基礎プログラミング演習 II 教材 (#10)

■ 関数プロトタイプ

教科書 pp.261〜263 参照。

□ 参考：ライブラリ関数の関数プロトタイプ

C 言語でプログラムを開発する場合には入出力関数や数学関数など様々なライブラリ関数を用いますが、それらについても当然ながら型についての情報をどこかから得る必要があります。

ライブラリ関数の関数プロトタイプはまとめて書かれたものが用意されています。よく使っている#include <stdio.h> などのインクルード命令は、これをソースファイルに取り込むものです。ライブラリ関数を使用する前にプロトタイプが必要なので、通常はソースファイルの先頭などでインクルードします。

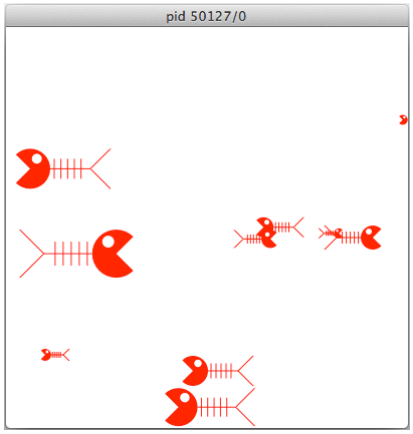
例えば man sin でライブラリ関数 sin のマニュアルをみると、戻り値や引数の型とともに、使用すべきインクルードの対象ファイル（この場合は#include <math.h>）が書かれています。

```
SYNOPSIS
        #include <math.h>
        double
        sin(double x);
```

> ライブラリ関数の使用は、プログラムをいくつかの部分に分けて開発する手法（分割コンパイル）の一種に当たりますが、ここでは説明しません。

include 対象となるプロトタイプ等が書かれたファイルをヘッダファイルと呼びます。拡張子は通常 .h です。

■ 課題 1. 配列を使ってサカナを複数出す



前回の課題 2.を修正し、右図のように複数のサカナが泳ぐようにしてください。全体的な処理方針を示します。

- サカナの制御と描画に必要な変数は x, y, dx, dy, size だった
- これらを配列を用いて必要な匹数ぶん確保し、
- 一コマぶんの描画をするたびに全ての要素について順繰りに移動・描画の処理をする

つまり main() 関数内で匹数ぶんループし、サカナを描画する関数には配列そのもの（x など）ではなくひとつの要素（x[i]など）を渡せば良いわけです。サカナを描画する関数はそのまま再利用できることが分かるでしょう。

ところで「全てのサカナが同じところから同じ方向に出発」するわけにはいきませんから、各サカナの初期的な位置・進行方向（と速度）を乱数によって決める必要があります。

□ 乱数

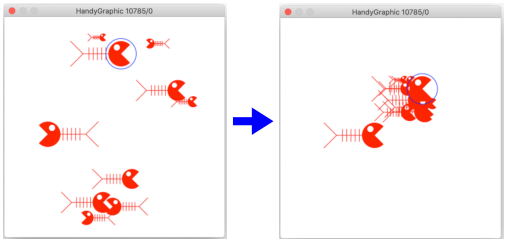
```
int i, number;
srand(time(NULL));  // 乱数列の初期化
for (i = 0; i < 10; i++) {
  number = rand(); // 乱数を得る
  printf("%d\n", number);
}
```

