基礎プログラミング演習 II 教材 (#11)

（この単元は二週分として出します。課題 2, 3 はすぐ出来るでしょうから一週目限定としますが、課題 4, 5 を二週間使って取り組むことを進めます。課題 4, 5 は時間が掛かると思いますから。）

■ 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

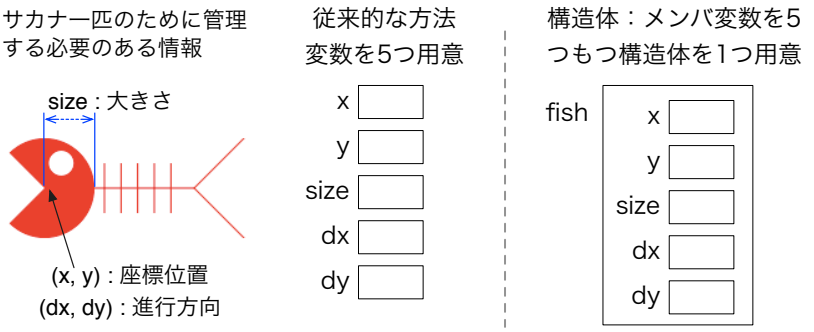
```
typedef struct Car {
  int num;
  double gas;
} Car;

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);
}
```

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。

□ 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(size)、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  170.0  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  170.0  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。なおこの課題では BMI 値が同値になるデータは無いものとして作ってくれて構いません。

```
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);
}
```

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。

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)
$
```

■ ゲームを作る：構造体を使って記述する

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 fish7.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 Fish {
    double x, y;
    double size;
    double dx, dy;
} Fish;
```

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. Fish 構造型の変数 fish を用意し、
2. fishSetup( ) 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。

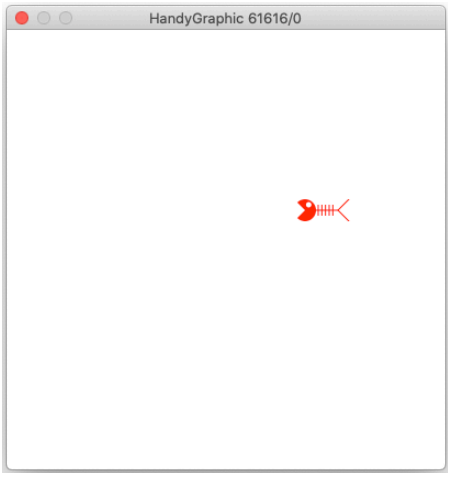
```
Fish fish; // Fish 変数を用意する
fish=fishSetup(200.0, 200.0, 10.0, -8.0, 4.0);
// fish 変数をセットする
while(1) {
    HgClear();            // 画面を消去
    fishDraw(fish);       // 描画する
    fish=fishMove(fish);// 移動する
    HgSleep(0.1);         // 少し待つ
}
```

特にプログラム全体が、fish 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

```
Fish fish;                // fish 変数を用意する
fish=fishSetup(200.0,…… // fish 変数をセットする
fishDraw(fish);           // 描画する
fish=fishMove(fish);      // 移動する
```

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。

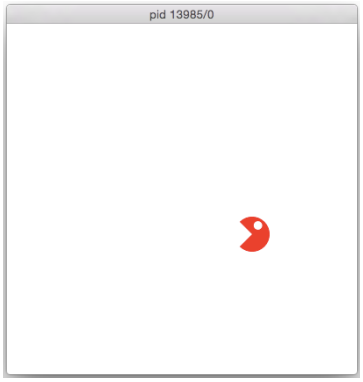
以下に fish7.c プログラムの全体構造を示します。一度プログラムをダウンロードして、動かして、コードを眺めて下さい。これが次の課題の母体になります。



