

第12回新しい型と構造体



1

今回の目標

- 新しい型の定義法を理解する。
- 構造体を理解する。

☆複素数同士を足し算する関数を作成し、その関数を利用するプログラムを作成する。

2

複素数の足し算

複素数は実部と虚部の2つの実数で、表現される。

$$z = a + bi$$

2つの複素数 $z_1 = a_1 + b_1i$ と $z_2 = a_2 + b_2i$ の和 $z_3 = a_3 + b_3i$ は、次式で与えられる。

$$\begin{aligned} z_3 &= z_1 + z_2 \\ &= (a_1 + a_2) + (b_1 + b_2)i \end{aligned}$$

3

typedef文(型の別名の付け方)

C言語では、いろいろなデータ型に自分の好きな名前をつけることができる。

宣言

```
typedef 既に使える型 別名;
```

例

```
typedef int Count;
Count i;
```

int, double, char や
int *, double *, char* や
既に定義した型

この宣言で、
Count型(int 型)
の変数iが用意される。

セミコロンを
忘れずに。

参考 `int i;`

新しい型名は、
大文字+小文字で命名すること。
(スタイル規則参照)

4

練習1

```
/*newtype.c 新しい型実験 コメント省略*/
#include <stdio.h>

#define SIZE 5

typedef int Count; /*序数を表す型*/
typedef double Real; /*実数を表す型*/

int main()
{
    /*ローカル変数宣言*/
    Count i; /*カウンタ*/
    Real x[SIZE]; /*実数データ*/
    /*次に続く。*/
```

```
/*続き*/
/*ローカル変数の初期化*/
i=0;
for(i=0;i<SIZE;i++)
{
    x[i]=(Real) i;
    /*全て異なる値にするため、インデックスで初期化。
    カウント型から実数型へキャストしている。*/
}

/*出力処理*/
for(i=0;i<SIZE;i++)
{
    printf("x[%d] = %6.2f ¥n",i,x[i]);
}
return 0;
}
```

構造体

(レコード型と呼ぶこともある。)

構造体とは、いくつかのデータを
1つのまとまりとして扱うデータ型。
プログラマが、定義してから使う。

一まとまりのデータ例

複素数: 実部と虚部	名刺: 所属、名前、連絡先
点: x座標、y座標	日付: 年、月、日、曜日
2次元ベクトル: x成分、y成分	本: 題名、著者、ISBN

7

構造体テンプレートの宣言 (構造体の定義)

宣言

```
struct 構造体タグ名
{
    型1   メンバ名1;
    型2   メンバ名2;
    型3   メンバ名3;
    :
};
```

構造体を構成する要素を
メンバといいます。

これを
構造体テンプレートという。

int, double, char
や
int *, double *, char*
や
既に定義した構造体型等

例

```
struct complex
{
    double real;
    double imag;
};
```

関数の記述と似ているが
セミコロンを忘れずに。

8

構造体の宣言

(構造体型の変数の用意の仕方)

宣言

```
struct 構造体タグ名 変数名;
```

ここに空白がある。

例

```
struct complex z;
```

この2つで、一つの型を表わしているので注意すること。

参考

```
int i;  
double x;
```

9

typedef文と構造体

(構造体型の変数の用意の仕方2)

構造体型の宣言は、
2つの文字列で一つの型を意味するので紛らわしい。
typedef文を使うとすっきりする。

宣言

```
typedef struct 構造体タグ 別名;
```

これ2つで、一つの型

例

```
typedef struct complex Complex;  
Complex z;
```

別名

参考

```
int i;
```

この宣言で,Complex型(複素数型)の変数zが用意される。

構造体のイメージ



char



int



double

既存の型

構造体テンプレート

```
struct complex  
{  
    double real;  
    double imag;  
};
```

セミコロンを忘れずに。

雛形の作成。



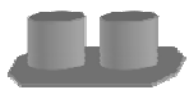
struct complex型の雛形

11

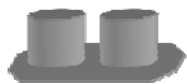
別名の定義

```
typedef struct complex Complex;
```