基礎プロ I の「高度なテクニック集」の冒頭に乱数を使うサンプルがあります。参考までにランダムな数を 10 個出すプログラムを右に示します。

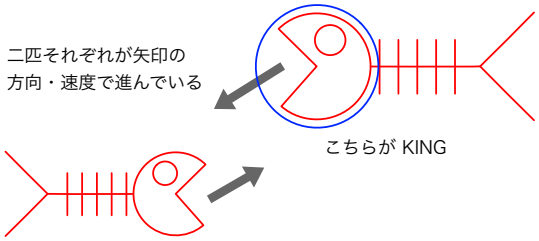
- はじめに一度だけ srand() 関数を実行し、
- その後 rand() 関数を繰り返して呼び出すとその度にランダムな数が得られる。
- srand()関数に実行のたびに異なる値を与える必要があるため、現在時刻を返す time(NULL)を用いた
- rand()関数の戻り値は int 型範囲の正の整数全範囲から得られる（だいたい巨大な数が返ってくる）
- 限られた範囲（例えば 0〜99）でランダムな値が欲しい場合は rand()%100 などとすれば良い
- srand(), rand() のために stdlib.h、time() のために time.h のインクルードが必要

なお基礎プロ I と教科書に合わせて srand(), rand()を例示しましたが、同じ使い方ができる srandom(), random()関数の方がより質の良い乱数を生成するため、こちらの利用が推奨されています。

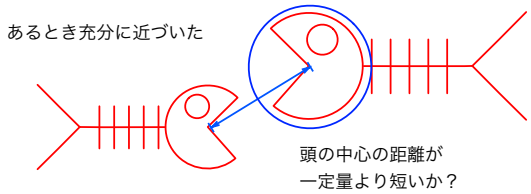
■ 課題 2. 群れになって泳ぐサカナ



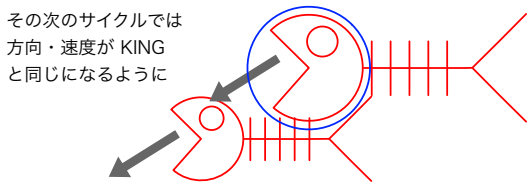
先の課題 1. に少し工夫して、徐々に群れをなして泳ぐようになることを目標にします。つまり初めはバラバラの位置、向きで泳ぎだしたけれど、時間が経つにつれて徐々に一つの群れとして泳ぐようにするのです。
群れを作らせる方法はいろいろあると思いますが、今回は一匹を特別扱いして(KING と呼ぶことにします)、その周りに集まるようにします。KING は目立つように青丸をつけておきましょう。



KING と普通のサカナが泳いでるとします。
KING は左下、もう一匹は右上に進むようにそれぞれの dx, dy がセットされています。
この時点では二匹のサカナは離れています。



あるとき二匹が充分に近づいたとします。
「充分かどうか」の判定は、左図の青矢印、つまり二匹の頭の中心の座標位置（x, y）の距離が、ある値、例えば 50.0 より短いかどうか、で良いでしょう。



近づいたと判定したら、二匹の方向と速度を揃えてしまいましょう。そうすれば次のサイクル以降は同じ方向に向けて進む、つまり群れて泳ぐようになるでしょう。
どう方向を揃えると本物っぽく見えるかは考えどころですが、今回は KING に揃えることにしましょう。

いきなり課題 1. のコードをいじるのではなく、以下の手順で作業を進めると良いでしょう。

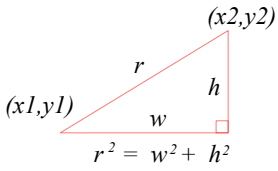
1. 配列（x, y, dx, dy, size）の 0 番要素を KING として、KING の描画時にだけ目印（頭に青丸）をつける
2. 近づいたかどうかを判定する関数を作る
   （その関数のアルゴリズム（手順）を明らかにして関数の仕様、つまり引数、戻り値などを決める）
3. 10 匹の描画が終わった後に、KING と 1〜9 番要素のサカナとの距離を調べる処理を加える
   （近づいたらその相手の要素番号を printf() するだけで良い）

ここまで正しく動作していることが確認できたら、

4. printf() するのではなく、相手のサカナの移動方向を KING のものに書き換える

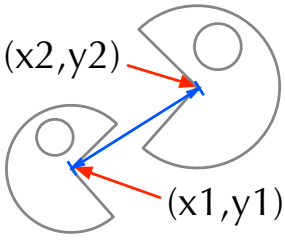
と良いでしょう。

□ 距離を調べる方法



水平方向と垂直方向の距離が分かっている二点間の直線距離を得るには、ピタゴラスの定理（右図参照, $r^2 = w^2 + h^2$）がそのまま使えます。

つまり、二つのサカナの頭の座標情報、たとえば (x1, y1), (x2, y2) から、上の三角形における *w, h* の値を求められます。そこから r が得られるはずです。



