情報システムプログラミングⅡ（2回目）

2024年4月19日（金）

3～4限

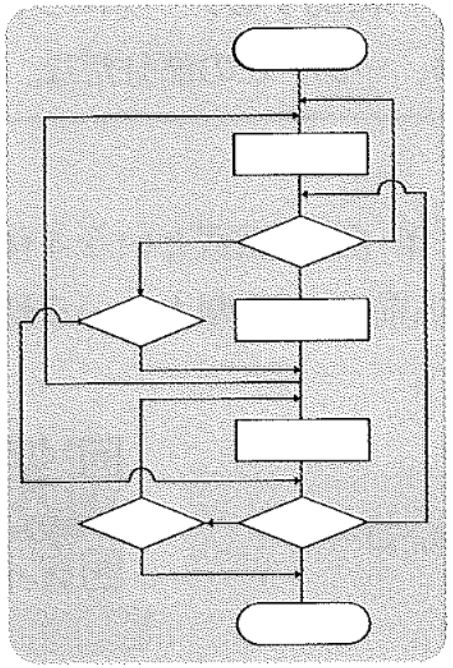
授業内容

- 講義内容（教科書の190～203ページ＋α）
  - 現実的な開発に必要なもの
  - 構造体とは
  - 構造体の使い方
  - 構造体宣言のテクニック
  - scanf関数の利用例
- 演習課題

