

シラバス

21,22. 構造体

構造体による多数のデータをまとめて処理する方法の理解

事前学修：授業スライド「第11回 構造体」と教科書p.171～p.182の熟読。（60分）

事後学修：授業スライドの用語とその意味の理解。構造体
型定義の利点，C言語における構造体型定義と変数の宣
言，構造体変数の初期化と取り扱い，構造体の配列の宣言
と取り扱い方法，関数の引数と戻り値として構造体を使う
方法，typedef宣言を説明できること。（60分）

演習課題のプログラムの完成と提出。（180分）

プログラミングの基礎及び演習

日本大学 工学部 情報工学科

プログラミングの基礎

■ 構造体

- 複数のデータ型の扱い
- 構造体変数
- 構造体の配列
- 構造体と関数
- typedef宣言と構造体

- 教科書 p.171～p.182, p.112 (typedef宣言)

データ型 (型)

■ データの種類のカテゴリ

- 言語で提供されている基本的な型を,
プリミティブ型と呼ぶ
 - 整数(int), 実数(double), 文字(char), ...

■ データ型の役割

- コンパイル時に不適切な演算や代入を認識
 - 型の導入で安全性や最適化が可能

多くのデータ型を扱う手法

- 配列: 同じ型のデータの集まりを扱う

例) `int array[10];`

- **構造体**: 複数の型のデータの集まりを扱う

例) 野球チームの打撃データ

3つの型を含む構造体

名前(char型の配列)	一郎	...	山崎
打率(double型)	0.352	...	0.246
ホームラン数(int型)	11	...	39

構造体

■ 構造体の構成

- タグ: 構造体を区別する名前
- メンバ: 構造体の構成要素

■ 構造体の宣言

- struct タグという構造体(新しい型)ができる

```
struct タグ {  
    データ型 メンバ;  
    ...  
    データ型 メンバ;  
};
```

型の宣言であって
変数の宣言ではない

構造体の例

■ 野球選手のバッティング成績を表す構造体 struct batting 型

□ メンバ

- 名前: char型配列(文字列)
- 打率: double型
- ホームラン数: int型

```
struct batting {  
    char    name[20];  
    double  ave;  
    int     homer;  
};
```

名前

打率

ホームラン数

セミコロン「;」が必要

プログラミングの基礎

■ 構造体

- 複数のデータ型の扱い
 - 構造体変数
 - 構造体の配列
 - 構造体と関数
 - typedef宣言と構造体
-
- 教科書 p.171～p.182, p.112 (typedef宣言)

構造体変数

■ 構造体変数

- 宣言された構造体のメンバを持つ変数

■ 構造体変数の宣言

- struct タグ 型の変数ができる

```
struct タグ 変数名;
```

例: 構造体変数の例

- 構造体: struct batting 型
 - 野球チームの打撃データ
- 構造体変数: struct batting 型の変数
 - 例: 一郎君の打撃データ

名前(char型の配列)	一郎	...	山崎
打率(double型)	0.352	...	0.246
ホームラン数(int型)	11	...	39

3つの型のメンバを持つ構造体の変数

例: 構造体変数の宣言

```
struct batting {  
    char name[20];  
    double ave;  
    int homer;  
};  
  
int main(void) {  
    struct batting first;  
    int i, j, k;  
  
    ..  
  
    return 0;  
}
```

} struct batting 型の宣言
(構造体の宣言)

} struct batting 型の変数
(構造体変数)である
firstの宣言

構造体型変数の宣言は
struct タグ名 変数名;

構造体変数の参照

- 構造体変数のメンバは, ドット演算子で扱う

構造体変数名.メンバ名

ドット演算子

- 例

□ struct batting型の変数firstのメンバに対して

- first.name, first.ave, first.homer により表す

```
struct batting {  
    char name[20];  
    double ave;  
    int homer;  
};  
struct batting first;
```



構造体変数の各メンバへのデータ入力

```
#include <stdio.h>
struct batting {
    char name[20];
    double ave;
    int homer;
};
```

各メンバに値を入力

各メンバの
値の表示

```
int main(void) {
    struct batting first;

    printf("名前:");
    scanf("%s", first.name);
    printf("打率");
    scanf("%lf", &first.ave);
    printf("ホームラン数");
    scanf("%d", &first.homer);

    printf("name: %s¥n", first.name);
    printf("average: %f¥n", first.ave);
    printf("homerun: %d¥n", first.homer);

    return 0;
}
```

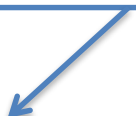
構造体変数の初期化

- 初期化子による代入
 - メンバ順に初期値を記述

```
struct batting {  
    char name[20];  
    double ave;  
    int homer;  
};
```

```
first.name   = "ichiro";  
first.ave    = 0.303;  
first.homer  = 15;  
と同等
```

```
int main(void) {  
    struct batting first = {"ichiro", 0.303, 15};  
    ..  
}
```



プログラミングの基礎

■ 構造体

- 複数のデータ型の扱い
 - 構造体変数
 - 構造体の配列
 - 構造体と関数
 - typedef宣言と構造体
-
- 教科書 p.171～p.182, p.112 (typedef宣言)

