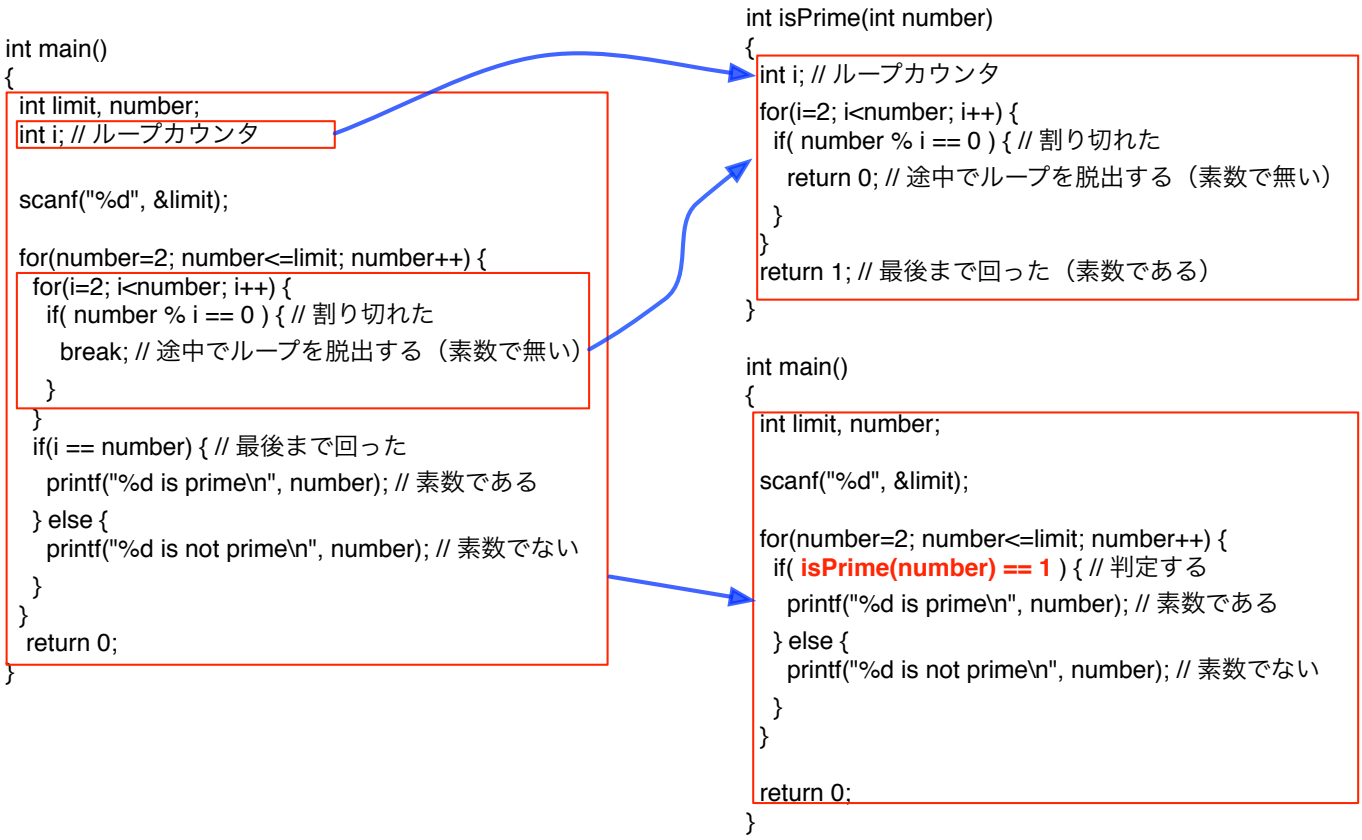
基礎プログラミング演習 II 教材（#9）

■ 変数とスコープ

教科書 pp.249〜255 参照。

□ 局所化できると何が嬉しいか（ローカル変数の何が良いか）

前回の課題 8. 素数判定関数を作るところでは、素数判定関数の中でまさに（ループカウンタ変数のために）ローカル変数を使うことになったと思います。
（以下はサンプル。受講生が実際に作ったプログラムとは異なるかもしれませんが。）

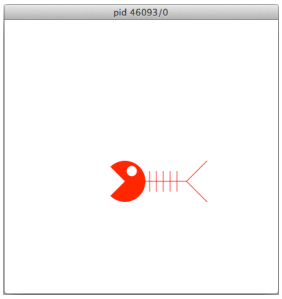


ローカル変数を使うことによって、関数として外に出した処理に「だけ」必要な変数を呼び出し元の main()処理から見えなくすることができました。つまり呼び出す関数の中でどんな変数がどのように使われているのか、呼び出す側は関知しなくて良くなったわけです。
つまり関数をひとつの「機能部品」として考え、設計し、書くことになります。

プログラムの規模が大きくなっていくにつれて、このことが重要になります。
次にプログラムの行数が増していく中で、プログラムを機能単位で分割して書いていく例を示します。

■ プログラムの分割

右図のようにサカナが壁に跳ね返りながら泳ぐプログラム fish.c があります。取得して、実行し、動きとプログラムを見比べて、内容を理解して下さい。



関数を用いてプログラムを書く価値の一つは、処理をまとまった単位ごとに分割して、全体の構造を解りやすく記述できることです。下図、左はサンプルのプログラム、つまりすべての処理をまとめて書いたものです。このうちサカナを描画する部分を抜き出して関数化した場合のプログラム全体の構造を右側に示しています。こうすることで主たる制御処理の構造がよりわかりやすく記述できています。
（更に描画処理が長くなった場合のことも想像してください。）