fish7.c プログラムの全体構造

```
typedef struct Fish {
   構造体の定義
} Fish;

Fish fishSetup(double x, double y, double size, double dx, double dy) {
   所定の値をセットした Fish 変数を返す
}

Fish fishMove(Fish fish) {
   移動したあとの Fish 変数を返す
}

void fish_left(Fish fish) {
   左向けのサカナを描く
}

void fish_right(Fish fish) {
   右向けのサカナを描く
}

int main() {
    Fish fish; // Fish 変数を用意する

    HgOpen(400.0, 400.0);

    fish=fishSetup(200.0, 200.0, 10.0, -8.0, 4.0); // fish 変数をセットする
    while(1) {
        HgClear();          // 画面を消去
        if(fish.dx < 0.0) {
            fish_left(fish);  // 左向きの魚を描く
        } else {
            fish_right(fish); // 右向きの魚を描く
        }
        if( fish.y < fish.size || fish.y > (400.0 - fish.size) ) fish.dy
        if( fish.x < fish.size || fish.x > (400.0 - fish.size) ) fish.dx
        fish=fishMove(fish); // 移動
        HgSleep(0.1); // 少し待つ
    }

}
```

□ 動きまわるキャラクター



これから、右図のようなキャラクターを操作して動き回ることができるゲームを作りましょう。どのような動きの、どんなゲームを想定しているか分かるように動画を用意してありますので、確認すると良いでしょう。

まず、ただこのキャラクターをキー操作によって動かすだけの例を用意してみました。(教材 web の pacman01_base.c)
つまり方向の操作として、I, J, K, L キーを押せば、それぞれ上、左、下、右に方向を変えて進み続けるようになります。壁に衝突すると、このプログラムは終了します。

pacman01_base.c プログラムの全体構造を右に示します。実際のコードと見比べて、どんな動き方をしているか、理解してください。

```
int main() {
    初期設定処理
    x = WIN_SIZE / 4.0; y = WIN_SIZE / 2.0;
    dx = PAC_STEP; dy = 0.0; // 右に移動
    while(1) {
        HgClear();          // 画面を消去
        hgevent *event = HgEventNonBlocking();
        if(event != NULL &&
            ......          キー入力処理
        }
        移動に関する処理
        描画に関する処理
        if( 衝突判定に関する処理 == 1 ) break;
        HgSleep(0.05);      // 少し待つ
    }
    return 0;
}
```

HgSetEventMask()関数、HgEventNonBlocking()関数、つまりキー入力処理とイベント処理のことは理解できなくてかまいません。
このような書き方をすると、キー入力が無かった時はキー入力処理のなかの if 文は素通りする (条件が偽になる) ことだけ把握してください。

□ 課題 4. レベル 1. 構造体対応・関数化

この pacman01_base.c プログラムを、構造体に対応させて、キー入力処理・移動処理・描画処理・衝突判定処理などを関数化してください。全体構造は前のページの fish7.c と同じにすれば良いでしょう。

そのために、まずキャラクターの構造体を作ります。例えば次のように作れば良いでしょう。

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

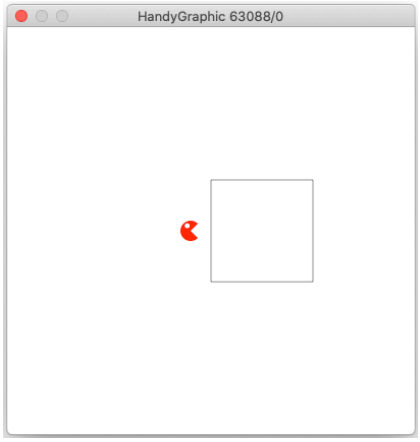
この構造体を使って、前のページで説明した fish7.c 同様の関数の呼び出し構造にすれば、このキャラクターが動き回るだけのプログラムが作れると思います。例えば右のような状態です。
このような形で作って、「課題 4.」として提出してください。

```
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
    pac=pacKeyIn(pac); // き一入力に反応する
    pac=pacMove(pac);  // 移動させる
    pacDraw(pac);       // 描画する
    if( hitWall(pac) == 1 ) break; // 衝突判定
    HgSleep(0.1);       // 少し待つ
}
```

