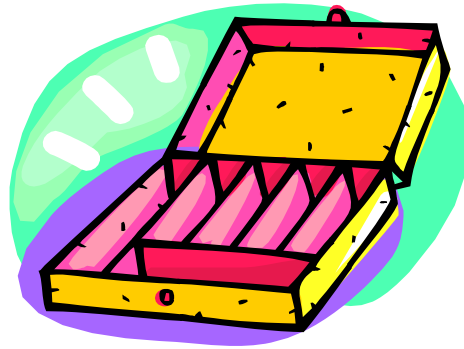


第12回構造体



1

今回の目標

- 構造体を理解する。
- 構造体の定義の仕方を理解する。
- 構造体型を理解する。
- 構造体型の変数、引数、戻り値を理解する。

☆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

構造体

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。
構造体型の変数、引数、戻り値等が利用できるよ
うになる。

(他の言語ではレコード型と呼ぶこともある。)

ひと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

3

構造体型の定義

(構造体テンプレートの宣言)

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。

4

構造体型の変数の用意の仕方 (構造体型の変数宣言)

宣言

```
struct 構造体タグ名 変数名;
```

ここに空白を書く。

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;
```

5

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。

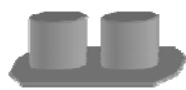
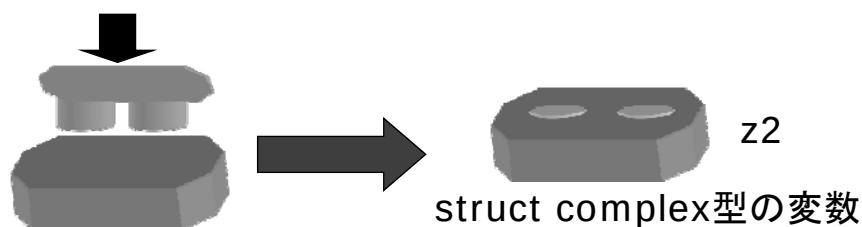
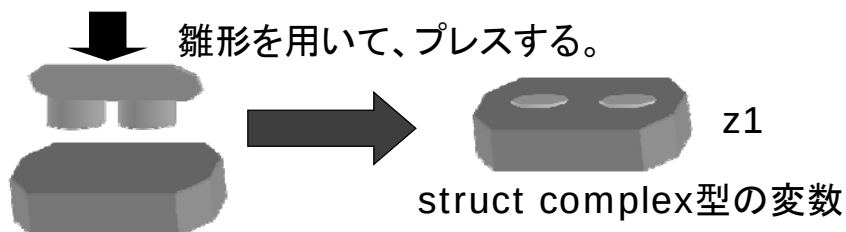


struct complex型の雛形

6

構造体型の変数宣言

```
struct complex z1;  
struct complex z2;
```

struct 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;  
};
```

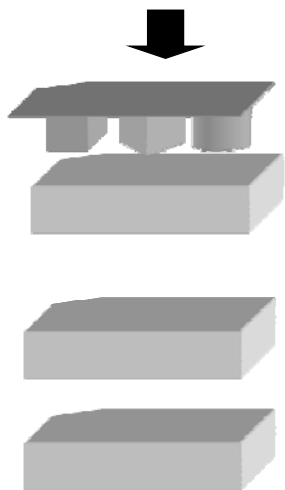
雛形の作成。

struct card 型の
雛形

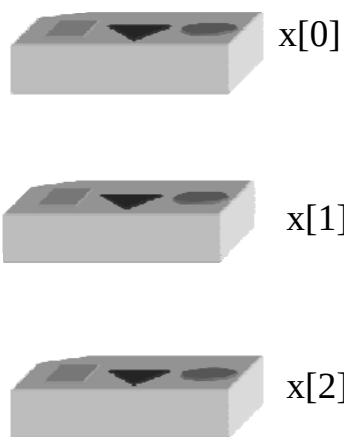
いろいろな型のデータを
一まとめりであつかうときには、
構造体はとくに便利。

構造体型の要素をもつ配列の宣言

```
#define MAXCARD 3
struct card x[MAXCARD];
```



struct card型の配列要素



9

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

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].initial

これはchar 型の変数である。

10

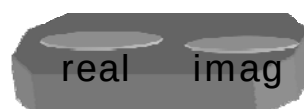
参照のイメージ

```
struct complex  z1;
```

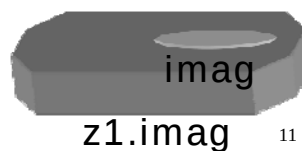
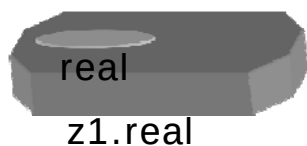
struct complex型の雛形



struct complex型の変数



z1



11

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。

.(ドット演算子) > ++
--

`x[0].age++;` は `(x[0].age)++;` の意味

(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。

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。

```
struct complex  z1;  
  
(z1.real)=1.0;  
(z1.imag)=2.0;
```

