配列、関数定義

■ 配列

サンプルプログラム snow1.c を以下に示す。講師の教材ページなどからとってきて実行せよ。seed （後述）となる数値を入力すれば、白い四角を上から落とし、下まで到達すればランダムな横位置を設定してふたたび上から落とす。

今回特別なポイントは得になし。(いままでやってきたことをそのままやっているだけ)

```
#include <eggx.h>
#include <stdlib.h>
int main() {
  int win, seed;
  double sx,sy;
  printf("please set the seed :");
  scanf("%d",&seed);
  srand(seed);  /* seed の設定 */
  /* 雪の位置を決める */
  sx=(double)(rand()%400);
  sy=(double)(rand()%400);
  win=gopen(400,400);  /* 描画ウィンドウを開く */
  winname(win, "snow"); /* 名前をつける */
  newpen(win, 1);
  while(1) {  /* 永遠に繰り返し */
    fillrect(win, sx-2.0, sy-2.0, 4.0, 4.0);
    sy=sy-3.0;
    if(sy<2.0) {
      sx=(double)(rand()%400);
      sy=400.0;
    };
    usleep(60000);
    gclr(win);
  };
}
```

ところでこのまま三つの雪つぶを降らせようと思った場合、最も簡単には右の例のように、三セットの変数を用意して処理する方法がある。
（サンプルプログラムが snow2.c として用意してあるので試すと良い。）

```
double sx1,sy1,sx2,sy2,sx3,sy3;
/* 雪の位置を決める */
sx1=(double)(rand()%400);
sy1=(double)(rand()%400);
sx2=(double)(rand()%400);
sy2=(double)(rand()%400);
sx3=(double)(rand()%400);
sy3=(double)(rand()%400);
```

しかしこれでは数が増えるとプログラミングするのに手間が掛かって大変である。こうしたケースでは配列変数を利用すると簡単にプログラミングできる。
次の例を見よ。

```
double sx[20];
```

と書くことで、sx[0] から sx[19] までの 20 個ぶんの変数が使えるようになる。sx[5]とは、配列変数 sx の 6 番目の（5 の位置にある）「要素」と表現する。要素を指定する [ ] の指定のことを配列の「添え字」という。添え字には変数や計算式が使える。

```
sx[5]=100.0;
sy[i]=sy[i]*1.5;
```

といった使い方ができる。

```
  double sx[20],sy[20];
….
  for(i=0; i<20; i++) {
    /* 雪の位置を決める */
    sx[i]=(double)(rand()%400);
    sy[i]=(double)(rand()%400);
  };
```

上の例は for 文によって添え字を変数 i に与え、要素 0 から 19 までのすべてにランダムな値（x, y 座標）を与えるプログラムとなっている。

■ 関数定義（サブルーチン）

サンプルプログラム dora1.c では簡単な似顔絵を描いている。このプログラムではただ簡単な絵を描くだけであり、dora2.c では目を開いたり閉じたりするアニメーションを描く。

```
  /* 右は点、左はへの字 */
  newpen(win, 0);
  fillarc(win, x-12.0, y+120.0, 8.0, 20.0, 0.0, 360.0, 1);
  line(win, x+12.0, y+110.0, PENUP); line(win, x+27.0, y+125.0, PENDOWN);
  line(win, x+12.0, y+111.0, PENUP); line(win, x+27.0, y+126.0, PENDOWN);
  line(win, x+12.0, y+112.0, PENUP); line(win, x+27.0, y+127.0, PENDOWN);
  line(win, x+12.0, y+113.0, PENUP); line(win, x+27.0, y+128.0, PENDOWN);
  line(win, x+27.0, y+125.0, PENUP); line(win, x+42.0, y+110.0, PENDOWN);
  line(win, x+27.0, y+126.0, PENUP); line(win, x+42.0, y+111.0, PENDOWN);
  line(win, x+27.0, y+127.0, PENUP); line(win, x+42.0, y+112.0, PENDOWN);
  line(win, x+27.0, y+128.0, PENUP); line(win, x+42.0, y+113.0, PENDOWN);
```

上の記述は右目（向かって左）に黒丸、左目が「へ」の字になるように斜めの線を入れている。（一本だけだと細いので四本並べて引いている）

```
line(win, x+12.0, y+110.0, PENUP); line(win, x+27.0, y+125.0, PENDOWN);
```