■ 課題 5. できるところまで、ゲームっぽい動作を作り込む

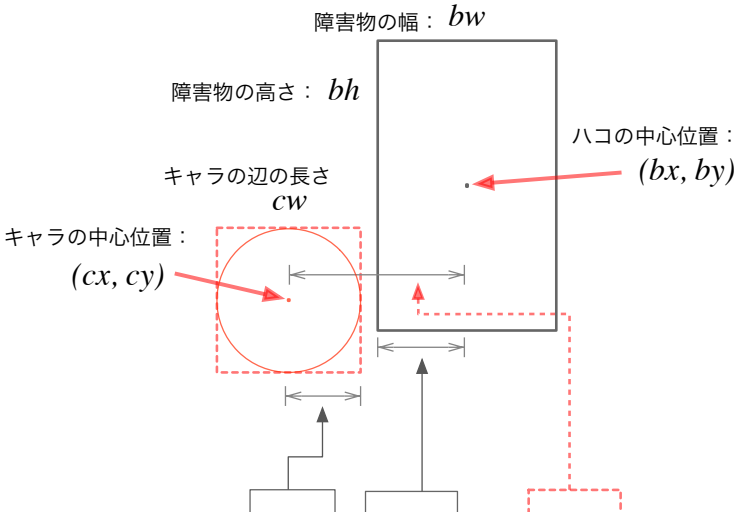
授業として全員に同じ内容で提出を求める課題としてはこれが最後になります。そこで受講生の実力を見るために、何段階かの目標レベルを示しますから、できるところまでやって、それを「課題 5」として提出してください。（どのレベルまで出来たか、プログラムの冒頭に level 3 などと書いておいて貰えると助かります。）

□ レベル 2. 障害物を出す



障害物を出すことを考えましょう。右図のように、画面のどこかに障害物となるハコ（block）を置き、壁だけで無くこれに衝突しても終了するようにします。

キャラクターを円だと考えると、四角形と円の衝突はそれなりに判定が難しいので、ここでは簡単さのために「キャラクターも四角だ」と考えることにして判定処理を検討しましょう。



(1/2 cw + 1/2 bw) < (bx - cx)

横方向について衝突**なし**と言える条件：
“キャラの幅1/2 と障害物の幅1/2 を合わせても、
障害物とキャラの中心間距離に届かない”

障害物が左にある時は bx - cx が負になる → 絶対値で比較

(1/2 cw + 1/2 bw) < | bx - cx |

左に、ざっと判定アルゴリズムを図示してみます。

考え方としては、

- X 方向、Y 方向、それぞれで衝突している位置関係にあるかどうか調べ、
- X,Y のどちらかでも「衝突」して「いない」場合は「衝突なし」としよう
- キャラクターの中心位置と、障害物であるハコの中心位置の X 距離が、キャラクターの幅半分と、ハコの幅半分の合計より大きい＝衝突なし
- Y 方向でも同様にチェックすれば良い

計算式としては、左に書いたとおりです。キャラクターとハコの位置関係が左右逆の（ハコが左でキャラが右にある）場合は、ハコとキャラの中心位置の距離を求める bx − cx の値が負になってしまうので、そこは絶対値を取るところがポイントです。

絶対値は数学関数の fabs() で求められます。

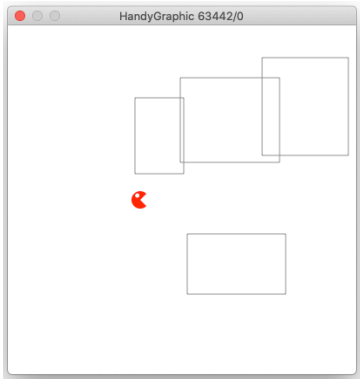
□ レベル 3. 障害物の位置・大きさをランダムで出すようにする

障害物の位置が毎度同じではゲームっぽくなりません。ランダムな位置にランダムな大きさで出すようにしましょう。しかし、ただランダムな位置にランダムな大きさで出してしまうと、ゲームが始まった時点でキャラクターと障害物の位置が重なってしまいます。これではやはりゲームにならないので、それを避けるようにして下さい。

ただし「キャラクターは左から出るから、障害物は右半分のどこかに出ることにする」といった逃げ手はダメです。キャラクターの初期位置を固定位置として、障害物の位置をランダムに出してみたところ、初期的に衝突している場合は「もう一度ランダムな値を出して」やり直す、といった方法で回避してください。

