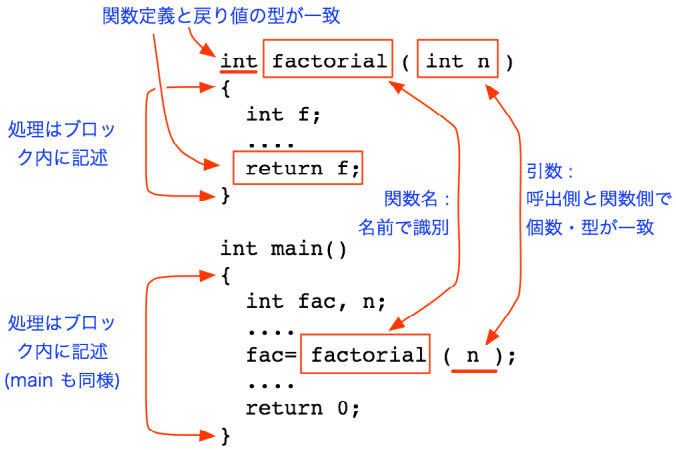
基礎プログラミング演習 II 教材　(#19)

■ 関数の記述・構造（復習）

右に前回のサンプルプログラムの構造を抜き出したものを示します。



・関数は main より前に書く
・main() は「main 関数」である
・関数の処理はブロック { } 内に書く
・関数には関数名が付く
・メイン側は関数名を用いて呼び出す
・引数の型と個数を両者で一致させる
・戻り値には型がある

「引数を受け取って処理が飛び込み、戻り値をもって帰る」関数の動作イメージがありますか？

■ 引数の無い関数・戻り値の無い関数

引数のある関数ばかりを使って説明してきましたが、引数は無くても構いません。

関数の定義では引数の列の代わりに void と書き、呼び出しの際は function( ) のように、関数であることを示すカッコだけを付けます。（右図参照）

（教科書 p.233 に void の記述が少しだけあります）

```
int function( void ) {
  .....
  return 0;
}

int main( ) {
  .....
  a=function( );
  .....
```

同様に戻り値がない関数もあり得ます。例えば引数の数値を出力するだけで戻り値が不要な場合です。

関数の定義では戻り値の型（関数自身の型）に void と書き、return には戻り値を置きません。（return は無くても、ブロックの最後に到達すれば関数からは戻ってくるため、return そのものを省略する場合も多い）

```
void function( int n ) {
  printf("n=%d\n", n);
  return;
}

int main( ) {
  .....
  function( 10 );
  .....
```

引数も戻り値もない関数は例えば以下のように定義され、呼び出されるでしょう。

定義：

```
void function( void ) {
    ....
    return;  <<-- 省略しても良い
}
```

呼出：

```
function( );
```

■ 変数の有効範囲・スコープ

★教科書 p.236, 13.2.1「変数の有効範囲とスコープ」を参照。

押さえて欲しいポイント：
・変数には有効範囲がある
・関数ごとに変数の有効範囲は限定され、他の関数に影響を与えない
　（全体をつらぬいて異なる名前の変数を用意し続けるのは困難である）
・これらをローカル変数と呼ぶ（厳密には限定単位は関数でなくブロック）
・全てのブロックの外（通常はプログラム先頭）で宣言する変数はグローバル変数と呼ぶ
・グローバル変数の有効範囲はプログラム全体である（全ての関数で共通に扱える）

なお、
・局所変数、大域変数とも言います。

□ 課題 1.

右のプログラムの実行結果を予想し、なぜそうなるか説明してください。また実際に実行してその予想と合っているか確認してください。
（予想せず実行するのは避けましょう。どんな結果になり、なぜそうなるかが重要です。）

ポイント：
・頭の中（あるいは机上）で変数の値を追いながら実行する
・大域変数 value が全ての関数から参照・代入できていること

```
int value;

void plus(int number)
{
  value = value + number;
}

void minus(int number)
{
  value = value - number;
}

int main(void)
{
  value=0;

  plus(5);  printf("value=%d\n", value);
  minus(8); printf("value=%d\n", value);
  plus(2);  printf("value=%d\n", value);

  return 0;
}
```

□ 課題 2.

以下の二つのプログラムは共に関数の呼び出し回数を（関数自身が）数えようとするものです。しかし左側はそれがうまくできず、右側はできています。なぜそうなるか、なぜ両者の結果が異なるのかを説明してください。また実際に実行してその予想と合っているか確認してください。

```
int f(void)
{
  int x;

  x=0;
  x++;
  return x;
}

int main(void)
{
  printf("answer=%d\n", f());
  printf("answer=%d\n", f());
  printf("answer=%d\n", f());
  return 0;
}
```

```
int x;

int f(void)
{
  x++;
  return x;
}

int main(void)
{
  x=0;
  printf("answer=%d\n", f());
  printf("answer=%d\n", f());
  printf("answer=%d\n", f());
  return 0;
}
```

□ 変数の初期化

局所変数でも大域変数でも、変数を宣言するときには初期設定を行うことが出来ます。例えば右のように記述します。
すなわち初期設定の構文は次のようになります。

```
int x=100;
int y=200;
```

データ型　変数名 = 初期値式;

従って課題 2.のプログラムでは、右のように二行に分けず、まとめて書くことが可能です。(特に課題 2.の右側の例では記述を一カ所にまとめられます。)

```
int x;
x=0;
```

