**testComplex.c**

#include <math.h>

#include <stdio.h>

// 複素数構造体を使う時には次の行を活かす

#include "complex.h"

#include "testCommon.h"

void testMakeComp() {

complex ans;

testStart("makeComp");

// 1+2jを作成

ans = makeComp(1.0, 2.0);

assertEqualsDouble(ans.real, 1.0);

assertEqualsDouble(ans.image, 2.0);

// 3.4+5.6jを作成

ans = makeComp(3.4, 5.6);

assertEqualsDouble(ans.real, 3.4);

assertEqualsDouble(ans.image, 5.6);

}

void testSubComp() {

complex in1, in2, ans;

testStart("subComp");

in1 = makeComp(1.0, 2.0);

in2 = makeComp(0.2, 0.4);

ans = subComp(in1, in2);

assertEqualsComplex(ans, makeComp(0.8, 1.6));

in1 = makeComp(sqrt(3), sqrt(3));

in2 = makeComp(0.2, 0.7);

ans = subComp(in1, in2);

assertEqualsComplex(ans, makeComp(sqrt(3) - 0.2, sqrt(3) - 0.7));

in1 = makeComp(-1.0, -2.0);

in2 = makeComp(3.2, 0.4);

ans = subComp(in1, in2);

assertEqualsComplex(ans, makeComp(-4.2, -2.4));

}

void testMulComp() {

complex in1, in2, ans;

testStart("mulComp");

in1 = makeComp(1.0, 2.0);

in2 = makeComp(0.2, 0.4);

ans = mulComp(in1, in2);

assertEqualsComplex(ans, makeComp(-0.6, 0.8));

in1 = makeComp(-5.0, -3.0);

in2 = makeComp(-0.5, -1.4);

ans = mulComp(in1, in2);

assertEqualsComplex(ans, makeComp(-1.7, 8.5));

in1 = makeComp(1.0, sqrt(2.0));

in2 = makeComp(sqrt(0.2), 0.4);

ans = mulComp(in1, in2);

assertEqualsComplex(

ans, makeComp(sqrt(5.0) / 5 - 0.4 \* sqrt(2.0), (2 + sqrt(10)) / 5));

}

void testConjComp() {

complex in1, ans;

testStart("conjComp");

in1 = makeComp(1.0, 2.0);

ans = conjComp(in1);

assertEqualsComplex(ans, makeComp(1.0, -2.0));

in1 = makeComp(-2.3, -4.5);

ans = conjComp(in1);

assertEqualsComplex(ans, makeComp(-2.3, 4.5));

in1 = makeComp(sqrt(5), sqrt(3));

ans = conjComp(in1);

assertEqualsComplex(ans, makeComp(sqrt(5), -sqrt(3)));

}

void testDivComp() {

complex in1, in2, ans;

testStart("divComp");

in1 = makeComp(2.5, 8.5);

in2 = makeComp(4.6, 2.6);

ans = divComp(in1, in2);

assertEqualsComplex(ans, makeComp(420.0 / 349.0, 815.0 / 698.0));

in1 = makeComp(-5.3, 4.5);

in2 = makeComp(3.7, -8.1);

ans = divComp(in1, in2);

assertEqualsComplex(ans, makeComp(-2803.0 / 3965.0, -1314.0 / 3965.0));

in1 = makeComp(sqrt(7), sqrt(5));

in2 = makeComp(-sqrt(3), sqrt(17));

ans = divComp(in1, in2);

assertEqualsComplex(ans, makeComp(-(sqrt(21) - sqrt(85)) / 20.0,

-(sqrt(119) + sqrt(15)) / 20.0));