struct complex型の雛形



Complex型の雛形

2つの文字列で、
一つの型を表すと、
間違えやすい。

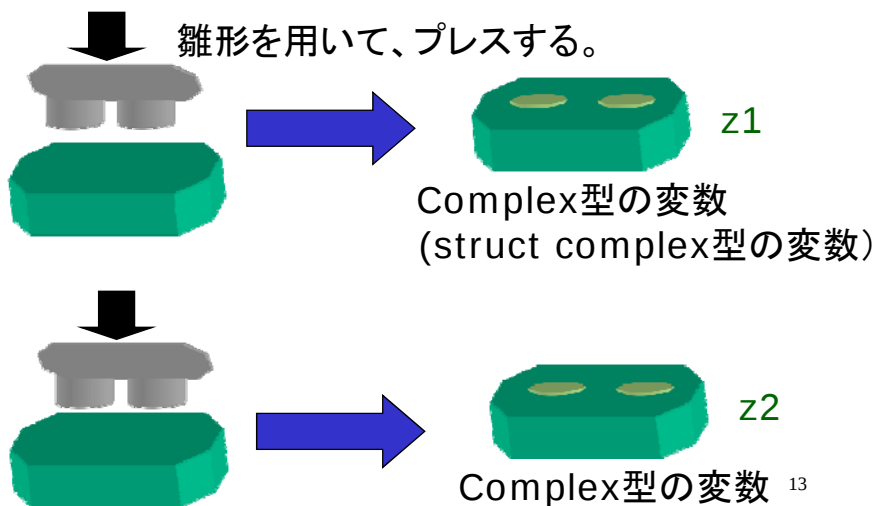
12

構造体型の変数宣言

```
Complex z1;
Complex z2;
```



Complex型の雛形



構造体のイメージ2



char



int



double

```
struct card
{
    char    initial;
    int     age;
    double  weight;
};
```

雛形の作成。



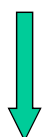
struct card 型の雛形

いろいろな型のデータを
一まとまりであつかうときには、
構造体はとくに便利。

別名の定義

```
typedef struct card Card;
```

struct card 型の雛形



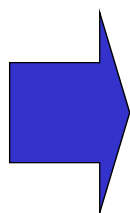
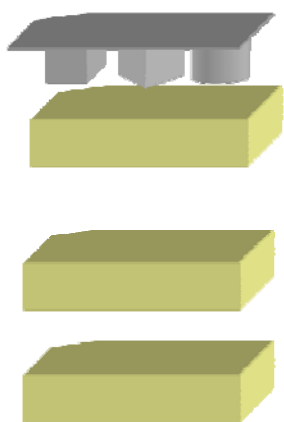
Card型の雛形



15

構造体型の配列宣言

```
#define MAXCARD 3  
Card x[MAXCARD];
```



Card型の変数



x[0]



x[1]



x[2]

16

構造体のメンバの参照

struct 型の変数のメンバの参照の仕方

書式

変数名. メンバ名

ドット(演算子の一つ)

これらを、メンバ名を定義している 型の変数として扱える。

例

z1.real

これはdouble 型の変数である。

x[0].inital

これはchar 型の変数である。

17

参照のイメージ

Complex z1;

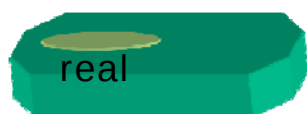
Complex型の雛形



Complex型の変数



z1



z1.real



z1.imag

18

演算子の結合力

演算子の結合力は他のどの演算子よりも強い。

`x[0].age++;` は `(x[0].age)++;` の意味

Complex * p; のとき、

`*p.real;` は `*(p.real);` の意味になってしまう

両方間違い。(メンバrealは、ポインタではない。)

```
(*p).real;
```

典型的には、
これが正しい。

(ソースの可読性の向上のため)他の演算子と一緒に使うときには、括弧を用いて意図を明確にすること。

19

構造体と代入演算子1 (構造体への値の入れ方1)

