

C プログラミング

— 構造体 —

早稲田大学

本日の目標

- 構造体の復習と発展
 - 構造体変数の宣言
 - メンバの参照
 - 構造体へのポインタ
 - 構造体を返す関数
 - 構造体配列

構造体と配列

【配列】

- 配列は 1 つ以上の変数を集められるデータ構造
- ただし、変数の型は同じものでなければならない

テスト点数								
91	78	69	84	86	92	77	81	90

データ構造

int 型

【構造体】

- **構造体**とは 1 つ以上の要素を集めて作られるデータ構造
- 配列のように全要素が同じ型である必要はなく、異なる型の要素を自由に組み合わせられる。

学生	学生 1	学生 2	学生 3
学籍番号	001	002	003
得点	90	94	72
偏差値	64.2	67.1	56.3

...

データ構造

int 型

int 型

double 型

構造体変数の宣言

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

```
struct student{  
    char name[20];  
    int  math;  
    int  phys;  
};
```

```
int main(void){  
    struct student a, b;  
    struct student c={"Frank", 90};  
    :  
    return 0;  
}
```

- **構造体タグ**は宣言した構造体の型を代表する識別子である.
- **メンバ**が構造体を構成する変数.
- 初期化は配列と同様. 各メンバに対する初期値を, 先頭から順に並べる.
- 与えられていない要素は0で初期化される.

メンバの参照

- 構造体の各メンバを参照するには,
構造体メンバ演算子 (.) を利用する.

```
#include <stdio.h>
#include <string.h>          /*strcpy 関数を使うのに必要*/

struct student{
    char name[20];
    int  math;
    int  phys;
};

int main(void){
    struct student a, b;

    strcpy(a.name,"Frank");
    a.math = 90;
    a.phys = 83;
    ...
}
```

構造体型変数の代入

- 代入演算子(=)を用いて、構造体全体を一括してコピーできる.
- ただし、両辺の構造体の型は同一でなければいけない.
- 配列では一括代入ができない

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

```
struct student{
    char name[20];
    int math;
    int phys;
};
int main(void){
```

```
    struct student a,b;
```

```
    strcpy(a.name, "Frank");
    a.math = 90;
    a.phys = 83;
```

```
    printf("    Name:%s\n",a.name);
    printf("    Math:%d\n",a.math);
    printf("    Physics:%d\n",a.phys);
```

```
    b=a;
    printf("    Name:%s\n",b.name);
    printf("    Math:%d\n",b.math);
    printf("    Physics:%d\n",b.phys);

    return 0;
```

```
}
```

【実行結果】

```
    Name:Frank
    Math:90
    Physics:83
    Name:Frank
    Math:90
    Physics:83
```

構造体へのポインタ

- 構造体変数を宣言する際、ポインタとして宣言することもできる.
- 構造体を指すポインタは構造体の先頭のメンバのアドレスをもつ.
- 構造体型変数のアドレスは、アドレス演算子 (&) を用いる.

```
struct student{  
    char name[20];  
    int  math;  
    int  phys;  
};  
  
int main(void){  
    struct student a, *pa;  
    pa = &a;  
    ...  
}
```

ポインタ	メモリ内容
pa	a.name
	a.math
	a.phys

と宣言した場合右のようにメモリが確保される.

構造体へのポインタ

- 構造体変数を宣言する際、ポインタとして宣言することもできる.
- 構造体を指すポインタは構造体の先頭のメンバのアドレスをもつ.
- 構造体型変数のアドレスは、**アドレス演算子 (&)** を用いる.

```
struct student{
    char name[20];
    int  math;
    int  phys;
};

int main(void){
    struct student a, *pa;
    pa = &a;
    ...
}
```

ポインタ	メモリ内容
pa	a.name
	a.math
	a.phys

と宣言した場合右のようにメモリが確保される.

構造体へのポインタ

- ポインタの指す構造体のメンバを参照する際は、**アロー演算子 (->)** を利用して、**ポインタ名 -> メンバ名**とする。
- これは **(*ポインタ名).メンバ名**と同じ意味だが一般には上を使う。

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

struct student{
    char name[20];
    int math;
    int phys;
};

int main(void){

    struct student a,*pa;

    strcpy(a.name, "Frank");
    a.math = 90;
    a.phys = 83;

    printf("    Name:%s\n",a.name);
    printf("    Math:%d\n",a.math);
    printf("    Physics:%d\n",a.phys);

    pa=&a;
    strcpy(pa->name, "Thomas");
    pa->phys = 92;

    printf("    Name:%s\n",pa->name);
    printf("    Math:%d\n",pa->math);
    printf("    Physics:%d\n",pa->phys);

    return 0;
}
```

【実行結果】

```
Name:Frank
Math:90
Physics:83
Name:Thomas
Math:90
Physics:92
```

構造体へのポインタ

- ポインタの指す構造体のメンバを参照する際は、**アロー演算子 (->)** を利用して、**ポインタ名 -> メンバ名**とする。
- これは **(*ポインタ名).メンバ名**と同じ意味だが一般には上を使う。

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

struct student{
    char name[20];
    int math;
    int phys;
};

int main(void){

    struct student a,*pa;

    strcpy(a.name, "Frank");
    a.math = 90;
    a.phys = 83;

    printf("    Name:%s\n",a.name);
    printf("    Math:%d\n",a.math);
    printf("    Physics:%d\n",a.phys);

    pa=&a;
    strcpy(pa->name, "Thomas");
    pa->phys = 92;

    printf("    Name:%s\n",pa->name);
    printf("    Math:%d\n",pa->math);
    printf("    Physics:%d\n",pa->phys);

    return 0;
}
```

