基礎プログラミング演習 II 教材 (#11)

■ 構造体

教科書 pp.350〜364 参照。

押さえて欲しいポイント：
- typedef で新しい型として宣言できる
- 構造体は複数のメンバ変数を持てる
- メンバ変数には個別にアクセスできる

右例にはありませんが、構造体どうしでの代入ができる事も重要です。

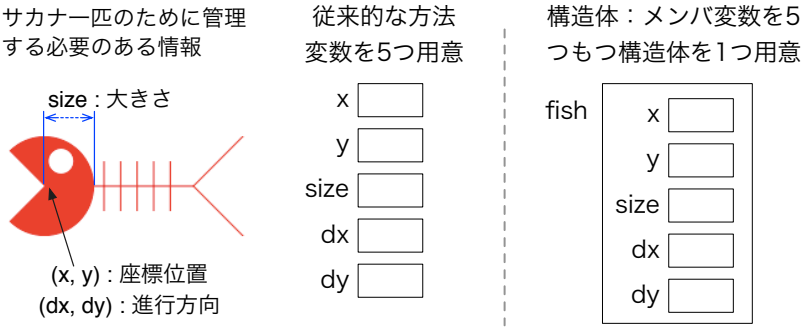
```
typedef struct Car {
  int num;
  double gas;
} Car;

int main() {
  Car car1; // 構造体型の変数（構造体）を宣言
  car1.num = 1234; // メンバに値を代入する
  car1.gas = 25.5;
  printf(" %d  %f\n", car1.num, car1.gas);
}
```

□ 概念的なまとめ

魚が泳ぐアニメーションの例題を思い出して下さい。一匹の魚を表現・操作するために必要な情報は座標位置(x, y)、サイズ、進行方向(dx, dy)の五つで、これをそれぞれ独立した変数として用意しました。

構造体はこれをひとまとめにして扱うものです。五つの要素をもつ、その用途専用の変数を定義するのです。



これを使うと右のようにプログラムとしてシンプルで、分かりやすい書き方ができるようになります。
つまり、例に出した x, y, size, dx, dy 変数群は常に一セットとして扱う必要があり、そのために毎回列挙して書く必要があったところ、それを短く書けるようになったのです。すると、
- 五つの変数を宣言したいのではなく、魚を一匹宣言したいのだ
- 三つの変数を使って描画したいのではなく、この魚を描画したいのだ
- 十個の変数を使って衝突判定がしたいのではなく、二つの魚の衝突判定がしたいのだ
- 五つの変数を一斉に代入したいのではなく、ある魚の情報を別のところに移したいのだ

ということが明確になります。

サカナ一匹のための変数宣言

```
double x, y, size, dx, dy;
        ↓
Fish fish;
```

一匹描画のための関数呼び出し

```
drawFish(x, y, size);
        ↓
drawFish(fish);
```

二匹の衝突判定の関数呼び出し

```
collision(x1, y1, size1, dx1, dy1,
          x2, y2, size2, dx2, dy2);
        ↓
collision(fish1, fish2);
```

あるサカナの情報を別の変数に移す

```
x1 = x2;
y1 = y2;
size1 = size2;   ➡  fish1 = fish2;
dx1 = dx2;
dy1 = dy2;
```

□ 課題 1. メンバ変数を追加する

右に、構造体 Person を使ったプログラムを用意しました。
- 名前・身長(cm)のデータをメンバ変数に持つ
- Person 型の変数（構造体）bob を宣言し、
- 初期値として bob, 170.0 を与えた
- これを printf()で確認する

このプログラムを取得して動作を確認し、
- 体重を意味するメンバ変数を一つ追加して、
- 初期値として適当な値を与え、
- printf()の出力に追加した体重の項目を追加

してください。

```
typedef struct Person {
  char name[32]; // 名前
  double height; // 身長
} Person;

int main() {
  Person bob = {"bob", 170.0};
  printf("%10s %5.1f\n",
             bob.name, bob.height);
}
```

実行結果
```
$ ./a.out
       bob  172.0  65.0
$
```

□ 課題 2. 構造体を引数にとる関数を追加する

課題 1. のプログラムに、BMI の値を求めるよう機能追加してください。
BMI 値は以下のようにして求めます。w は体重（Kg)、t は身長（cm ではなくメートル単位であることに注意）です。

$$\mathrm{BMI} = \frac{w}{t^2}$$

引数として Person 型（構造体）の値を一つ与えると、戻り値として BMI 値（実数）を返す関数を追加して、それを使って結果を出すようにしてください。

正しい計算結果が得られていることは、適当に Google などで検索すれば、BMI 値を求める web サービスがみつけられると思います。それと比較してみると良いでしょう。

構造体型を引数にとる関数の形
```
double getBMI(Person p)
{
    // いろいろ手続き
}
```

呼び出し方の例：
```
  bmi = getBMI(bob);
```

実行結果
```
$ ./a.out
       bob  172.0  65.0 ( 22.5)
$
```

□ 課題 3. 配列を使って複数のデータを扱う

右のプログラムは構造体を配列で使う例です。宣言した際に初期値を設定し、内容を順繰りに出力しています。

このプログラムを取得して動作を確認し、
- 課題 2. で作成した BMI 値計算関数を組み込み、
- BMI 値の最も高いデータを選び出して、
- 最後にそれを出力