なお、乱数については単元#10「関数」で、サカナをランダムな位置に出すために使いました。使い方はその教材を見ると良いでしょう。

□ レベル 4. 障害物を構造体で取る



ブロックは位置（x, y）と幅・高さの情報をもっていますから、これをやはり構造体で取ってしまうのが良いです。

□ レベル 5. 障害物を複数出す

右図のように、障害物を複数出しましょう。

□ レベル 6. ポイントを付ける

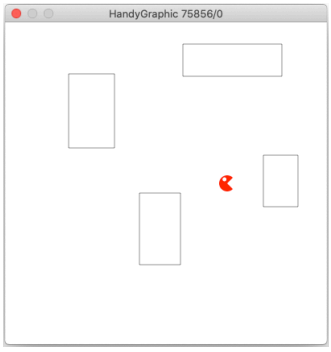
障害物をよけながら走り続けた距離を、ゲームのポイントとして得るようにしましょう。ただし、それだけでは同じ場所で行ったり来たりする人が出ますから、「曲がったら X ポイント減らす」といった方法をとって、できるだけ長い距離を走り回ったことが高得点につながるような工夫をして下さい。例えば「現在の進行方向から左あるいは右に曲がったら X ポイント減点、引き返したらその 10 倍減点」など良いですね。

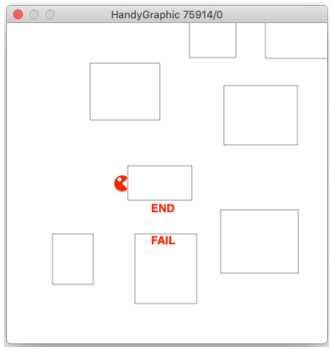
□ レベル 7. 自由な拡張を加える

なんでも良いので、何かしら工夫を加えて下さい。他のゲーム要素を加えるも良し。レイヤーを使って高速化するも良し。何かきれいな映像効果を入れるも良し。(映像効果などを入れたければレイヤーを使わざるを得ないと思いますが、、)

ただ、この授業の目的は「アルゴリズムとデータ構造を検討し、目的とする処理を実現できる」ことですから、単に絵柄に凝ったりしても何も評価されません。コードがきれいになる（シンプルで合理的なものになる)、複雑なことを短いコードで実現する、といったことが重要です。

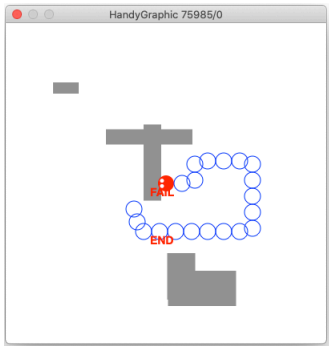
たとえば障害物ではなく迷路のようなものを出すのも良いですが、それを実現するために HgLine() 関数を 100 行書いて壁の線を引き、if 文を 400 行書いて壁との衝突をチェックする、などと「分かってない」事がありありと分かるようなコードを出さないよう注意してください。
逆に迷路のデータを工夫して配列に収め、その衝突判定をきれいにループ処理の中に収めたりすると、もちろん評価は高くなります。





例えば右図は、配置する障害物どうしが衝突せず、かつ、キャラクターが通り抜けられるだけの間隔を必ずあけるように配置させた例です。これをちゃんと書くだけでもかなり良いですね。

障害物をテトリスのような形にして、きちんと衝突検知できるようにする、などなど、できることはたくさんあります。



もっと思い切りゲームっぽく改造しても良いでしょう。右に「動く時間に応じてシッポが伸びていく」ようにしたものを出しておきます。動画も置いてあります。このあたりまではちょっと頑張りすぎな感もありますが、この科目の範囲内で普通にこのくらいはできるようになる、という参考として。
（これでも空行、コメントなど含めて 250 行程度です。）

以上、できるところまでやって、それを「課題 5」として提出してください。
（どのレベルまで出来たか、プログラムの冒頭に level 3 などと書いておいて貰えると助かります。）

■ 参考：ポインタとの併用

今回の 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。

（今回の課題には含めない方が良いと思います。壊してしまいそうですから。）