補助資料 #09

■ 総論：始める前に

今回の「関数」、特に変数のスコープの概念は、C に限らずほぼすべてのプログラミング言語で登場する重要な部分です。対面授業では少しずつ、手を動かしながら、教室での受講生の様子（理解度・進み具合）を見ながら、適切なタイミングで白板の上に図を描く、デモをするなどして進みます。遠隔授業ではこれができないので、あなた方自身が、どれだけ焦らず、自分をコントロールして、じっくり理解を進めていけるか、が鍵です。こればかりは教員にはどうすることもできませんので、ホント、しっかり、手を抜かず進めて下さいね。

以下に教材の各段階ごとに、教室で説明する時に押さえているポイントなどについてコメントします。この補助資料と、元の教材を並列に見ながら少しずつ作業するようにしてください。

■ 作業の進め方

以下に教材の各段階ごとに、教室で説明する時に押さえているポイントなどについてコメントします。この補助資料と、元の教材を並列に見ながら少しずつ作業するようにしてください。

■ 変数とスコープ

ここの教科書参照部分は本当にまともに読んで、なんだったら教科書にあるコードを実際に入力して動作を確認しながら進めるのが良いです。p.250 の /* */ でコメントアウトされいてる printf() 文などが、コメント指示を外したらどうなるか（どんなコンパイルエラーが出るか）見るなどすると良いと思います。p.251 の図や、p.252 の説明を「なるほどそうなってるのね」と納得する（目的と仕組みが頭の中でつながる）ことがとても重要です。（目的についてはこのすぐ下を参照）

☐ 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)

ここに、ローカル変数を使う目的を書いてみました。教室ではかなり補足説明をするため、教材ではかなり簡単に書いてあります。（読んで下さい）

以下、そこを読んだあとの人向けに補足します。

もしすべてがグローバル変数だった場合、例えばあなたがこれから作る少し大きなプログラムが、全部で 3000 行あり、関数が 50 個ほど含んでいるとして、その中で使う変数（例えば 500 個）について、すべて名前が衝突しないように付けることなんて出来ないでしょう。また、あなたが誰かから「出来の良い関数」を部品として貰ってきても、そこで使われている変数が自分がこれまで使っていた変数と「全部かぶっていない」ことを確認しなければいけない、となると、作業工数が爆発してしまいます。

「その関数が正しく動作することが分かっておれば、そのまま自分のプログラムに含めても同じように動作する（挙動が変わる、誤動作するようなことはない)」ことが重要です。つまり関数は「機能部品」として独立性を高める必要があるのです。局所変数はそのために必要なアイディアなのです。