OK

複素数だからって
こんなふうには
かけない。

間違いの例

X

```
z1=1.0+2.0i;
```

NG

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

NG

ベクトル風にも
かけない。

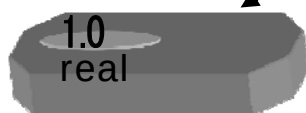
13

イメージ

```
struct complex  z1;
```



```
(z1.real)=1.0;
```

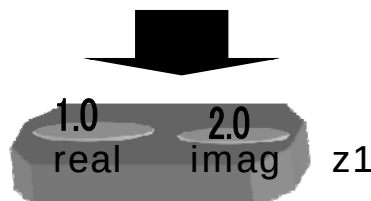


z1.real

```
(z1.imag)=2.0;
```



z1.imag



14

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。

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

15

イメージ

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

構造体の値設定
(各メンバへの代入)

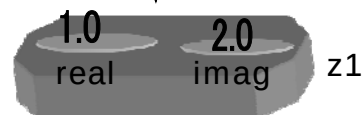
z2



z2 structure with 'real' and 'imag' members containing garbage values (ゴミ).

1.0 2.0
real imag

z1

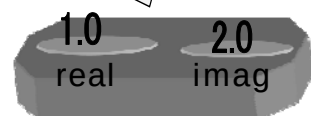


z1 structure with 'real' member containing 1.0 and 'imag' member containing 2.0.

構造体の代入

z2=z1;

z2



z2 structure with 'real' member containing 1.0 and 'imag' member containing 2.0.

1.0 2.0
real imag

z1



z1 structure with 'real' member containing 1.0 and 'imag' member containing 2.0.

16

プログラム例1： 構造体の効果確認

```
/*test_struct.c 構造体の効果確認 コメント省略*/
#include <stdio.h>

struct complex
{
    double real;
    double imag;
};

/* 次が続く */
```

17

```
int main()
{
    struct complex z1;
    struct complex z2;

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

18

```

/*    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;
}

```

19

プログラム例2の原理: 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。

$$z = a + bi$$

↑
実部
↑
虚部

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。

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

20

プログラム例2: 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: B00B0xx
    ソースファイル: addcomp.c
    実行ファイル: addcomp
    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。
          和は複素数であり、その実部と虚部は実数。
*/
/* 続く */
```

21

```
/* 続き */
#include <stdio.h>

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

/* プロトタイプ宣言 */
struct complex scan_complex();
/* 標準入力から複素数を読み込む関数 */

void print_complex(struct complex z);
/* 標準出力へ複素数を出力する関数 */

struct complex add_complex (struct complex z1,
                           struct complex z2);
/* 2つの複素数の和を求める関数 */
/* 続く */
```

22

```
/* main関数の定義 */
int    main()
{      /*ローカル変数宣言*/
    struct complex z1; /* 入力された複素数1 */
    struct complex z2; /* 入力された複素数2 */
    struct complex sum; /* 二つの複素数の和 */

    /* 足し合わせるべき二つの複素数の入力 */
    z1 = scan_complex();
    z2 = scan_complex();

    /* 複素数の足し算 */
    sum = add_complex (z1, z2);

/*main関数続く*/
```

23

```
/*続き main関数*/

    /* 計算結果の出力 */
    print_complex(z1);
    printf("+");
    print_complex(z2);
    printf("=");
    print_complex(sum);
    printf("¥n");

    /*正常終了*/
    return 0;
}
/*main関数終了*/
/*続く*/
```

24

```
/*続き */

/* 標準入力から複素数を受け取る関数。
   標準入力から実部、虚部の順にdouble 値を受け取る。
   仮引数 :なし
   戻り値:読み込まれた二つの実数値をそれぞれ実部、虚部とする複素数。
*/
struct complex scan_complex()
{
    /*ローカル変数宣言*/
    struct complex z; /*読み込まれる複素数*/
    /*入力処理*/
    scanf("%lf", &(z.real)); /*実部*/
    scanf("%lf", &(z.imag)); /*虚部*/

    return z;
}
/*関数 scan_complexの定義終了 */

/*続く*/
```

25

```
/* 続き */

/*複素数を「( 実部+(虚部)i )」の形式で標準出力に出力する関数。
   仮引数 z:表示すべき複素数
   戻り値:なし
*/
void print_complex(struct complex z)
{
    /*出力処理*/
    printf(" ( %4.1f + (%4.1f)i )", z.real, z.imag);
    return;
}
/*関数 print_complexの定義終了 */

/*続く*/
```

26

```
/* 続き */
/* 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)
*/
struct complex add_complex (struct complex z1,
                             struct complex z2)
{
    /*ローカル変数宣言*/
    struct complex sum; /*2つの複素数の和を蓄える*/
    /*計算処理*/
    (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 add_complex の定義終了 */
/* プログラム addcomp.c の終了 */
```

27

プログラム例2の実行結果

```
$. /addcomplex
2つの複素数z1,z2を入力して下さい。
( 4.0+( 6.0)i)=( 1.0+( 2.0)i)+( 3.0+( 4.0)i)
$
```

28