⇨

```
int x=0;
```

初期設定が行われるタイミングは変数の種類によって異なります。

1. ローカル変数は定義されている関数（ブロック）に入るごとに初期設定されます。
2. 大域変数はプログラムの実行開始時に一度だけ行われます。

> 実際には実行効率を上げるために、ほとんどの場合 2.の初期設定はコンパイル時にすでに行われています。このために細かいことをいえば 2.の初期設定で初期値式に使える式には多少制限があります。2.の場合はコンパイル時にその値があらかじめ計算して定まるような式でないといけません。一方、1.の場合は実行時にブロックに入るたびに値が異なるような式でも構いません。

さらに、変数の初期設定が行われていない場合の初期値については、これも変数の種類によって 2 通りに分かれます。

1'. ローカル変数ではその値は不定です。
2'. 大域変数では暗黙に 0（すべてのビットが 0 の値）に初期設定されます。

従って課題 2.右側のサンプルプログラムでは、実際には大域変数 t の初期設定をしなくても正しく動作します。(しかし意図的にゼロで初期化したい場合は =0 と明示的に書く事を勧めます。)

■ プログラムの分割

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、左はすべてのまとめて書いた場合、右は関数を用いて準備処理と描画処理を抜き出した場合の例です。プログラム全体の流れなどがよりわかりやすく記述できたことがわかるでしょう。

```
#include <stdio.h>
#include <stdlib.h>
#include <eggx.h>

int main() {
  int win, seed, i;
  float x[10], y[10], w, dx[10], dy[10];

  printf("random seed = ");
  scanf("%d",&seed);
  srand(seed);

  win=gopen(400,400);
  winname(win, "sample 1");

  gsetbgcolor(win, "white");
  newpen(win, 2);

  w=10.0;
  for(i=0; i< 10; i++) {
      x[i]=(float)(rand() % 400);
      y[i]=(float)(rand() % 400);
      dx[i]=(float)(rand() % 15 + 2) / 10.0 * w;
      dy[i]=(float)(rand() % 15 + 2) / 10.0 * w;
  }

  while(1) {
    gclr(win);
    for(i=0; i< 10; i++) {
        fillarc(win, x[i], y[i], w, w, 0.0, 360.0, 1);
        y[i]+= dy[i];
        if( y[i] < 0.0 || y[i] > (400.0 - w) ) dy[i]*=(-1.0);
        x[i]+= dx[i];
        if( x[i] < 0.0 || x[i] > (400.0 - w) ) dx[i]*=(-1.0);
    }
    msleep(50);
  }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <eggx.h>

void setup( int n )
{
  ........;
  ........;
  ........;
}

void draw( int n )
{
  ........;
  ........;
  ........;
}

int main() {
  ........;
  ........;
  ........;

  setup(10);

  while(1) {
    draw(10);
    msleep(50);
  }
}
```

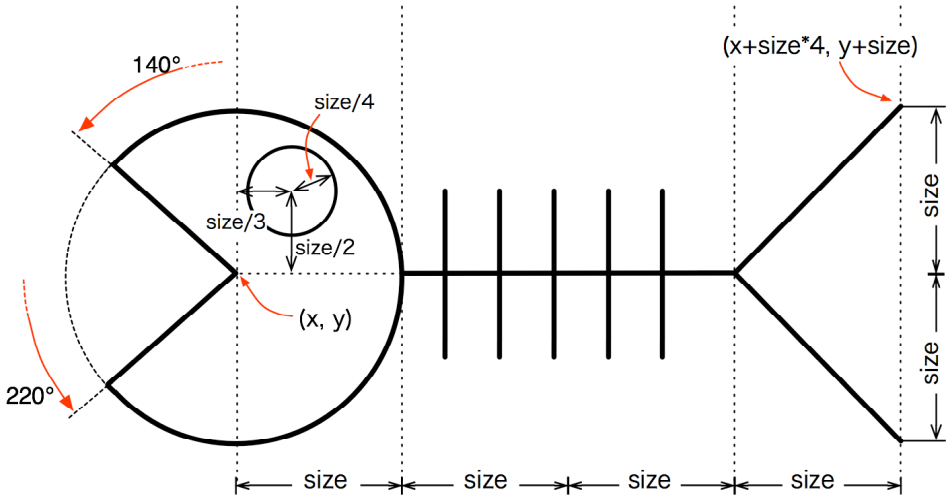
準備処理
（初期設定など）

描画処理

全体制御：
準備処理の後に、無限ループで描画を繰り返すこと、少し sleep することなどがよく分かる。

□ 課題 3.

サンプルプログラム(fish.c)を入手して、内容をチェックして、実行してください。これは過去にやった「跳ね返るボール」とほぼ同じプログラムですが、ただ描くものが円一つでなく、下図のような魚の絵になっています。
具体的には魚の頭の中心を変数 x, y の位置 (x, y) として、変数 size で指定された距離を用いて下図のような構造で描いています。



このプログラムを修正して、魚を描画する処理を関数として独立させてください。

描画する座標位置やサイズを引数として渡せば良いでしょう。
全ての描画関数に必要で、内容に変化がない変数 win は大域変数としても良いかもしれません。

□ 課題 4.

課題 3. 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。(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）にしてください。)

方針としては、
・課題 3.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。
（他の方法でも構いません）