}

void testGetTheta() {

complex in1;

testStart("getTheta");

in1 = makeComp(1.2, 2.3);

assertEqualsDouble(getTheta(in1), atan(2.3 / 1.2));

in1 = makeComp(3.4, -4.2);

assertEqualsDouble(getTheta(in1), -atan(4.2 / 3.4));

in1 = makeComp(-5.3, 2.9);

assertEqualsDouble(getTheta(in1), atan2(2.9, -5.3));

in1 = makeComp(-1.3, -sqrt(3));

assertEqualsDouble(getTheta(in1), atan2(-sqrt(3), -1.3));

}

void testPrintCompRT() {

complex in1;

testStart("testPrintCompRT");

printf("~~ Print test ~~\n");

in1 = makeComp(1.0, 1.0);

printCompRT(in1);

in1 = makeComp(3.0, -4.0);

printCompRT(in1);

in1 = makeComp(-5.0, 5 \* sqrt(3));

printCompRT(in1);

in1 = makeComp(-24, -10);

printCompRT(in1);

}

void testMakeCompRT() {

complex ans;

testStart("makeCompRT");

// 3∠0

ans = makeCompRT(3.0, 0.0);

assertEqualsComplex(ans, makeComp(3.0, 0.0));

// 2∠π/2

ans = makeCompRT(2.0, M\_PI / 2.0);

assertEqualsComplex(ans, makeComp(0.0, 2.0));

// 1∠(-π/4)

ans = makeCompRT(1.0, -M\_PI / 4.0);

assertEqualsComplex(ans, makeComp(1 / sqrt(2), -1 / sqrt(2)));

}

void testAddComp() {

complex in1, in2, ans;

testStart("AddComp");

in1 = makeComp(1, 2);

in2 = makeComp(0.1, 1.2);

ans = addComp(in1, in2);

assertEqualsComplex(ans, makeComp(1.1, 3.2));

in1 = makeComp(sqrt(2), sqrt(2));

in2 = makeComp(33, 10);

ans = addComp(in1, in2);

assertEqualsComplex(ans, makeComp(33 + sqrt(2), 10 + sqrt(2)));

in1 = makeComp(-1, -5);

in2 = makeComp(4, 3.3);

ans = addComp(in1, in2);

assertEqualsComplex(ans, makeComp(3, 3.3 - 5));

}

void testCmulComp() {

complex in1, ans;

double a = 2;

testStart("cmulComp");

in1 = makeComp(3, 4);

ans = cmulComp(in1, a);

assertEqualsComplex(ans, addComp(in1, in1));

a = 1.2;

in1 = makeComp(5.1, 8.8);

ans = cmulComp(in1, a);

assertEqualsComplex(ans, makeComp(1.2 \* in1.real, 1.2 \* in1.image));

a = 0.3;

in1 = makeComp(-5, 10);

ans = cmulComp(in1, a);

assertEqualsComplex(ans, makeComp(0.3 \* in1.real, 0.3 \* in1.image));

}

void testGetR2() {

complex in1;

double ans;

in1 = makeComp(10, 3);

ans = getR2(in1);

testStart("getR2");

assertEqualsDouble(ans, 10 \* 10 + 3 \* 3);

in1 = makeComp(1.2, 4.5);

ans = getR2(in1);

assertEqualsDouble(ans, 1.2 \* 1.2 + 4.5 \* 4.5);

in1 = makeComp(sqrt(3), sqrt(5));

ans = getR2(in1);

assertEqualsDouble(ans, 3 + 5);

}

void testInvComp() {

complex in1, ans;

in1 = makeComp(3, 19);

ans = invComp(in1);

testStart("invComp");

assertEqualsComplex(ans, makeComp(3.0 / 370, -19.0 / 370));

in1 = makeComp(3, -sqrt(5));

ans = invComp(in1);

assertEqualsComplex(ans, makeComp(3.0 / 14, sqrt(5) / 14));

in1 = makeComp(2, 1);

ans = invComp(in1);

assertEqualsComplex(ans, makeComp(0.4, -0.2));

