複素数の使用法（シラバス１２回目）

【1】複素数

複素数（**complex** numbers）zは虚数単位

$$\begin{cases} i \\ i^2 = -1 \end{cases}$$

を使って２つの実数x, yから

$$z = x + iy$$

と作ります。とくに、

xをzの**実**数部（**real** part）：$x = \mathrm{Re}(z)$

yをzの**虚**数部（**imarginary** part）：$y = \mathrm{Im}(z)$

といいます。また、iを$-i$に変える変換を

複素**共役**変換（complex **conjugate**）

と呼んで、z^*や$\overline{z}$で表し

$$\begin{cases} z^* = x - iy \\ \overline{z} = x - iy \end{cases}$$

になります。また、複素数とベクトルは１対１に対応し、

$$\text{座標}(x, y) : \begin{cases} \boldsymbol{x} = x\boldsymbol{i} + y\boldsymbol{j} \\ z = x + iy \end{cases}$$

と表す事ができます。(x, y)までの距離$\sqrt{x^2 + y^2}$は、

$$x^2 + y^2 = \begin{cases} \boldsymbol{x} \cdot \boldsymbol{x} = |\boldsymbol{x}|^2 \\ zz^* = z^* z = |z|^2 \end{cases} \Rightarrow \sqrt{x^2 + y^2} = \begin{cases} |z| \\ |\boldsymbol{x}| \end{cases}$$

になります。$|\ |$は**絶対**値（**absolute** number）を表す記号で、ベクトルや複素数の長さを与えます。

【2】＋、－、×、÷、||

複素数の四則演算は、それぞれ実数部や虚数部ごとの演算と$i \times i = -1$に注意して計算します。

足し算（**add** number）

$$\begin{cases} z_1 = x_1 + iy_1 \\ z_2 = x_2 + iy_2 \end{cases} \Rightarrow z_1 + z_2 = x_1 + x_2 + i(y_1 + y_2)$$

引き算（**subtract** number）

$$\begin{cases} z_1 = x_1 + iy_1 \\ z_2 = x_2 + iy_2 \end{cases} \Rightarrow z_1 - z_2 = x_1 - x_2 + i(y_1 - y_2)$$

掛け算（**multiply** number）

$$\begin{cases} z_1 = x_1 + iy_1 \\ z_2 = x_2 + iy_2 \end{cases} \Rightarrow z_1 z_2 = (x_1 + iy_1)(x_2 + iy_2) = x_1 x_2 - y_1 y_2 + i(y_1 x_2 + x_1 y_2)$$

割り算（**devide** number）

$$\begin{cases} z_1 = x_1 + iy_1 \\ z_2 = x_2 + iy_2 \end{cases} \Rightarrow \frac{z_1}{z_2} = \frac{x_1 + iy_1}{x_2 + iy_2} = \frac{x_1 x_2 + y_1 y_2 + i(y_1 x_2 - x_1 y_2)}{x_2^2 + y_2^2}$$

複素共役（complex **conjugate**）

$$z = x + iy \Rightarrow z^* = x - iy$$

絶対値（**absolute** number）

$$|z| = \sqrt{x^2 + y^2} \Rightarrow \frac{z_1}{z_2} = \frac{x_1 + iy_1}{x_2 + iy_2} = \frac{x_1 x_2 + y_1 y_2 + i(y_1 x_2 - x_1 y_2)}{x_2^2 + y_2^2}$$

と計算します。

【3】C言語での複素数

C言語は実数による演算ができますが、複素数の演算はできません。そこで、複素数の演算規則を教えないと行けません。C言語での複素数を作成するのに都合の良い方法が用意されています。**構造体**を作成する方法です。そこで、複素数用の構造体を作成します。これは、次のような書式で与えられます。

まず、名前を **COMPLEX** とします。これは、 **int** や **double** と同じで

- int:**整数型**
- double:**倍精度浮動小数点型**
- COMPLEX:**複素数型**

となります。自作の型として複素数型を作ります。その型は、2つの実数(rとi)を同時に持て

```
typedef struct tagCOMPLEX{
    double r; // real=実
    double i; // imarginary=虚
} COMPLEX;
```

と表します。この複素数型（**COMPLEX**）は、整数型（**int**）と同じ使い方ができて、

```
int x;
COMPLEX z;
```

と使います。この時、zは2つの実数を持ち

- z.r
- z.i

として表せます。これが複素数 z の実数部と虚数部になります。例えば、2つの複素数、z_1 と z_2 の足し算 $z = z_1 + z_2$ は、

$$z = z_1 + z_2$$

$$\underbrace{\underbrace{x}_{z.r} + i\underbrace{y}_{z.i}}_{z} = \underbrace{x_1}_{z1.r} + \underbrace{x_2}_{z2.r} + i\left(\underbrace{y_1}_{z1.i} + \underbrace{y_2}_{z2.i}\right)$$