なお、二点の距離を得るために便利な hypot 関数もあります。数学関数として用意されています。

■ 配列を引き数に与える

教科書 pp.309〜311（の図 10-5 まで。ポインタは扱わない。）

注意：配列を引き数で渡した場合、関数内で値を修正すると呼び出し元でも反映されます。これは値渡しにならず、参照渡しになるからですが、その説明はちゃんとやりません。そういうものだと思って下さい。
（説明にはポインタの理解が必要で、基礎プロ II ではポインタを扱わないのです）

□ 課題 3. 最大の数を求める関数（#4 教材の課題 2. から）

```
$  echo 22 80 57 60 50 | ./a.out
max : 80
$
```

5 つの数を入力して配列に保存し、次にその内容を調べて最も大きな数を出力するプログラムがありました。
このプログラムの最大値を調べる処理を関数化してください。
具体的には以下のような仕様の関数を作って、それを使うように書き直してください。

- 引き数：長さ 5 の一次元配列
- 戻り値：最大値

□ 二次元配列を引き数に渡す

```
void sub( int matrix[][] )
{
    ……
}
```

ところで二次元配列を同様に引き数で受け取ろうとすると、右のような記述になるでしょう。しかしこのような書き方をすると以下のようなコンパイルエラーが出ます。

```
$ cc array2d_arg.c
array2d_arg.c:5:21: error: array has incomplete element type 'int []'
void sub( int matrix[][] )
                    ^
1 error generated.
$
```

詳細は説明しませんが、二次元配列を引き数にとるには、要素数（右例の 10 がそれ）を明示する必要があります。
なお、前方（一番左）の要素数は書かなくても大丈夫です。
（右例であれば matrix[][10] となる。更に意味不明でしょうが、ここでは説明しません。）

```
void sub( int matrix[10][10] )
{
```

□ 課題 4. 魔方陣判定関数

8	1	6
3	5	7
4	9	2

魔方陣とは右図のように、1 から始めて全て異なる整数を正方のマス目に並べ、縦横斜めの合計が全て同じ値になる組み合わせを言います。3x3 の魔方陣は右に示す形と、その対照形しかありません。

右に示すように 9 つの数値を入力し、3x3 に配置したときにその並びが魔方陣を成すかどうか判定するプログラムを作って下さい。
（入力が 1〜9 の数で重複がない事は仮定して構いません）

```
$ echo 1 2 3 4 5 6 7 8 9 | ./a.out
Not Magic Square
$ echo 8 1 6 3 5 7 4 9 2 | ./a.out
Magic Square
$
```

以下のような仕様の関数を作って実現してください。
・引き数：3x3 の二次元配列
・戻り値：判定結果（0 なら魔方陣不成立, 1 なら成立）

```
Magic Square の例
  8 3 4 1 5 9 6 7 2
Not Magic Square の例
  5 3 7 1 8 6 9 4 2
  6 5 7 1 9 8 2 4 3
  9 2 7 1 3 5 8 4 6
  9 7 8 1 2 3 5 6 4
  6 4 5 7 8 9 2 3 1
  9 4 2 7 5 3 8 6 1
  7 6 8 5 4 3 9 2 1
```

注意：

- うまく作るとプログラムコードはかなり簡素化できます。時間があれば（後からでも）挑戦してください。
- 列の合計は 15 になりますが、if(sum == 15) のようにハードコードして判定してはいけません。「縦横斜めの合計が全て同じ値になるかどうか」で判定してください。
- 右の判定例を試してすべてその通りになることを確認してください。