現実的な開発に必要なもの

■プログラムの複雑性（複雑さ）

- プログラムが大規模になると複雑性が顕著になる
- 様々な構文や型を活用することで対応する

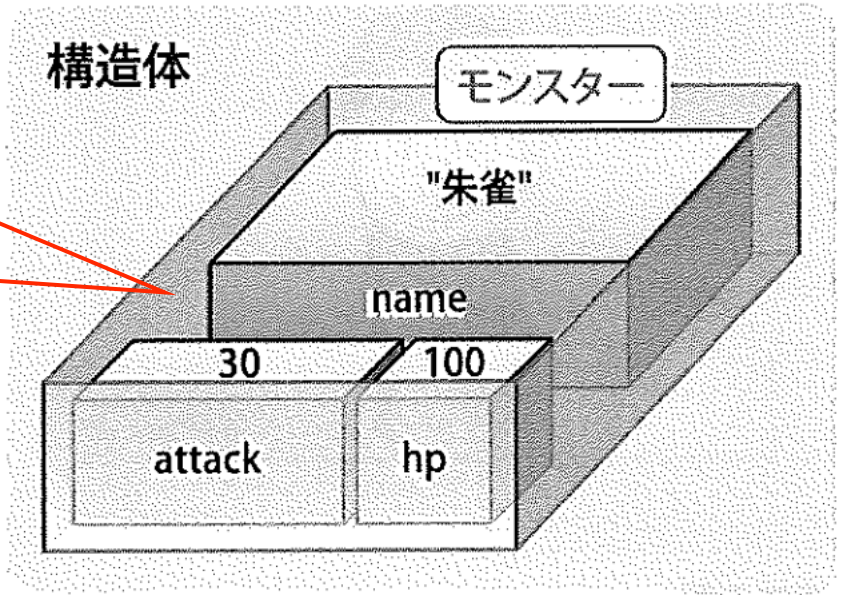


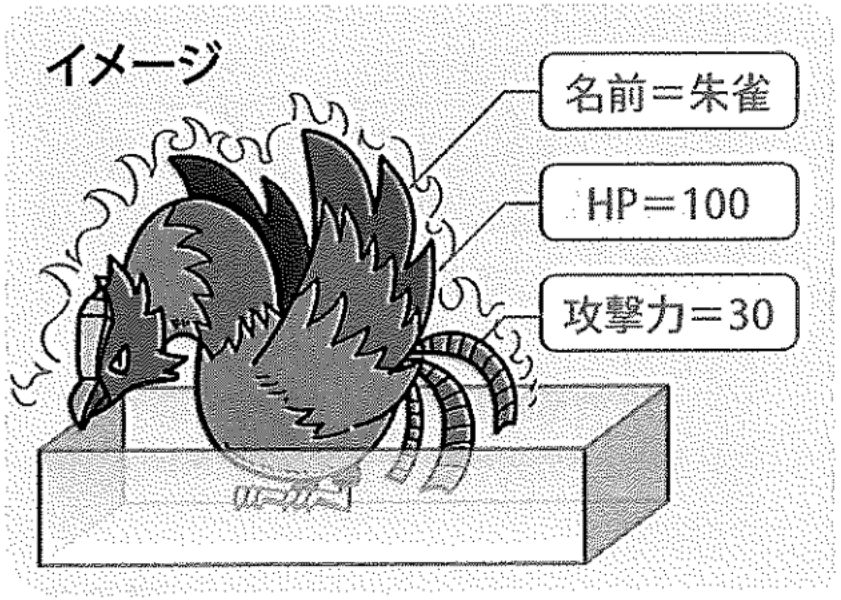
構造体とは

■構造体

- 1つの変数の中に異なるデータを複数格納できる型
- 構造体に含まれる各データはメンバという

メンバとして
「name」，「attack」，
「hp」の各変数がある

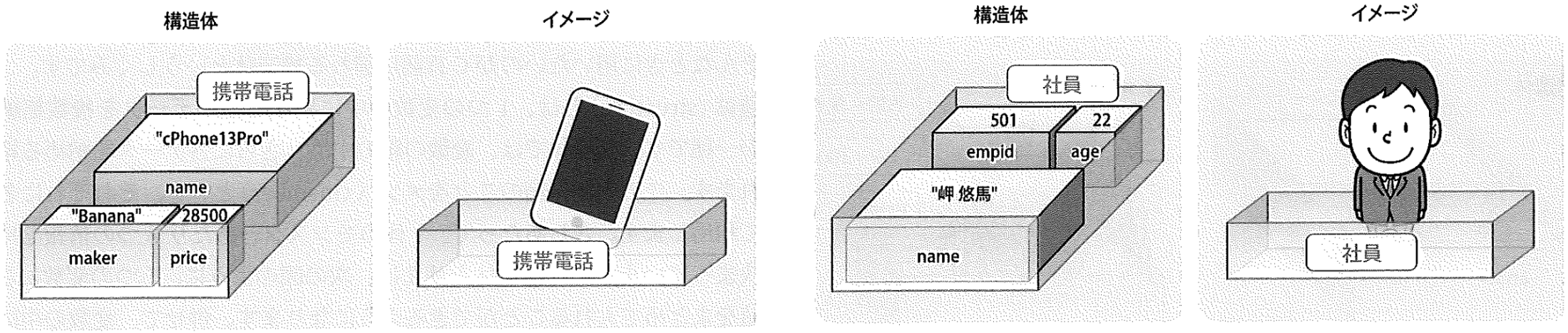




構造体とは

■構造体のイメージ

- ある対象に関連するデータをまとめて管理できる



構造体の使い方

■構造体の宣言

- 構造体を利用するには，構造体の構造を定義した後に変数の宣言（生成）が必要
- （原則的な）構造体を定義するための構文

複数の構造体の
定義を区別する
ための名前がタグ名

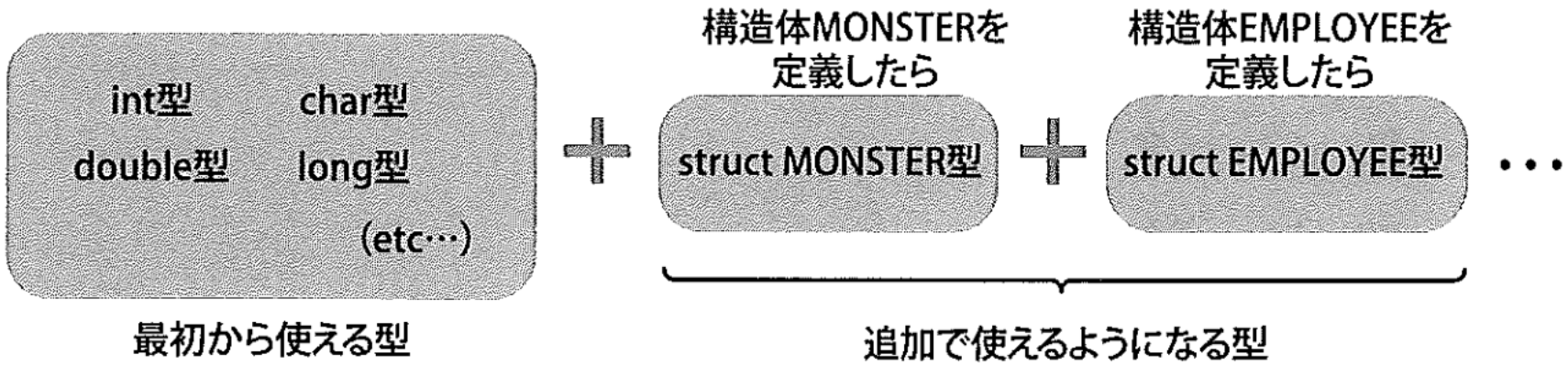
```
struct タグ名 {
    型 メンバ名 1;
    型 メンバ名 2;
      ⋮
};
```

```
struct MONSTER {
    String name;
    int hp;
    int attack;
};
```