に習って、成分ごとに計算します：

```
COMPLEX z1, z2;
COMPLEX z;

z.r = z1.r+z2.r;
z.i = z1.i+z.2i;
```

とすれば、2 つの複素数 z1, z2;の足し算が z になります（COMPLEX z1, z2, z;とまとめても良い）。

定数の複素数は、

- z=2+3i: z={2, 3};

と記述します。従って、**虚数単位 i** は、C言語で ai で表すと

- ai = {0, 1};

になります。

【4】複素数演算用の自作関数

複素数演算をする自作関数を作ります。

- 足し算：$z = z_1+z_2$
  z = **add**(z1, z2)
- 引き算：$z = z_1-z_2$
  z = **subtract**(z1, z2)
- 掛け算：$z = z_1 \times z_2$
  z = **multiply**(z1, z2)
- 割り算：$z = z_1/z_2$
  z = **devide**(z1, z2)
- 複素共役：z^*
  w = **conjugate**(z)
- 絶対値：$x = |z|$
  x = **absolute**(z)
  (*) x は double x です。
- x 倍：$x = x \times z$
  x = **multilpy_number**(z)

になります。**【３】**の例では、足し算の作り方を示しています。

自作関数 add(z1, z2)を完成するには、C言語のルールにより、

- 使う変数はすべて宣言する
  - 使う変数は、z1, z2,と計算結果の z
    - **COMPLEX** z1

- ✧ **COMPLEX** z2
- ✧ **COMPLEX** z;

● 計算結果を return で返す

➢ return z;とする。

- ✧ 関数は複素数の COMPLEX 型を返す
- ✧ **COMPLEX** add

になります。以上を踏まえて

$$\boxed{z = z_1 + z_2}$$

```
COMPLEX add(COMPLEX z1, COMPLEX z2)
{
    COMPLEX z;
```

$$\boxed{z = \overbrace{x}^{z.r} + i\overbrace{y}^{z.i} = z_1 + z_2 = \overbrace{x_1}^{z1.r} + \overbrace{x_2}^{z2.r} + i\left(\overbrace{y_1}^{z1.i} + \overbrace{y_2}^{z2.i}\right)}$$

```
    z.r = z1.r+z2.r;
    z.i = z1.i+z2.i;

    return z;
}
```

と作成できます。

第７回目レポート

レポートの回数(７回目)、学生証番号と氏名を明記すること

必ず表紙を付け、最後のページ(の添付用)を表紙の次に入れる

（Ａ４レポート用紙使用のこと）

1） 下に記述してあるプログラム(**複素数操作関数プログラム**)で、自作関数

subtract $\boxed{z = \overbrace{x}^{z.r} + i\overbrace{y}^{z.i} = z_1 - z_2 = \underbrace{x_1}_{z1.r} - \underbrace{x_2}_{z2.r} + i\left(\underbrace{y_1}_{z1.i} - \underbrace{y_2}_{z2.i}\right)}$

multiply $\boxed{z = \overbrace{x}^{z.r} + i\overbrace{y}^{z.i} = z_1 z_2 = \left(\underbrace{x_1}_{z1.r} + i\underbrace{y_1}_{z1.i}\right)\left(\underbrace{x_2}_{z2.r} + i\underbrace{y_2}_{z2.i}\right) = \underbrace{x_1}_{z1.r}\underbrace{x_2}_{z2.r} - \underbrace{y_1}_{z1.i}\underbrace{y_2}_{z2.i} + i\left(\underbrace{x_1}_{z1.r}\underbrace{y_2}_{z2.i} + \underbrace{x_2}_{z2.r}\underbrace{y_1}_{z1.i}\right)}$

divide $\boxed{z = \overbrace{x}^{z.r} + i\overbrace{y}^{z.i} = \frac{z_1}{z_2}}$

conjugate $\boxed{z^*}$

absolute $\boxed{|z|}$

multiply_number $\boxed{xz}$

が未完成になっている。？？？(1 行分とは限らない)を完成し、プログラムを実行せよ。提出は

- プログラム(下に記述してある複素数操作関数プログラムを完成させる)
- 実行したときの画面コピー

  【確認用】 正しくプログラムが作成されると、以下の数値が表示されます:

```
z1=1.000000+2.000000i
z2=4.000000+5.000000i

z = add(z1, z2): 5.000000+7.000000i
z = subtract(z1, z2): -3.000000+-3.000000i
z = multiply(z1, z2): -6.000000+13.000000i

z = divide(z1, z2): 0.341463+-0.073171i
z = conjugate(z1): 1.000000+-2.000000i
x = absolute(z1): 2.236068
z = multiply_number(2, z1): 2.000000+4.000000i
```

