix2);
    Matrix result_multiply = matrix1.multiply(matrix2);

    std::cout << "Addition:" << std::endl;
    for (size_t i = 0; i < result_add[0].size(); ++i) {
        for (size_t j = 0; j < result_add[0].size(); ++j) {
            std::cout << result_add[i][j] << " ";
        }
        std::cout << std::endl;
    }

    std::cout << "Subtraction:" << std::endl;
    for (size_t i = 0; i < result_subtract[0].size(); ++i) {
        for (size_t j = 0; j < result_subtract[0].size(); ++j) {
            std::cout << result_subtract[i][j] << " ";
        }
        std::cout << std::endl;
    }

    std::cout << "Multiplication:" << std::endl;
    for (size_t i = 0; i < result_multiply[0].size(); ++i) {
        for (size_t j = 0; j < result_multiply[0].size(); ++j) {
            std::cout << result_multiply[i][j] << " ";
        }
        std::cout << std::endl;
    }
```

```
        }
    } catch(const std::invalid_argument& e) {
        std::cerr << e.what() << std::endl;
    }

    return 0;
}
```

```
Console  Shell
~/HWORK$ ls
main.cpp  second.cpp
~/HWORK$ g++ second.cpp -o result
~/HWORK$ ./result
Addition:
10 10 10
10 10 10
10 10 10
Subtraction:
-8 -6 -4
-2 0 2
4 6 8
Multiplication:
30 24 18
84 69 54
138 114 90
~/HWORK$
```