構造体の使い方

■構造体の宣言

- 構造体の定義では「struct タグ名」型という名前の新しい型を利用できるようにしている



構造体の使い方

■構造体の宣言

- （原則的な）構造体の変数を宣言するための構文

struct タグ名 変数名；

※「struct タグ名」で１つの型名となる。

定義してから宣言する

```
int main(void)
{
  struct MONSTER {
    String name;
    int hp;
    int attack;
  };

  struct MONSTER seiryu;
  struct MONSTER suzaku;
  struct MONSTER byakko;
  struct MONSTER genbu;

  return 0;
}
```

構造を定めて、MONSTER というタグ名を付ける

「struct MONSTER 型」を使って４つの変数を生成

構造体の使い方

■メンバへのアクセス（メンバの利用）

- メンバを通常の変数のように利用できる
- メンバにアクセスするための構文

もちろんメンバの値を別の変数に代入することもできる
例：int a = suzaku.hp;

```
suzaku.hp = 100;
suzaku.attack = 30;
suzaku.name = "朱雀";
```

これはダメ

文字列に関する型は，
初期化を除いて＝で
代入できないため

構造体宣言のテクニック

■構造体の初期化

- 構造体を宣言する際に初期化（値を代入）できる

```
struct タグ名 変数名 = {
    メンバ１の初期値,
    メンバ２の初期値,
     ⋮
};
```

※初期値は、構造体に定義したメンバの順に記述する。

※メンバの数に対して初期値の数が不足する場合、以降は０で初期化される。

```
struct MONSTER suzaku = {"朱雀", 100, 30};
```

順次指定で初期化

構造体宣言のテクニック

■型に別名を与える

- typedefにより，ある型に対して新しい型名を付与できる
- typedefにより型名を付与するための構文

typedef 型名 型に付ける別名 ;

```
typedef struct MONSTER Monster;
Monster suzaku;
```

struct MONSTER 型に Monster 型という別名を付与

以降、Monster 型を利用可能

宣言時にstructを省略できる（「struct MONSTTER」も引き続き利用できる）

typedefで別名を付与しておかないと「MONSTER」という型が存在しないため

```
MONSTER suzaku;
```

先頭に struct を付け忘れてエラーになる

構造体宣言のテクニック

■構造体の定義時に型名を付与する

- 基本的にはこれを利用すれば良い
- 構造体の定義と型名の付与を
  同時に行うための構文

```
typedef struct {
    型名 メンバ名;
    :
} 構造体型名;
```

```
{
  typedef struct {
    String name;
    int hp;
    int attack;
  } Monster;
```

構造体定義と同時に別名 Monster を付与

```
Monster seiryu = {"青龍", 80, 15};
Monster suzaku = {"朱雀", 100, 30};
Monster byakko = {"白虎", 100, 20};
Monster genbu = {"玄武", 120, 10};
```

構造体変数を宣言すると同時に初期化

```
const String TEMPLATE = "%s : HP=%3d 攻撃力=%2d¥n";
printf(TEMPLATE, seiryu.name, seiryu.hp, seiryu.attack);
printf(TEMPLATE, suzaku.name, suzaku.hp, suzaku.attack);
printf(TEMPLATE, byakko.name, byakko.hp, byakko.attack);
printf(TEMPLATE, genbu.name, genbu.hp, genbu.attack);
```

scanf関数の利用例

■複数の値を同時に（一行で）入力（格納）する

- 変換指定子を半角スペースで区切り連続で記述する

```
int a;
unsigned int b;
double c;
char d;
char e[11];

scanf("%d %u %lf %c %s", &a, &b, &c, &d, e);

printf("a:%d b:%u c:%f d:%c e:%s", a, b, c, d, e);
```

scanf関数の利用例

■1文字の入力（格納）を繰り返す

- 変換指定子（%c）の直前に半角スペースを入れる

直前に半角スペースがないと
正常に動作しない
（入力完了を意味するEnter
キーの入力を1文字として
認識してしまう）

```
char a;

while (a == 'x'){
    scanf(" %c", &a);
    printf("xが入力されるまで繰り返す！¥n")
}
```

- 1文字の入出力にはgetchar関数とputchar関数も利用できる