**Question no. 1**

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

class Complex {

public:

// Default constructor (no arguments)

Complex() : real(0), imag(0) {}

// Constructor with two arguments (real and imaginary parts)

Complex(double r, double i) : real(r), imag(i) {}

// Constructor with one string argument (e.g., "1,2")

Complex(const string& str) {

size\_t comma = str.find(',');

real = stod(str.substr(0, comma));

imag = stod(str.substr(comma + 1));

}

// Addition

Complex operator+(const Complex& other) const {

return Complex(real + other.real, imag + other.imag);

}

// Subtraction

Complex operator-(const Complex& other) const {

return Complex(real - other.real, imag - other.imag);

}

// Multiplication

Complex operator\*(const Complex& other) const {

return Complex(real \* other.real - imag \* other.imag,

real \* other.imag + imag \* other.real);

}

// Division

Complex operator/(const Complex& other) const {

double denominator = other.real \* other.real + other.imag \* other.imag;

if (denominator == 0) {

throw runtime\_error("Division by zero");

}

return Complex((real \* other.real + imag \* other.imag) / denominator,

(imag \* other.real - real \* other.imag) / denominator);

}

// Magnitude

double magnitude() const {

return sqrt(real \* real + imag \* imag);

}

// Angle in degrees

double angle() const {

return atan2(imag, real) \* 180 / M\_PI;

}

// Conjugate

Complex conjugate() const {

return Complex(real, -imag);

}

// Print

void print() const {

cout << "(" << real << ", " << imag << ")";

}

private:

double real;

double imag;

};

int main() {

Complex c1(1, 2);

Complex c2(3, 4);

cout << "c1: ";

c1.print();

cout << endl;

cout << "c2: ";

c2.print();

cout << endl;

cout << "c1 + c2: ";

(c1 + c2).print();

cout << endl;

cout << "c1 - c2: ";

(c1 - c2).print();

cout << endl;

cout << "c1 \* c2: ";

(c1 \* c2).print();

cout << endl;

cout << "c1 / c2: ";

try {

(c1 / c2).print();

} catch (const runtime\_error& e) {

cout << e.what() << endl;

}

cout << "Magnitude of c1: " << c1.magnitude() << endl;

cout << "Angle of c1 (degrees): " << c1.angle() << endl;

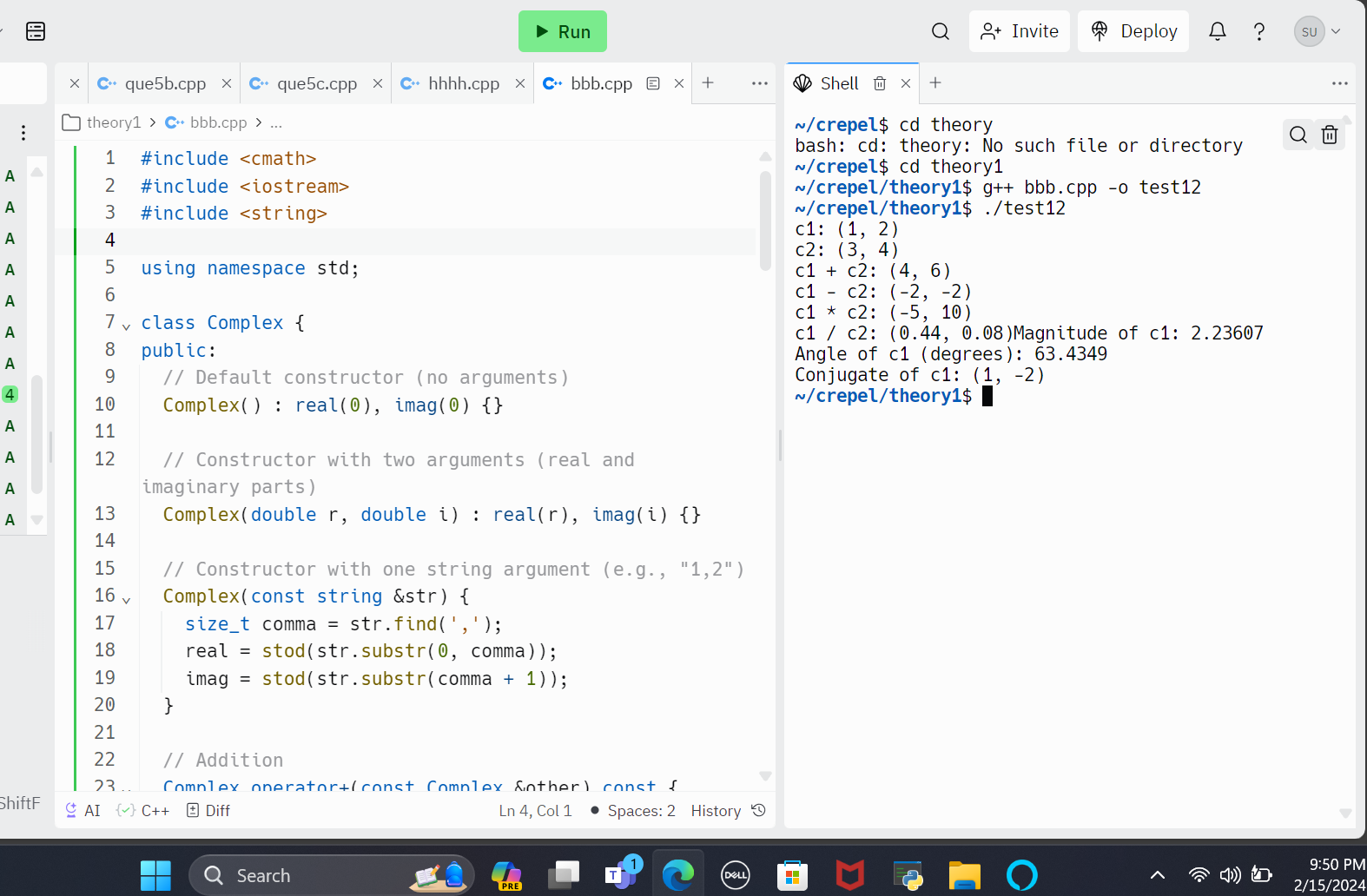
cout << "Conjugate of c1: ";