■ プログラムの分割

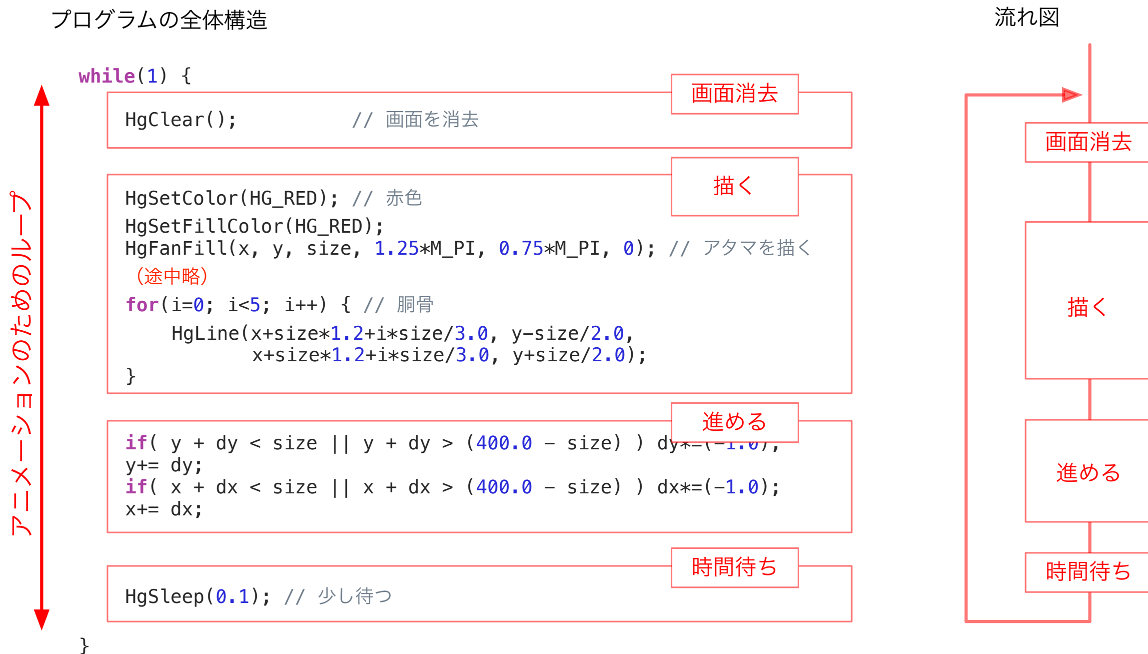
fish.c プログラムの解説として、次ページ以降に説明します。

☐ 課題 1. サカナの描画を関数として独立させる
☐ 課題 2. 右向けの絵も作る

これも課題の意図、として次ページ以降に説明します。

■ fish.c プログラムの解説（「プログラムの分割」部分を読んでからここを読むように）

fish.c プログラムをまず良く見て下さい。以下のような全体構造、動作になっていることが分かるでしょうか。幾つかのブロックに分けて把握してくださいね。



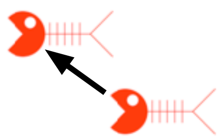
理解すべき事は以下の二つです。
・「進める」および「反転する」仕組み
・アニメーションの構造
これらのことを理解してから、課題を作るようにしてください。以下にそれぞれ説明します。

□ 「進める」および「反転する」仕組み

プログラムをよく調べて、上の図の「進める」ブロックの処理が以下のようになっていることを確認して下さい。
・通常状態でループするごとに少しずつ進む（図 1）
・しかし壁を越えて移動することは無いので（図 2）
・このまま進むと壁に当たるほど近づいたときは、反転して進む（図 3）

1

通常状態で「一歩」進む

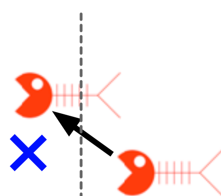


```
dx = -0.8 * size;
dy = 0.4 * size;
```
の状態で、
```
x += dx;
y += dy;
```
とすると、少し左上に移動する

(dx,dy)が「一歩」のベクトルと思えば良い

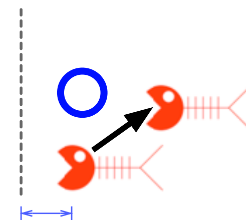
2

壁を越えて移動はしない



3

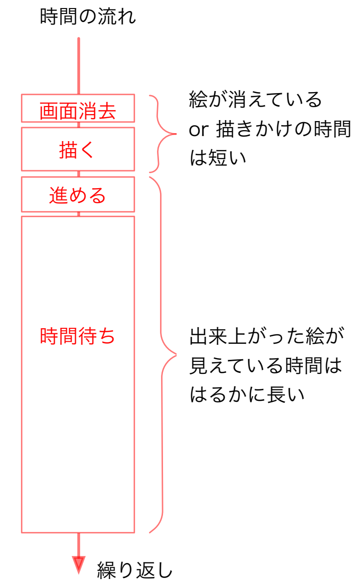
壁に当たる場合は反転する



一定以上壁に近かったら反転させる
```
dx *= (-1.0);
```
とすれば、それ以降の
```
x += dx;
```
進行方向が逆になる

dxの符号を反転させれば逆に進むよね？

□ アニメーションの構造



アニメーションは「静止画をパラパラを出す」ものです。fish.c では、画面を消してから描き終わるまでほぼ一瞬なのに対して、0.1 秒という「長時間」待ちます。この間に目には一コマが焼き付けられます。

次のループではほんの少し進んだ位置の絵ががまた「一瞬」で描画されて、また焼き付けられます。この繰り返しが「動いている」と人間には感じられるのです。プログラムがそのような動きをすることが分かりますか？

実際の各処理が消費する時間を（それっぽく）右に示します。
つまり一ページ前の「流れ図」では、行数に合わせて描く処理のブロックが大きくなったりしていますが、時間の割合としては「画面消去」「描く」「進める」は一瞬で、「時間待ち」が非常に長くなっていることを意識してください。

■ 課題の意図

□ 課題 1. サカナの描画を関数として独立させる

課題文をよく読んで、そこにある事項（描画処理を関数化する、引数と戻り値を指定のようにする）に従ってください。以下の点にも注意して下さい。

- あばら骨を描くためのループ処理は関数内の処理ですから、ループのためのカウンタ変数は関数内のローカル変数になるはずですね
- main() 関数側からはループ処理が無くなったので、そのループカウンタ変数は main() 側からは消すべきですね。

□ 課題 2. 右向けの絵も作る

こちらもどのようにするべきかは課題文に記述があります。

全体の構造としては右図のようになるはずですね。

```
void 左向きのサカナを描く( ……… ) {
    ……
    ……
}
```

```
void 右向きのサカナを描く( ……… ) {
    ……
    ……
}
```

```
int main() {

    …前処理…

    while(1) {
        HgClear();          // 画面を消去
        if ( 「右向き」に進んでいたら ) {
            「右向きの魚を描く」関数を呼ぶ
        } else {
            「右向きの魚を描く」関数を呼ぶ
        }
        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
    }
}
```