Khoi Duong

Prof. Yang

CS360

5/31/2022

ASSIGNMENT #1

1.

Source code:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>
#include <bits/stdc++.h>

using namespace std;

class Complex{
    private:
float real, imag;
float r_polar, theta;
string r, i;

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

//Constructor with 2 arguments
    Complex(float r, float i){
        real = r;
        imag = i;
    }
```

```
//Constructor that build complex number from string
    Complex(string str){
        vector <int> v;
        stringstream ss(str);
        for (int i; ss >> i;) {
            v.push_back(i);
            if (ss.peek() == ',')
                ss.ignore();
        }
        real = v[0];
        imag = v[1];
    }

//Length method
    void length(void){
        r = std::to_string(real);
        i = std::to_string(imag);
           cout << "The length of string is: " << r.length() << "(real), " <<
i.length() << "(imaginary).";
    }

//Empty method
    void empty(void){
        r = std::to_string(real);
        i = std::to_string(imag);
        if (r.empty()) {cout << "Real part is an empty string.";}
        if (i.empty()) {cout << "Imaginary part is an empty string.";}
    }

//Calculate the conjugate of the complex number
    Complex com_conjugate(void){
        Complex conjugate;
        conjugate.real = real;
        conjugate.imag = -1 * imag;
        return conjugate;
    }

//Add function
    Complex plus(Complex c1){
        Complex ans;
        ans.real = real + c1.real;
        ans.imag = imag + c1.imag;
        return ans;
    }
```

```
//Subtract function
    Complex minus(Complex c1){
        Complex ans;
        ans.real = real - c1.real;
        ans.imag = imag - c1.imag;
        return ans;
    }

//Multiply function
    Complex multiply(Complex c1){
        Complex ans;
        ans.real = (real * c1.real) + (-1 * (imag * c1.imag));
        ans.imag = (imag * c1.real) + (real * c1.imag);
        return ans;
    }

//Divide function
    Complex divide(Complex c1){
        Complex ans;
        //Call conjugate function to find conjugate of c1
        Complex temp = c1.com_conjugate();
        //Calculate numerator and denominator
        Complex num = (*this).multiply(temp);
        Complex den = c1.multiply(temp);
        //Divide the numerator by denominator
        ans.real = num.real / den.real;
        ans.imag = num.imag / den.real;
        return ans;
    }

//Change the complex number to polar form with magnitude and angle
    int angle(void){
        r_polar = sqrt(pow(real,2) + pow(imag,2));
        theta = atan2(imag,real);
              cout << "(" << std::setprecision(3) << r_polar << " > " <<
std::setprecision(3) << theta << ")" << endl;
        return 0;
    }

//Print method
    void print(void){
    // if imaginary part is positive then
    //  it display real + i img complex number
    if (imag >= 0) {
```

```
        cout << std::setprecision(3) << real << "+i" << std::setprecision(3) <<
imag << endl;
    }

    // if imaginary part is negative then
    //  it display real - i img complex number
    else {
        cout << std::setprecision(3) << real << "-i" << std::setprecision(3) <<
(-1) * imag << endl;
    }
    }
};

int main(){
    Complex c1(6, 3);
    Complex c2("7, -5");
    cout << "Complex c1 polar form: ";
    c1.angle();
    cout << "Complex c2 polar form: ";
    c2.angle();
    Complex c3 = c1.plus(c2);
    cout << "Addition: ";
    c3.print();
    cout << "Addition (in polar form): ";
    c3.angle();
    Complex c4 = c1.minus(c2);
    cout << "Minus: ";
    c4.print();
    cout << "Minus (in polar form): ";
    c4.angle();
    Complex c5 = c1.multiply(c2);
    cout << "Multiply: ";
    c5.print();
    cout << "Multiply (in polar form): ";
    c5.angle();
    Complex c6 = c1.divide(c2);
    cout << "Divide: ";
    c6.print();
    cout << "Divide (in polar form): ";
    c6.angle();

    return 0;
}
```

Run code and result:

First complex number: 6 + 3i

Second complex number: 7 - 5i

```
PS D:\VS CODE\C C++\CS360\HW#1> cd "d:\VS CODE\
Complex c1 polar form: (6.71 > 0.464)
Complex c2 polar form: (8.6 > -0.62)
Addition: 13-i2
Addition (in polar form): (13.2 > -0.153)
Minus: -1+i8
Minus (in polar form): (8.06 > 1.7)
Multiply: 57-i9
Multiply (in polar form): (57.7 > -0.157)
Divide: 0.365+i0.689
Divide (in polar form): (0.78 > 1.08)
PS D:\VS CODE\C C++\CS360\HW#1>
```

2.

Source code:

```
#include <iostream>
#include <string>
#include <vector>
#include <bits/stdc++.h>
#include <algorithm>

using namespace std;

class Matrix{
    private:
    int k = 0, row, column, count = 0;
    std::vector<int> digits;
    int s[20][20];
    char *input;

    public:
```

```
Matrix(const string str){
    //Allocate string str to input
    input = new char[str.length()+1];
    assert(input != 0);
    const char* input = str.c_str();