は（x+12.0, y+110.0）の位置（真ん中より 12.0 右、110.0 上）に PENUP の状態で描画点（ペンの位置）を移動し、続いてそこから少し右上（27.0 右、125.0 上）に向けてPENDOWN の状態で移動する。つまりペンを握った状態で描画点を移動することで線を引くことができる。

dora2.c では、アニメーションを作るため、何度も一連の手続きを繰り返して実行することになる。そのとき上のようなプログラムを繰り返して記述するのは面倒である。（dora2.c の中身を確認すると良い。）

dora3.c では、こうした煩雑さを自分用の関数を定義することで解決している。以下に関数定義の記述を示す。pati() と kuri() という関数を二つ定義している。

```
int pati(int win, double xx, double yy)  /* 丸の黒目を描く */
{
  fillarc(win, xx, yy, 8.0, 20.0, 0.0, 360.0, 1);
  return(0);
}

int kuri(int win, double xx, double yy)  /* 閉じた目を描く */
{
  int i;
  for(i=0;i<4;i++) {
    line(win, xx, yy+i, PENUP); line(win, xx+15, yy+i-15.0, PENDOWN);
    line(win, xx, yy+i, PENUP); line(win, xx-15, yy+i-15.0, PENDOWN);
  };
  return(0);
}
```

これを `pati(win, x-12.0, y+120.0);` や `kuri(win, x+27.0, y+120.0);` のようにして呼び出すことで、一連の処理を実行させることができる。dora2.c と dora3.c を比べてみると良い。

呼び出し時に引数として渡す部分、それを受け取る部分の記述に注意すること。上の例では pati() の呼び出し時に与えた win, x-12.0, y+120.0 はそれぞれ win, xx, yy の三つの変数に渡されて実行される。

■ EGGX の関数（一部）

□ 点を打つ
pset(win, x, y)
座標位置（x, y）に点を打つ。x, y は実数型変数。

□ 線を引く
line(win, x, y, m)
最後に点や線などの図形を描いた位置（描画点と呼ぶことにする）から、座標位置 (x,y) に m の値に応じて線を引く。m がPENDOWN なら線を引き、PENUPなら線を引かずに描画点だけを移動する。
例えば (x1, y1) から (x2, y2) に向けて線を引くときは以下のようにする。
line(win, x1, y1, PENUP); line(win, x2, y2, PENDOWN);
例えば以下のようにすると、(x1, y1) — (x2, y2) — (x3, y3) をむすぶ折れ線を描く。
line(win, x1, y1, PENUP); line(win, x2, y2, PENDOWN); line(win, x3, y3, PENDOWN);

□ 四角い枠を描く
drawrect(win, x, y, w, h);
座標位置 (x, y) から幅 w、高さ h の四角を描く。x, y, w, h いずれも実数型変数。
例：座標位置 (250, 120) を中心とした幅 20, 高さ 10 の長方形の枠を描く。
x=250.0; y=120.0;
w=20.0; h=10.0;
drawrect(win, x-(w/2.0), y-(h/2.0), w, h);

□ なかを塗りつぶした四角を描く
fillrect(win, x, y, w, h);
上に示した drawrect() 関数と使い方は同じ。違いは中を塗りつぶすかどうかだけ。

□ 円を描く
circle(win, x, y, w, h);
座標位置 (x, y) から横幅 w 、高さ h の円を描く。x, y, w, h いずれも実数型変数。
例：座標位置 (250, 120) を中心とした横方向半径 20, 縦方向半径 10 の円を描く。
x=250.0; y=120.0;
w=20.0; h=10.0;
circle(win, x, y, w, h);

□ 円弧を描く
drawarc(win, x, y, w, h, s, e, d);
座標位置 (x, y) から横方向半径 w, 縦方向半径 h の弧を描く。その開始角度は s 度から e 度まで（0-360までの度を与える)。d が 1 なら左回りに描き、-1 なら右回りに描く。x, y, w, h, s, e は実数型変数。ｄのみ整数型。
例：右回りに 30 度から 60 度の位置に（30度ぶんの開き角となる）扇形を描く。
drawarc(win, x, y, w, h, 30.0, 60.0, -1);

□ なかを塗りつぶした円弧（扇形）を描く
fillarc(win, x, y, w, h, s, e, d);
上に示した drawarc() 関数と使い方は同じ。違いは中を塗りつぶすかどうかだけ。
s=0.0; e=360.0; としてfillarc()を呼び出すと中が塗りつぶされた円を描くことになる。

□ 文字列を描く
drawstr(win, x, y, s, t, format, v)
(x, y) から s のサイズで文字を描く。t は文字列の回転角を示すがいまは機能しないので0.0などを与えておく。format は printf() 関数で使われる印刷指示。v はそこで使われる変数。x, y, t は実数型、s は整数型で 1-24 を与える。format は文字型。v は format 次第で決まる。
例：座標位置 (250, 120) から 16 のサイズで「temperature=25」と描かせる。
v=25; s=16;
drawstr(win, 250.0, 120.0, s, 0.0, “temperature=%d”,v);
（漢字やひらがなを表示させるには以下のようにサイズに FONTSET と書く。
drawstr(win, 250.0, 120.0, FONTSET, 0.0, “室温=%d”,v);