の2点です。

2） 1)のプログラムで表示させると **subtract 関数の結果表示の箇所**で

```
z = subtract(z1, z2): -3.000000+-3.000000i
```

の様に

```
-3.000000-3.000000 i
```

と表示すべき所

```
-3.000000+-3.000000 i
```

と表示される。「-3.000000-3.000000 i」と表示されるように自作関数 printz を修正し、新たに自作関数 printz_minus として作成せよ(printz_minus では、printz 内の数値「z.i」を、if 文を使用して、正と負で場合分けして表示させます)。提出は

- 自作関数 printz_minus のプログラム
- 実行したときの画面コピー(「-3.000000-3.000000i」の表示を確認)

の2点です。

複素数操作関数プログラム

```
#include <stdio.h>
#include <math.h>

typedef struct tagCOMPLEX{
     double r;
     double i;
} COMPLEX;

// z1+z2
COMPLEX add(COMPLEX z1, COMPLEX z2)
{
     COMPLEX z;

     z.r = z1.r+z2.r;
     z.i = z1.i+z2.i;

     return z;
}

// z1-z2
？？？ subtract(？？？)
{
     COMPLEX z;

     ？？？

     return z;
}

// z1*z2
COMPLEX multiply(？？？)
{
     COMPLEX z;

     ？？？

     return z;
}

// z1/z2
COMPLEX divide(COMPLEX z1, COMPLEX z2)
{
     COMPLEX z;
     COMPLEX bunsi;
```

```
        // z2 = 0 の時、エラーメッセージを出すためのプログラムです
        if(？？？){
            COMPLEX zero = {0, 0};

            printf("エラー：０で割っています¥n");
            getchar();

            return zero;
        }

        bunsi.r = ？？？;
        bunsi.i = ？？？;

        z.r = bunsi.r/？？？;
        z.i = bunsi.i/？？？;

        return z;
}

// z1*
COMPLEX conjugate(COMPLEX z1)
{
        COMPLEX z;

        z.r = z1.r;
        ？？？

        return z;
}

// |z|
？？？ absolute(COMPLEX z)
{
        return sqrt(z.r*z.r + z.i*z.i);
}

// x*z1
COMPLEX multiply_number(double x, COMPLEX z1)
{
        COMPLEX z;

        ？？？

        return z;
}

void printz(COMPLEX z)
{
        printf("%f+%fi", z.r, z.i);
}

int main(void)
{
        COMPLEX z1 = {1.0, 2.0};
        COMPLEX z2 = {4.0, 5.0};
        COMPLEX z;
```

```
    double x;

    printf("z1=");
    printz(z1);
    printf("¥n");
    printf("z2=");
    printz(z2);
    printf("¥n¥n");

    z = add(z1, z2);
    printf("z = add(z1, z2): ");
    printz(z);
    printf("¥n");

    z = subtract(z1, z2);
    printf("z = subtract(z1, z2): ");
    printz(z);
    printf("¥n");

    z = multiply(z1, z2);
    printf("z = multiply(z1, z2): ");
    printz(z);
    printf("¥n");

    z = divide(z1, z2);
    printf("z = divide(z1, z2): ");
    printz(z);
    printf("¥n");

    z = conjugate(z1);
    printf("z = conjugate(z1): ");
    printz(z);
    printf("¥n");

    x = absolute(z1);
    printf("x = absolute(z1): ");
    printf("%f¥n", x);
    printf("¥n");

    z = multiply_number (2, z1);
    printf("z = multiply_number(2, z1): ");
    printfz(z);

    return 0;
}
```

提出例(使える表紙は次のページ)

１ページ目　　２ページ目　　３ページ目以降

コンピュータ物理学演習 II

第 7 回目

月　日提出

学籍番号

氏名

第７回目レポート（添付用）

１） 次のプログラムにおいて自作関数

subtract

multiply

divide

absokute

において？？？完成しプログラムを実行せよ。提出は

- プログラム
- 実行したときの画面コピー

の2点です。

２） 1)のプログラムで表示させると

z = subtract(z1, z2): -3.000000+-3.000000i

の様に

-3.000000−3.000000ii

と表示すべき所

-3.000000+−3.000000ii

と表示される。「-3.000000−3.000000ii」と表示されるように自作関数 printz を修正し、新たに自作関数 printz_minus として作成せよ(if 文を使用して、printz を正と負で場賄分けします。)。提出は

- 自作関数 printz_minus のプログラム
- 実行したときの画面コピー（「-3.000000−3.000000ii」の表示を確認）

の2点です。