構造体の配列

■ 構造体の配列

□ 同じ構造体の, 複数の構造体変数の集まり

- 例:
- 野球チームの打撃データの構造体の,
構造体変数の,
チーム全員の打撃データ

名前(char型の配列)	一郎					山崎
打率(double型)	0.352					0.246
ホームラン数(int型)	11					39

構造体の配列の初期化

■ 初期化子による代入

□ 要素が構造体であるが、配列の初期化と同様

```
struct  batting {  
    char name[20];  
    double ave;  
    int homer;  
};
```

```
int main(void) {  
    struct batting s[] = { {"ichiro", 0.303, 15},  
                           {"jiro", 0.285, 20},  
                           {"saburo", 0.220, 12} };  
    ...  
}
```

s[0]

s[1]

s[2]



構造体の配列の各メンバへのデータ入出力

```
#include <stdio.h>
struct batting {
    char name[20];
    double ave;
    int homer;
};
```

各メンバに値を入力
s[1], s[2]も同様に可能

各メンバの値の表示

```
int main(void) {
    struct batting s[3];

    printf("名前:");
    scanf("%s", s[0].name);
    printf("打率");
    scanf("%lf", &s[0].ave);
    printf("ホームラン数");
    scanf("%d", &s[0].homer);

    printf("name: %s¥n", s[0].name);
    printf("average: %f¥n", s[0].ave);
    printf("home run: %d¥n", s[0].homer);

    return 0;
}
```

構造体の配列の各メンバへのデータ入出力

```
#include <stdio.h>
#define N 3
struct batting {
    char name[20];
    double ave;
    int homer;
};
int main(void) {
    struct batting s[N];
    for (i = 0; i < N; i++) {
        printf("名前:");
        scanf("%s", s[i].name);
        printf("打率");
        scanf("%lf", &s[i].ave);
        printf("ホームラン数");
        scanf("%d", &s[i].homer);
    }
```

プリミティブ型の配列と同様、
繰り返し処理で添字を
利用することが可能

プログラミングの基礎

■ 構造体

- 複数のデータ型の扱い
 - 構造体変数
 - 構造体の配列
 - 構造体と関数
 - typedef宣言と構造体
-
- 教科書 p.171～p.182, p.112 (typedef宣言)



構造体の例題

```
#include <stdio.h>

struct student {
    char name[20];
    double weight;
    int schols;
};
```

```
int main(void) {
    struct student yosida
        ={"Yoshida", 60.5, 10000};

    printf("氏名 : %s¥n", yosida.name);
    printf("体重 : %.1f¥n", yosida.weight);
    printf("奨学金 : %d¥n", yosida.schols);

    yosida.weight -= 1.0;
    yosida.schols -= 2000;

    printf("氏名 : %s¥n", yosida.name);
    printf("体重 : %.1f¥n", yosida.weight);
    printf("奨学金 : %d¥n", yosida.schols);

    return 0;
}
```

実行結果はどうなるか？



構造体の例題

```
#include <stdio.h>

struct student {
    char name[20];
    double weight;
    int schols;
};
```

氏名 : Yoshida
体重 : 60.5
奨学金 : 10000

氏名 : Yoshida
体重 : 59.5
奨学金 : 8000

```
int main(void) {
    struct student yosida
        ={"Yoshida", 60.5, 10000};

    printf("氏名 : %s¥n", yosida.name);
    printf("体重 : %.1f¥n", yosida.weight);
    printf("奨学金 : %d¥n", yosida.schols);

    yosida.weight -= 1.0;
    yosida.schols -= 2000;

    printf("氏名 : %s¥n", yosida.name);
    printf("体重 : %.1f¥n", yosida.weight);
    printf("奨学金 : %d¥n", yosida.schols);

    return 0;
}
```

構造体と関数

- 構造体は、関数に引数として渡せる
 - 引数の型に構造体を記述
- 構造体は関数の戻り値とできる
 - 戻り値の型に構造体を記述

```
struct student {  
    ...  
};  
  
int main(void) {  
    struct student s1, s2;  
    func1(s1);  
    s2 = func2();  
    ...  
}
```

```
void func1(struct student s) {  
    ...  
};  
  
struct student func2() {  
    struct student s;  
    ...  
    return s;  
}
```



関数の引数として構造体を渡す

■ 構造体に対する同じ処理を関数でまとめる

```
#include <stdio.h>

struct student {
    char name[20];
    double weight;
    int schols;
};
```

```
int main(void) {
    struct student yosida
        ={"Yoshida", 60.5, 10000};

    printf("氏名 : %s¥n", yosida.name);
    printf("体重 : %.1f¥n", yosida.weight);
    printf("奨学金 : %d¥n", yosida.schols);

    yosida.weight -= 1.0;
    yosida.schols -= 2000;
    printf("氏名 : %s¥n", yosida.name);
    printf("体重 : %.1f¥n", yosida.weight);
    printf("奨学金 : %d¥n", yosida.schols);

    return 0;
}
```