してください。

処理方針としては、
- `Person maxPerson;` などとして、ひとり分のデータを保持する Person 構造体を一つ用意し、
- ループによって person[ ]配列を順繰りにチェックして
- `maxPerson` と `person[i]`の `BMI` 値を比較して、より大きなデータが見つかったら、
  `maxPerson = person[i];`として構造体まるごと代入し、
- 最大の `BMI` 値をもつデータを `maxPerson` に残す

ようにしてください。
ただし、そのためには `maxPerson` 構造体は最小の `BMI` を示すデータで初期化しておく必要があることに注意してください。

```
Person person[5]={
  {"Alice", 155.0, 45.0},
  {"Bob",   170.0, 65.0},
  {"Carol", 163.0, 55.0},
  {"Dave",  180.0, 85.0},
  {"Ellen", 160.0, 56.0}
};
int i;
for(i=0; i<5; i++) {
  printf("%10s  %5.1f %5.1f\n",
      person[i].name,
      person[i].height,
      person[i].weight);
}
```

実行結果（機能追加後）
```
$./a.out
     Alice  155.0  45.0 ( 18.7)
       Bob  170.0  65.0 ( 22.5)
     Carol  163.0  55.0 ( 20.7)
      Dave  180.0  85.0 ( 26.2)
     Ellen  160.0  56.0 ( 21.9)
== MAX is
      Dave  180.0  85.0 ( 26.2)
$
```

□ 参考：構造体を使った例

構造体をアニメーションのキャラクター・データ管理のために使った例を用意してみました。(教材 web の pacman1.c)
以前にサンプルとして紹介したサカナが動くアニメーションでは、キャラクター（サカナ）の描画に必要な情報、つまり x, y, size, dx, dy をそれぞれ別の配列変数として持っていました。

```
typedef struct Pacman {
    double x, y;
    double size;
    double dx, dy;
} Pacman
```

しかし構造体を使えば、これら 5 つの変数を一つにまとめて「あるサカナ（キャラクター）が保持しているデータ」として扱うことができます。
例えば右のようになります。

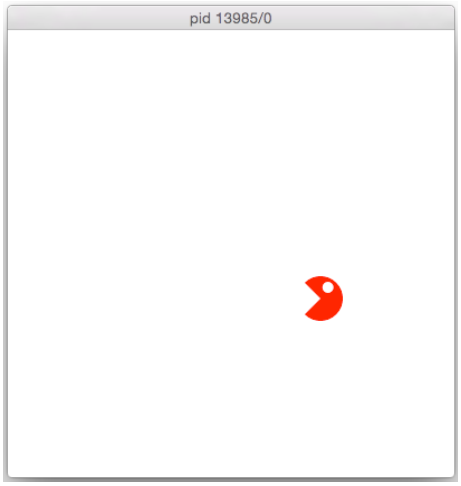
1. Pacman 構造型の pac を用意し、
2. pacSetup( ) でその内容を作成し、
3. アニメーションに必要な描画、移動処理を、とてもシンプルに記述できます。

```
Pacman pac;
// Pacman 構造体を用意する
pac=pacSetup(200.0, 200.0, 20.0, -8.0, 5.0);
while(1) {
    HgClear();          // 画面を消去
    pacDraw(pac);       // パックマンを描画する
    pac=pacMove(pac);   // 移動させる
    HgSleep(0.1);       // 少し待つ
}
```

特にプログラム全体が、pacman キャラクターに関わるデータをひとまとめにして記述できるため、まるで

```
// Pacman 構造体を用意する
pac=pacSetup(200.0, 200.0, 20.0, -8.0, 5.0);
pacDraw(pac);       // パックマンを描画する
pac=pacMove(pac); // 移動させる
```

といった処理が、そのまま読める、つまり処理の概要が直感的に分かりやすくなったことが重要です。



壁に当たると色が変わる機能を追加した pacman2.c プログラムも用意しておきました。追加した機能は main( ) の機能（位置の制御、アニメーション処理）には関係がないので、その部分の変更なしに、つまり上に示した「処理の概要」の分かりやすさを一切損なわずに機能追加ができています。
プログラムが扱う対象の抽象度が少し上がった（x 座標、y 座標を扱うのではなく、魚を扱う、という形になった）ことで、いろいろ良い事がでてくるわけです。

□ （さらに難しい）参考：ポインタとの併用

上の pacman の例では `pac=pacMove(pac);` などとして、

- 構造体を関数に渡して、
- 関数の中で何らかの処理をして、受け取った構造体のデータを更新し、
- return で戻す（そして呼び出し側で元の構造体データに丸ごと代入する）

といったやり方をとっています。

しかしこの方法は内部的な処理効率の問題から余り望ましい方法ではありません。関数は引数の受け取り時に、その内容をコピーします。規模の大きな構造体を引数にすると、まず呼び出し時にそれを丸ごとコピーし、return の際に丸ごと呼び出し側に戻して、その後丸ごと別の変数に代入（つまりコピー）します。

更新が生じるのは構造体が含むデータの一部でしょうから、毎回このような大量コピーを行うのは非効率です。これを避けるために、構造体を関数に渡すときはその実体ではなくポインタを与えることが良く行われます。
教科書 p.370 にもそのような例が挙げられています。もう一歩進みたいひとはトライしてみると良いでしょう。

構造体のような複合データ型と、それを用いて抽象度を高めるアイディアはほとんどすべてのプログラミング言語（高級言語）に取り込まれています。発展プログラミングで扱う Java でも多用されていますから、興味があれば少し深掘りしてやってみることを勧めます。