全てのメンバに値を代入する。

```
Complex  z1;  
(z1.real)=1.0;  
(z1.imag)=2.0;
```

間違い例

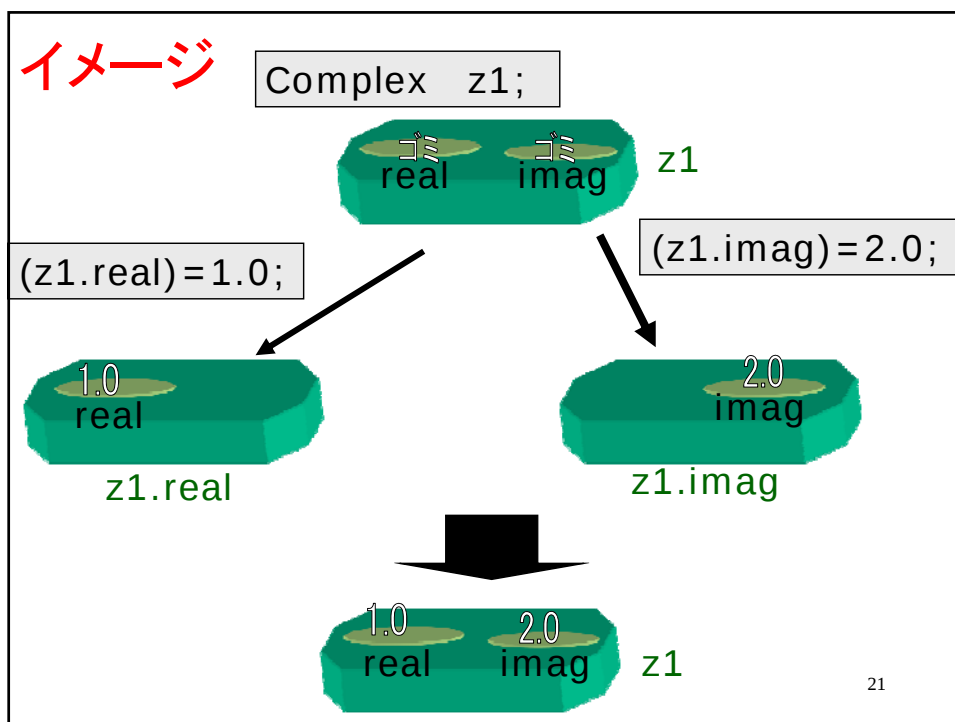

$$z1 = 1.0 + 2.0i;$$

```
z1 = (1.0, 2.0);
```

複素数だからって
こんなふうには
かけない。

ベクトル風にも
かけない。

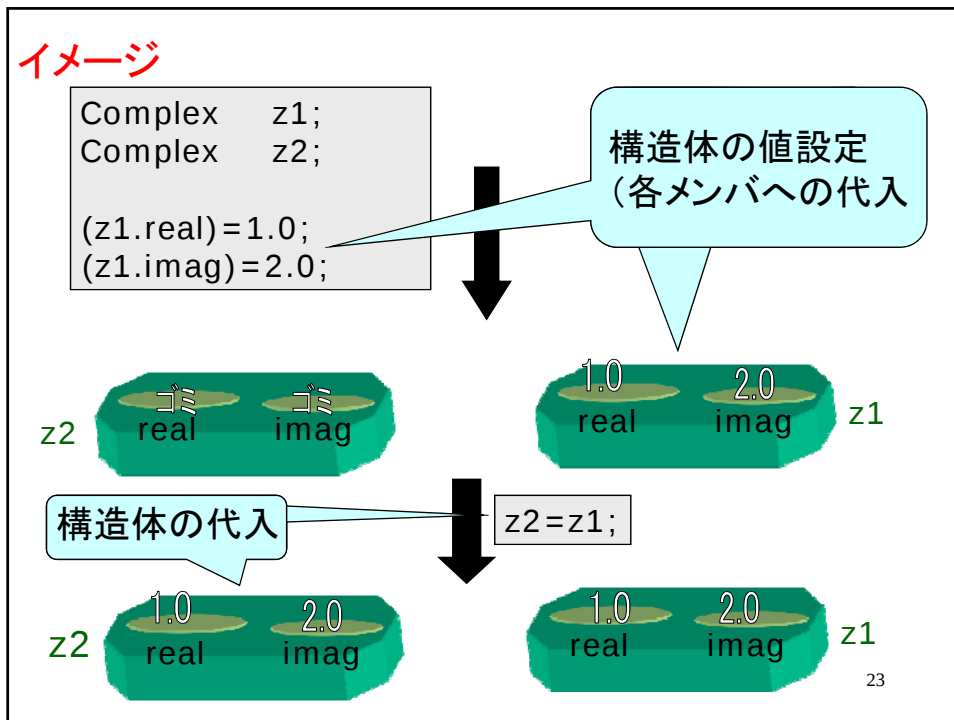
20



構造体と代入演算子2 (構造体への値の入れ方2)

同じ型の構造体同士で代入する。

```
Complex z1;  
Complex z2;  
  
(z1.real) = 1.0;  
(z1.imag) = 2.0;  
  
z2 = z1;
```