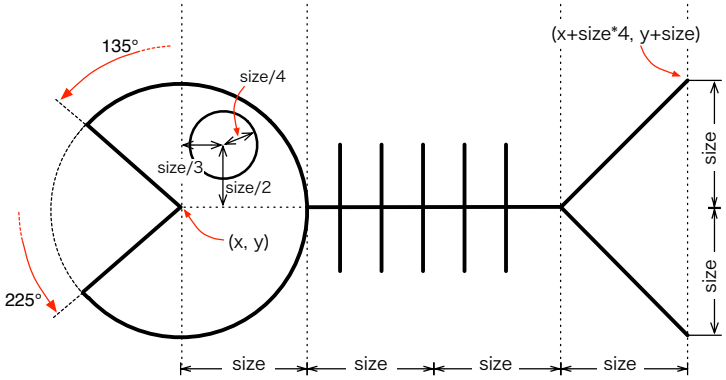
```
#include <stdio.h>
#include <math.h>
#include <handy.h>

int main() {

    int i;
    double x, y, size, dx, dy;

    HgOpen(400.0, 400.0);

    x=200.0; y=200.0; // x, y 座標の初期設定
    size=10.0;        // アタマの半径を決める
    dx=-0.9 * size;   // x 方向の移動速度
    dy=0.5 * size;    // y 方向の移動速度

    while(1) {
        HgClear();              // 画面を消去

        .................
        ........
        ............
        ......................
        ........................

        if( y + dy < size || y + dy > (400.0 - size) ) dy*=(-1.0);
        y+= dy;
        if( x + dx < size || x + dx > (400.0 - size) ) dx*=(-1.0);
        x+= dx;
        HgSleep(0.1); // 少し待つ

    }
}
```

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

void fish(double x, double y, double size)
{
    .................
    ........
    ............
    ......................
    ........................
}

int main() {

    double x, y, size, dx, dy;

    HgOpen(400.0, 400.0);

    x=200.0; y=200.0; // x, y 座標の初期設定
    size=10.0;        // アタマの半径を決める
    dx=-0.9 * size;   // x 方向の移動速度
    dy=0.5 * size;    // y 方向の移動速度

    while(1) {
        HgClear();              // 画面を消去
        fish(x, y, size);       // サカナを描く

        if( y + dy < size || y + dy > (400.0 - size) ) dy*=(-1.0);
        y+= dy;
        if( x + dx < size || x + dx > (400.0 - size) ) dx*=(-1.0);
        x+= dx;
        HgSleep(0.1); // 少し待つ

    }
}
```

描画処理

□ 課題 1. サカナの描画を関数として独立させる

サンプルのプログラムを修正して、魚を描画する処理を関数（上図の例の fish( ) 関数）として独立させてください。この関数には引き数として描画する座標位置やサイズを渡せば良いでしょう。戻り値は必要ありませんね。



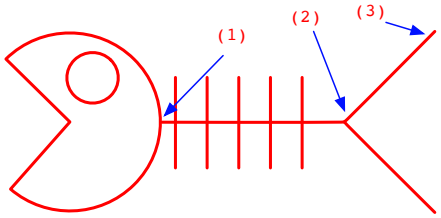
□ 線を引く関数

サンプルプログラムでは線を引く方法を 2 タイプ使ってみました。

・HgMoveTo(), HgLineTo() の組み合わせ
HgMoveTo() によって起点をセットしたあと HgLineTo() を呼ぶと、起点から指定した座標まで線を引きます。
もう一度続けて HgLineTo() を呼ぶと、いま線を引いたところを起点として、指定した座標まで線を引きます。

```
HgMoveTo(x+size, y); // 胴骨 (1)
HgLineTo(x+size*3.0, y); // 胴骨 (2)
HgLineTo(x+size*4.0, y-size); // 尾びれ (3)
```

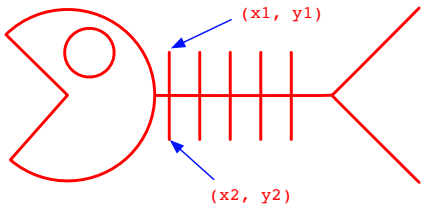
と書いた場合、
(1) を起点としてセットし、そこから
(2) まで線を引き、今度はそこから
(3) まで線を引く、といった動作をします。



・HgLine()
二組の座標点を指定して HgLine() を呼び出すと、その座標点の間に線を引きます。

```
HgLine(x1, y1, x2, y2);
```

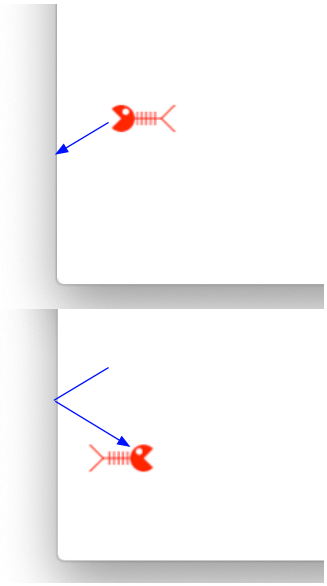
と書いた場合、
(x1, y1) から (x2, y2) まで線を引きます。



□ 課題 2. 右向けの絵も作る

課題 1. のプログラムは右に進むときでも魚が左向きになっています。これを修正して、魚が正しい方向を向いて進むようにしてください。

つまり横の壁に当たって進行方向が逆転したら、描く魚を右向き（あるいは左向き）にするのです。



方針としては、
・課題 1.で「左向きの魚を描く」関数が出来たでしょうから、
・今度は「右向きの魚を描く」関数を作成し、
・「左右の進行方向に合わせて」どちらの関数を呼び出すかを変える
でどうでしょう。