【実行結果】

```
Name:Frank
Math:90
Physics:83
Name:Thomas
Math:90
Physics:92
```

例題：

- 下のプログラムを実行した場合の出力はどのようなになるか考えよ。

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

struct student{
    char name[20];
    int math;
    int phys;
};

int main(void){
    struct student a,*pa;
    strcpy(a.name, "Frank");
    a.math = 90;
    a.phys = 83;

    printf("    Name:%s\n",a.name);
    printf("    Math:%d\n",a.math);
    printf(" Physics:%d\n",a.phys);

    pa=&a;
    strcpy(pa->name, "Thomas");
    pa->phys = 92;
```

```
printf("    Name:%s\n",pa->name);
printf("    Math:%d\n",pa->math);
printf(" Physics:%d\n",pa->phys);
printf("    Name:%s\n",a.name);
printf("    Math:%d\n",a.math);
printf(" Physics:%d\n",a.phys);

return 0;
}
```

【実行結果】

```
Name:Frank
Math:90
Physics:83
Name:Thomas
Math:90
Physics:92
Name:Thomas
Math:90
Physics:92
```

例題：

- 下のプログラムを実行した場合の出力はどのようなになるか考えよ。

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

struct student{
    char name[20];
    int math;
    int phys;
};

int main(void){
    struct student a,*pa;
    strcpy(a.name, "Frank");
    a.math = 90;
    a.phys = 83;

    printf("    Name:%s\n",a.name);
    printf("    Math:%d\n",a.math);
    printf(" Physics:%d\n",a.phys);

    pa=&a;
    strcpy(pa->name, "Thomas");
    pa->phys = 92;
```

```
    printf("    Name:%s\n",pa->name);
    printf("    Math:%d\n",pa->math);
    printf(" Physics:%d\n",pa->phys);
    printf("    Name:%s\n",a.name);
    printf("    Math:%d\n",a.math);
    printf(" Physics:%d\n",a.phys);

    return 0;
}
```

【実行結果】

```
    Name:Frank
    Math:90
 Physics:83
    Name:Thomas
    Math:90
 Physics:92
    Name:Thomas
    Math:90
 Physics:92
```

● 構造体を使ったサンプルプログラム

```
#include <stdio.h>

struct student{ /*構造体の宣言*/
    char name[20];
    int  math;
    int  phys;
    double ave;
};

void Average(struct student *std){
/*平均値を計算する関数*/

    int sum;
    sum = std->math + std->phys;
    std->ave = (double) sum/2;
}

int main(void){

    struct student S1 = {"Frank",90,83};

    Average(&S1);

    printf("Name  =%s\n",S1.name);
    printf("Math  =%d\n",S1.math);
    printf("Phys  =%d\n",S1.phys);
    printf("Ave   =%.2f\n",S1.ave);

    return 0;
}
```

【実行結果】

```
Name =Frank
Math =90
Phys =83
Ave  =86.50
```

● 構造体を使ったサンプルプログラム

```
#include <stdio.h>

struct student{ /*構造体の宣言*/

    char name[20];
    int  math;
    int  phys;
    double ave;

};

void Average(struct student *std){
/*平均値を計算する関数*/

    int sum;
    sum = std->math + std->phys;
    std->ave = (double) sum/2;

}

int main(void){

    struct student S1 = {"Frank",90,83};

    Average(&S1);

    printf("Name  =%s\n",S1.name);
    printf("Math  =%d\n",S1.math);
    printf("Phys  =%d\n",S1.phys);
    printf("Ave   =%.2f\n",S1.ave);

    return 0;
}
```

【実行結果】

```
Name =Frank
Math =90
Phys =83
Ave  =86.50
```

構造体を返す関数

- 代入が可能な構造体は、関数の戻り値として使うこともできる.
- 前頁のプログラムを構造体を返す関数として関数 Average に関係する箇所を書き換えれば次のようになる.

```
struct student Average(struct student temp){  
    temp.ave = (double) (temp.math + temp.phys)/2;  
    return temp;  
}
```

```
int main(void){  
    ...  
    S1=Average(S1);  
    ...  
}
```

構造体の配列

- 構造体のメンバ全体を1つの要素とする配列を宣言できる.
 `struct 構造体 配列名 [要素数];`
- 今まで学習した配列や構造体と同様に扱うことができる.

```
struct student Std[20];  
...
```

```
for(i=0;i<N;i++){  
    Std[i].ave = (double)(Std[i].math+Std[i].phys)/2;  
}
```

本日のまとめ

- 構造体の復習と発展
 - 構造体変数の宣言
 - メンバの参照
 - 構造体へのポインタ
 - 構造体を返す関数
 - 構造体配列