c1.conjugate().print();

cout << endl;

return 0;

}

→ 

**Question no. 2**

#include <iostream>

#include <sstream>

#include <vector>

class Matrix {

private:

std::vector<std::vector<int>> data;

bool isNotAMatrix;

public:

// Constructor

Matrix(const std::string &matrixString) {

isNotAMatrix = false;

std::stringstream ss(matrixString);

char ch;

int num;

std::vector<int> row;

while (ss >> ch) {

if (ch == '(') {

row.clear();

} else if (ch == ')') {

if (!row.empty()) {

data.push\_back(row);

}

} else if (ch == ',' || ch == ' ') {

continue;

} else if (isdigit(ch)) {

ss.putback(ch);

ss >> num;

row.push\_back(num);

} else {

isNotAMatrix = true;

break;

}

}

}

// Destructor

~Matrix() {}

// Function to check if the matrix is Not a Matrix

bool IsNaM() const { return isNotAMatrix; }

// Indexing operator to access individual elements in the matrix

int operator()(int i, int j) const { return data[i][j]; }

// Function to get number of rows in the matrix

int rows() const { return data.size(); }

// Function to get number of columns in the matrix

int columns() const {

if (!data.empty()) {

return data[0].size();

} else {

return 0;

}

}

// Function to add two matrices

Matrix add(const Matrix &other) const {

if (rows() != other.rows() || columns() != other.columns()) {

return Matrix("Not A Matrix");

}

Matrix result("(");

for (int i = 0; i < rows(); ++i) {

std::vector<int> row;

for (int j = 0; j < columns(); ++j) {

row.push\_back(data[i][j] + other.data[i][j]);

}

result.data.push\_back(row);

}

return result;

}

// Function to subtract two matrices

Matrix subtract(const Matrix &other) const {

if (rows() != other.rows() || columns() != other.columns()) {

return Matrix("Not A Matrix");

}

Matrix result("(");

for (int i = 0; i < rows(); ++i) {

std::vector<int> row;

for (int j = 0; j < columns(); ++j) {

row.push\_back(data[i][j] - other.data[i][j]);

}

result.data.push\_back(row);

}

return result;

}

// Function to multiply two matrices

Matrix multiply(const Matrix &other) const {

if (columns() != other.rows()) {

return Matrix("Not A Matrix");

}

Matrix result("(");

for (int i = 0; i < rows(); ++i) {

std::vector<int> row;

for (int j = 0; j < other.columns(); ++j) {

int sum = 0;

for (int k = 0; k < columns(); ++k) {

sum += data[i][k] \* other.data[k][j];

}

row.push\_back(sum);

}

result.data.push\_back(row);

}

return result;

}

};

int main() {

Matrix matrix1("(1,2,3),(4,5,6),(7,8,9)");

Matrix matrix2("(9,8,7),(6,5,4),(3,2,1)");

if (matrix1.IsNaM()) {

std::cout << "Matrix 1 is Not A Matrix.\n";

return 1;

}

if (matrix2.IsNaM()) {

std::cout << "Matrix 2 is Not A Matrix.\n";

return 1;

}

Matrix result = matrix1.multiply(matrix2);

if (result.IsNaM()) {

std::cout << "Matrices cannot be multiplied.\n";

return 1;

}

std::cout << "Result of multiplication:\n";

for (int i = 0; i < result.rows(); ++i) {

for (int j = 0; j < result.columns(); ++j) {

std::cout << result(i, j) << " ";

}

std::cout << "\n";

}

Matrix result\_add = matrix1.add(matrix2);

if (result\_add.IsNaM()) {

std::cout << "Matrices cannot be added.\n";

return 1;

}

Matrix result\_subtract = matrix1.subtract(matrix2);

if (result\_subtract.IsNaM()) {

std::cout << "Matrices cannot be subtracted.\n";

return 1;

}

std::cout << "Result of addition:\n";

for (int i = 0; i < result\_add.rows(); ++i) {

for (int j = 0; j < result\_add.columns(); ++j) {

std::cout << result\_add(i, j) << " ";

}

std::cout << "\n";

}

std::cout << "Result of subtraction:\n";

for (int i = 0; i < result\_subtract.rows(); ++i) {

for (int j = 0; j < result\_subtract.columns(); ++j) {

std::cout << result\_subtract(i, j) << " ";

}

std::cout << "\n";

}

return 0;

}