    //Count row by counting the number of ")" in the string
    row = std::count(str.begin(), str.end(), ')');
    for (int i = 0; i < str.size(); i++) {
        //Check for digit numbers in the string
        if (isdigit(str[i])) {
            //Convert character type to int type
            int a = str[i] - '0';
            digits.push_back(a);
            count++;
            }
    }

    //Calculate the column to form a matrix
    column = count / row;
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < column; j++) {
            s[i][j] = digits[k];
            k++;
        }
    }
}

//Destructor free the memory
~Matrix() {delete[] input;}

int IsNaM(Matrix mat2, int mat_case){
    if (mat_case == 1){
        if (((*this).column == mat2.column) && ((*this).row == mat2.row)){
            return true;
        }
        else {return false;}
    }

    if (mat_case == 2){
        if ((*this).column == mat2.row) {return true;}
        else {return false;}
    }
    return 0;
}
```

```
void add(Matrix mat2){
    if ((*this).IsNaM(mat2,1) == true) {
        for (int i = 0; i < mat2.row; i++) {
            for (int j = 0; j < mat2.column; j++) {
                cout << s[i][j] + mat2.s[i][j] << " ";
            }
            cout << endl;
        }
    }
    else {
        cout << "Invalid matrix size for operation 'add'! " << endl;
    }
}

void subtract(Matrix mat2){
    if ((*this).IsNaM(mat2,1) == true) {
        for (int i = 0; i < mat2.row; i++) {
            for (int j = 0; j < mat2.column; j++) {
                cout << s[i][j] - mat2.s[i][j] << " ";
            }
            cout << endl;
        }
    }
    else {
        cout << "Invalid matrix size for operation 'subtract'! " << endl;
    }
}

void multiply(Matrix mat2){
    if ((*this).IsNaM(mat2,2) == true) {
        int i, j, k;
        int mult[20][20];

        // Initializing elements of matrix mult to 0.
        for(i = 0; i < (*this).row; ++i)
        {
            for(j = 0; j < mat2.column; ++j)
            {
                mult[i][j] = 0;
            }
        }

        // Multiplying 2 matrices and storing in array mult.
        for(i = 0; i < (*this).row; ++i)
```

```
            {
                for(j = 0; j < mat2.column; ++j)
                {
                    for(k=0; k<(*this).column; ++k)
                    {
                        mult[i][j] += s[i][k] * mat2.s[k][j];
                    }
                }
            }

            //Print out the mult matrices
            for (int i = 0; i < (*this).row; i++) {
                for (int j = 0; j < mat2.column; j++) {
                    cout << mult[i][j] << " ";
                    }
                cout << endl;
            }
        }
        else {
            cout << "Invalid matrix size for operation 'subtract'! " << endl;
        }
    }

    void print(){
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < column; j++) {
                cout << s[i][j] << " ";
                }
            cout << endl;
            }
    }
};

int main(){
    Matrix mat1("(1,2,3),(4,5,6),(7,8,9)");
    Matrix mat2("(9,8,7),(6,5,4),(3,2,1)");
    Matrix mat3("(4,9),(5,7),(6,3)");
    Matrix mat4("(2,5,8),(3,7,1)");
    cout << "Print matrix mat1: " << endl;
    mat1.print();
    cout << "Print matrix mat2: " << endl;
    mat2.print();
    cout << "Print matrix mat3: " << endl;
    mat3.print();
    cout << "Print matrix mat4: " << endl;
```

```
    mat4.print();
    cout << "mat1 + mat2: " << endl;
    mat1.add(mat2);
    cout << "mat3 + mat4: " << endl;
    mat3.add(mat4);
    cout << "mat1 - mat2: " << endl;
    mat1.subtract(mat2);
    cout << "mat1 * mat2: " << endl;
    mat1.multiply(mat2);
    cout << "mat3 * mat4: " << endl;
    mat3.multiply(mat4);

    return 0;
}
```

Run program & result:

```
PS D:\VS CODE\C C++\CS360\HW#1> cd "d:\VS
Print matrix mat1:
1 2 3
4 5 6
7 8 9
Print matrix mat2:
9 8 7
6 5 4
3 2 1
Print matrix mat3:
4 9
5 7
6 3
Print matrix mat4:
2 5 8
3 7 1
mat1 + mat2:
10 10 10
10 10 10
10 10 10
mat3 + mat4:
Invalid matrix size for operation 'add'!
mat1 - mat2:
-8 -6 -4
-2 0 2
4 6 8
mat1 * mat2:
30 24 18
84 69 54
138 114 90
mat3 * mat4:
35 83 41
31 74 47
21 51 51
PS D:\VS CODE\C C++\CS360\HW#1>
```