関数の引数として構造体を使う

■ 引数の型に構造体を記述

プロトタイプ宣言

```
void print_student(struct student gakusei);
```

```
void print_student(struct student gakusei) {  
    printf ("氏名 : %s¥n", gakusei.name);  
    printf("体重 : %.1f¥n", gakusei.weight);  
    printf("奨学金 : %d¥n", gakusei.schols);  
}
```

struct student型の
構造体変数を
引数として受け取る

```
int main(void) {  
    struct student yosida = {"Yoshida", 60.5, 10000};  
    print_student(yosida);  
    ...  
    return 0;  
}
```

struct student型の
構造体変数を関数に渡す



関数の戻り値として構造体を使う

■ 戻り値の型に構造体を記述

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

```
struct enzan {  
    double wa;  
    double sa;  
};
```

```
struct enzan kagen(double x, double y);
```

```
int main(void) {  
    struct enzan s;  
    double x=4.0, y=3.0;  
    s = kagen(x, y);  
    printf("和は%fで, 差は%fです¥n", s.wa, s.sa);  
    return 0;  
}
```

```
struct enzan kagen(double x, double y) {  
    struct enzan v;  
    v.wa = x + y;  
    v.sa = x - y;  
    return v;  
}
```

戻り値は
struct enzan型の変数v

プロトタイプ宣言は
構造体の定義より下に
書く必要がある

struct enzan型の戻り値を
構造体変数sで受けとる

構造体の配列と関数

- 構造体の配列は、関数に引数として渡せる
 - 引数の型に、構造体の配列を記述

✗ 関数は戻り値として配列を返すことはできない



関数の引数として構造体の配列を使う

■ 引数の型に，構造体の配列を記述

```
struct student { char name[20]; double weight; int schols; };
```

```
void print_students(struct student gakusei[]) {
```

```
    printf("氏名 : %s¥n", gakusei[0].name);
```

```
    printf("体重 : %.1f¥n", gakusei[0].weight);
```

```
    printf("奨学金 : %d¥n", gakusei[0].schols);
```

構造体の配列を
すべて受け取る

```
}
```

```
int main(void) {
```

```
    struct student gakusei[2]
```

```
        = {{ "Yoshida", 60.5, 10000 }, { "Yamada", 55.3, 5000 } };
```

構造体の配列
gakusei[]の宣言

```
    print_students(gakusei);
```

```
    return 0;
```

```
}
```

関数に構造体の配列をすべて渡す
(配列名のgakuseiを記述する)

プログラミングの基礎

■ 構造体

- 複数のデータ型の扱い
 - 構造体変数
 - 構造体の配列
 - 構造体と関数
 - typedef宣言と構造体
-
- 教科書 p.171～p.182, p.112 (typedef宣言)

typedef宣言

- 型に対する別の名前(同義語)を宣言
 - プログラムの記述や理解を容易にする
 - ・ 内部表現は元の型と同じ
 - ・ 大文字を使う
 - typedef宣言した型は, プログラム中で使える

- 書式

`typedef 型 型の同義語;`

typedef宣言

■ 例

□ 整数型(int)をWEIGHTとして宣言する

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

```
typedef int WEIGHT;
```

intの同義語としてWEIGHTを宣言する

```
int main(void) {  
    WEIGHT w1, w2, u[10];  
    int num, temp;
```

WEIGHT型の変数w1, w2,
WEIGHT型の配列uを宣言する

```
    w1 = 10;  
    printf("w1 = %d¥n", w1);  
    ...
```

WEIGHTの内部表現はintであるため,
intと同じ処理を行なう

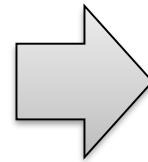
構造体とtypedef宣言

- typedef宣言により, 構造体の名前を短縮
 - プログラムの理解を容易にする
 - タグ名は省略可能

```
#include <stdio.h>

struct student {
    char name[20];
    int number;
};

int main(void) {
    struct student sasaki;
    ...
}
```



```
#include <stdio.h>

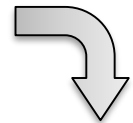
typedef struct {
    char name[20];
    int number;
} STUDENT;

int main(void) {
    STUDENT sasaki;
    ...
}
```

構造体のtypedef宣言と関数

■ typedef宣言した型で置き換えられる

```
struct student { char name[20]; int number; };  
int main(void) {  
    struct student sasaki;  
    ...  
}  
struct student search(struct student s[]) {  
    ...
```



```
typedef struct {char name[20]; int number;} STUDENT;  
int main(void) {  
    STUDENT sasaki;  
    ...  
}  
STUDENT search(STUDENT s[]) {  
    ...
```

【演習課題】

- 演習課題提出システムの「第11回」演習課題を実施
- 演習課題11-1, 11-2, 11-3, 11-4
 - 演習時間に, 設計後に演習環境にリモートログインしプログラムを作成
 - 演習環境でコンパイル, テストし, 完成
 - ソースファイル(11-x.c)をWinSCP等でPCへ転送
 - 課題提出システムへSubmitし各課題100点を目指す
 - 時間内に終了しない場合 ⇒ 宿題
- 授業翌週水曜日の午後5時までに必ず提出
- レポート, 今回はなし