}

void testGetR() {

complex in1;

double ans;

in1 = makeComp(1.0, -7.0);

ans = getR(in1);

testStart("getR");

assertEqualsDouble(ans, 5.0 \* sqrt(2.0));

in1 = makeComp(341.0, 587.0);

ans = getR(in1);

assertEqualsDouble(ans, 5.0 \* sqrt(18434.0));

in1 = makeComp(sqrt(5.0), -11.0);

ans = getR(in1);

assertEqualsDouble(ans, 3.0 \* sqrt(14.0));

}

void testPrintComp() {

complex in1;

in1 = makeComp(0.00000000001, 2);

testStart("printComp");

printf("~~ Print test ~~\n");

printComp(in1);

in1 = makeComp(0, 0);

printComp(in1);

in1 = makeComp(-5.5, -9);

printComp(in1);

in1 = makeComp(1.0 / 2, sqrt(2));

printComp(in1);

}

int main() {

testMakeComp();

testSubComp();

testMulComp();

testConjComp();

testDivComp();

testGetTheta();

testPrintCompRT();

testMakeCompRT();

testAddComp();

testCmulComp();

testGetR2();

testInvComp();

testGetR();

testPrintComp();

testErrorCheck(); // これは消さないこと

return 0;

}

**Complex.c**

#include "complex.h"

#include <math.h>

#include <stdio.h>

complex makeComp(double real, double image) {

complex ans;

ans.real = real;

ans.image = image;

return ans;

}

complex subComp(complex a, complex b) {

//

return makeComp(a.real - b.real, a.image - b.image);

}

complex mulComp(complex a, complex b) {

//

return makeComp(a.real \* b.real - a.image \* b.image,

a.real \* b.image + a.image \* b.real);

}

complex conjComp(complex a) {

//

return makeComp(a.real, -a.image);

}

complex divComp(complex a, complex b) {

//

return mulComp(a, invComp(b));

}

double getTheta(complex a) {

//

return atan2(a.image, a.real);

}

void printCompRT(complex a) {

printf("%10lf∠%12lf\n", getR(a), getTheta(a) \* 180 / M\_PI);

}

complex makeCompRT(double r, double theta) {

return makeComp(r \* cos(theta), r \* sin(theta));

}

complex addComp(complex a, complex b) {

return makeComp(a.real + b.real, a.image + b.image);

}

complex cmulComp(complex a, double k) {

return makeComp(a.real \* k, a.image \* k);

}

double getR2(complex a) {

return a.real \* a.real + a.image \* a.image;

}

complex invComp(complex a) {

return cmulComp(conjComp(a), 1 / getR2(a));

}

double getR(complex a) {

return sqrt(getR2(a));

}

void printComp(complex a) {

printf("%10lf%+10lfj\n", a.real, a.image);

}

**Complex.h**

#ifndef \_\_COMPLEX\_H\_\_

#define \_\_COMPLEX\_H\_\_

typedef struct {

double real;

double image;

} complex;

complex makeComp(double real, double image);

complex subComp(complex a, complex b);

complex mulComp(complex a, complex b);

complex conjComp(complex a);

complex divComp(complex a, complex b);

double getTheta(complex a);

void printCompRT(complex a);

complex makeCompRT(double r, double theta);

complex addComp(complex a, complex b);

complex cmulComp(complex a, double k);

double getR2(complex a);

complex invComp(complex a);

double getR(complex a);

void printComp(complex a);

#endif

**# 中間試験対策**

1. 複素数構造体 complex に関して以下の問に答えよ

1. 複素数構造体の型宣言を記述せよ。ただし、実部・虚部のメンバは、それぞれ倍精度型変数 real, image とする。

typedef struct {

double real;

double image;

} complex;