練習2

```
/*test_struct.c 構造体実験 コメント省略*/
#include <stdio.h>

struct complex
{
    double real;
    double imag;
};

typedef struct complex Complex;

/* 次に行く */
```

```
int    main()
{
    Complex    z1;
    Complex    z2;

    printf("メンバの読み込み¥n");
    printf("z1= (real?) + (imag?)i  ");
    scanf("%lf %lf",&(z1.real),&(z1.imag));

    printf("読み込み後¥n");
    printf("z1= %4.2f+ (%4.2f)i¥n",
           z1.real,z1.imag);
    printf("z2= %4.2f+ (%4.2f)i¥n",
           z2.real,z2.imag);

    /* 続く */
}
```

25

```
/*    続き    */
printf("z2=z1実行中¥n");
z2=z1;

printf("代入後¥n");
pritnf("z1= %4.2f + (%4.2f)i¥n",
       z1.real,z1.imag);
pritnf("z2= %4.2f + (%4.2f)i¥n",
       z2.real,z2.imag);

return 0;
}
```

26

複素数の和を求めるプログラム

```
/*
    作成日:yyyy/mm/dd
    作成者:本荘太郎
    学籍番号:B0zB0xx
    ソースファイル:pluscomp.c
    実行ファイル:pluscomp
    説明:構造体を用いて、2つの複素数の和を
        求めるプログラム。
    入力:標準入力から、2つの複素数z1とz2を入力。
        z1の(実部、虚部)、z2の(実部、虚部)の順
        で4つの実数を入力する。
    出力:標準出力にその2つの複素数の和を
        出力する。
*/
/* 次に行く */
```

27

```
/* 続き */
#include <stdio.h>

/* 構造体テンプレート */
struct complex /*複素数を表わす構造体*/
{
    double real; /*実部*/
    double imag; /*虚部*/
};

/* 新しい型名 */
typedef struct complex Complex; /*複素数型*/

/* プロトタイプ宣言*/
Complex plus_complex(Complex,Complex);
/*2つの複素数の和を求める関数*/
/* 次に行く */
```

28

```
/*main関数*/
int main()
{
    /*ローカル変数宣言*/
    Complex z1; /*複素数1*/
    Complex z2; /*複素数2*/
    Complex sum; /*複素数の和を蓄える変数*/
    /* 入力処理*/
    printf("2つの複素数z1,z2を入力して下さい。¥n");
    printf("z1の実部は?");
    scanf("%lf",&(z1.real));
    printf("z1の虚部は?");
    scanf("%lf",&(z1.imag));
    printf("z2の実部は?");
    scanf("%lf",&(z2.real));
    printf("z2の虚部は?");
    scanf("%lf",&(z2.imag));
    /* 次に行く */
}
```

```
/* 続き */
/*計算処理*/
sum=plus_complex(z1,z2);

/*出力処理*/
printf("sum=z1+z2 ¥n");
printf("( %4.2f+ (%6.2f)i )=",
        sum.real,sum.imag);
printf("( %4.2f+ (%4.2f)i )+",
        z1.real,z1.imag);
printf("( %4.2f+ (%4.2f)i)¥n",
        z2.real,z2.imag);
return 0; /*正常終了*/
}
/*main関数終了*/
```

```
/* 2つの複素数の和を求める関数
仮引数 z1,z2: 2つの複素数。
戻り値: 2つの複素数の和(z1+z2)
*/
Complex plus_complex(Complex z1, Complex z2)
{
    /* ローカル変数宣言 */
    Complex sum; /* 2つの複素数の和を蓄える */
    /* 命令記述 */
    (sum.real) = (z1.real) + (z2.real); /* 実部の計算 */
    (sum.imag) = (z1.imag) + (z2.imag); /* 虚部の計算 */

    return sum;
}
/* 関数 plus_complex の終了 */
/* プログラム pluscomp.c の終了 */
```

実行結果

```
$make
gcc pluscomplex.c -o pluscomplex
$./pluscomplex
2つの複素数z1,z2を入力して下さい。
z1の実部は? 1.0
z1の虚部は? 2.0
z2の実部は? 3.0
z2の虚部は? 4.0
sum=z1+z2
(4.00+(6.00)i)=(1.00+(2.00)i)+(3.00+(4.00)i)
$
```