2. 実部と虚部の実数を渡して、複素数を取得する関数 makeComp を作成したい。この関数のためのテスト testMakeComp を記述したが、テストに不具合があるため、間違った実装でもテストを通過してしまう。間違った実装を解答欄に記述せよ。

void testMakeComp() {

complex in;

testStart("makeComp");

in = makeComp(1.2, 3.4);

assertEqualsDouble(in.real, 1.2);

assertEqualsDouble(in.image, 3.4); // 3.4 = 1.2 + 2.2

in = makeComp(-1.2, 1.0);

assertEqualsDouble(in.real, -1.2);

assertEqualsDouble(in.image, 1.0); // 1.0 = -1.2 + 2.2

}

complex makeComp(double real, double image) {

complex ans;

ans.real = real;

ans.image = real + 2.2;

return ans;

}

3. この間違った実装でエラーを出すように、テスト関数を修正せよ(修正した行のみを記せばよい)。

in = makeComp(-1.2, 1.5);

assertEqualsDouble(in.image, 1.5);

4. 複素数構造体のさまざまな関数が作成できたので、R, L, C の各インピーダンスを複素数構造体に代入する `makeR`, `makeL`, `makeC` 関数をそれぞれ作成する。各関数のテストにおける\*\*「」\*\*内に当てはまる数値を解答欄に記入せよ。返り値や引数については、コメントから推測すること。C言語的には数式でも通るが計算力を問いたいので数値で回答すること。なお、e表記は使ってよい。

void testMakeR() {

testStart("makeR");

// 100[Ω]の抵抗

complex x = makeR(100.0);

assertEqualsComplex(x, makeComp(100.0, 0.0);

// 47[kΩ]の抵抗

x = makeR(4.7e4);

assertEqualsComplex(x, makeComp(4.7e4, 0.0));

}

void testMakeL() {

testStart("makeL");

// 25[mH]のコイル, ω=100[rad/sec]

x = makeL(25e-3, 100.0);

assertEqualsComplex(x, makeComp(0.0, 25e-1));

// 15[mH]のコイル, ω=1000[rad/sec]

x = makeL(15e-3, 1.0e3);

assertEqualsComplex(x, makeComp(0.0, 15);

}

XL = wL

void testMakeC() {

testStart("makeC");

// 25[μF]のコンデンサ, ω=100[rad/sec]

x = makeC(25.0e-6, 100.0);

assertEqualsComplex(x, makeComp(0.0, -400.0));

// 500[μF]のコンデンサ, ω = 1000[rad/sec]

x = makeC(500e-6, 1.0e3);

assertEqualsComplex(x, makeComp(0.0, -2.0));

}

XC = -1/(wc)

5. (2) や (4) において、関数を複数回テストしている理由を説明せよ

6. `makeR`, `makeL`, `makeC` を実装した。各関数の実装における\*\*「」\*\*内に当てはまる内容を解答欄に記入せよ。

complex makeR(double r) {

return makeComp(r, 0.0);

}

complex makeL(double l, double w) {

return makeComp(0.0, w \* l);

}

complex makeC(double c, double w) {

return makeComp(0.0, -1/(w \* c));

}

```

7. インピーダンスを並列接続する関数 para を作成したい。引数は二つの複素数構造体で、返り値は合成インピーダンスを示す複素数構造体である。関数の実装を作成せよ。なお、授業中に作成した makeComp 以外の関数は利用してよい  (invComp, mulComp, addComp,  divComp など)。

complex para(complex a, complex b) {

return invComp(addComp(invComp(a), invComp(b)));

}

8. 上記の関数群を用いて，以下の回路のインピーダンスを計算する関数を作成したい。各関数は実数型の抵抗値, コンデンサの容量, コイルのインダクタンス，および角周波数を図に書かれた変数名で受け取るものとする．返り値は `compex` 型のインピーダンスである。この関数の実装を記述せよ。`makeR`, `makeL`, `makeC`, `para`, `addComp` だけを使用し，makeComp で直接計算式を書かないこと。また，作成した関数をうまく使って、プログラム記述量をなるべく省くこと(前に作った関数を使ってよい)。

addComp(makeR(r), makeC(c, w))