プログラミング演習 A 教材 (#7)

■ グラフィクス、条件分岐

これ以降、EGGX を利用したグラフィクス・プログラミング（画像を描くプログラムを書く）をとりあげます。(C プログラミングガイド EGGX の章を参照)
以下のプログラムを入手して実行してください。

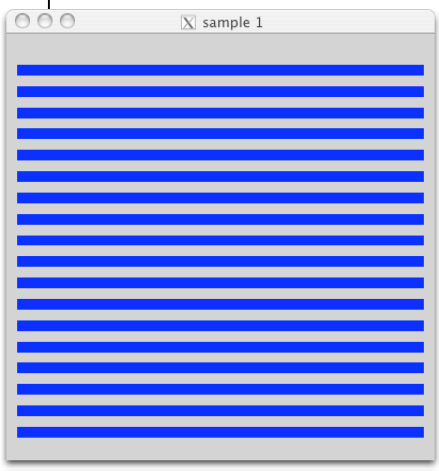
グラフィクスプログラムをコンパイルする場合は、cc ではなく egg コマンドを使います。
（egg コマンドによるコンパイルでも –o オプションによる実行ファイル名の指定が可能です。付けないばあいは a.out の名前で作成されます。）

```
axt22029(84)% egg –o egsample egsample.c
gcc -O2 -Wall -oegsample egsample.c -I/usr/bin -L/usr/bin -I/usr/X11R6/include
 -L/usr/X11R6/lib -leggx -lX11 -lm
axt22029(85)% ls
egsample  egsample.c
axt22029(86)% ./egsample
```

実行すると画面上に絵が表示され、何かキーを押すと終了することがわかるでしょう。

```
/*
 EGGX を使ったサンプル  473088 榎田裕一郎
*/
#include <stdio.h>
#include <eggx.h>

int main() {
  int win;
  float x,y;

  win=gopen(400,400);  /* 描画ウィンドウを開く */
  winname(win, "sample 1"); /* 名前をつける */

  newpen(win, 4);  /* 描画する色を設定 */

  x=10.0; y=20.0; /* x,y の初期座標位置 */

  while( y < 380.0 ) {
    fillrect(win, x, y, 380.0, 10.0);
    y+=20.0;
  }

  ggetch(win); /* キー入力を待つ */
  gclose(win); /* 描画ウィンドウを閉じる */

  return 0;
}
```

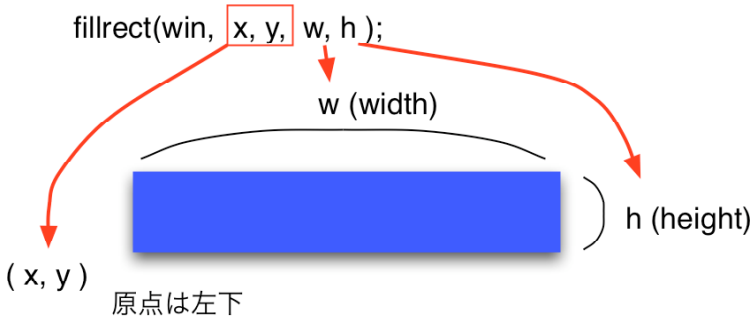
gopen( ) や fillrect( ) などのグラフィクスを描画するために使っているものは、printf( ) などと同じく「関数 (function) 」と呼ばれています。



押さえて欲しいポイント：

・#include <eggx.h> で EGGX に関する準備をする。
・gopen (400, 400) で 400 x 400 画素の描画ウィンドウを開く。
・newpen( ) で色を指定する。色番号は 0 – 15 が設定可能で、それぞれの色は
　0:黒 1:白 2:赤 3:緑 4:青 5:シアン 6: マゼンタ 7:黄 8:灰色(暗) 9:灰色
　10:赤(暗) 11:緑(暗) 12:青(暗) 13:シアン(暗) 14:マゼンタ(暗) 15:黄(暗)
・newpen(win, 4) などのグラフィクス関数の先頭にある引数 win はお決まりなので気にしない。

・fillrect( ) で長方形を描く。引数は左から描く長方形の位置（左下カドの座標）を x, y の順に、次に幅、高さを示す。（右図）
塗りつぶしの色は fillrect( ) 実行前にnwpen( ) で指定した色。

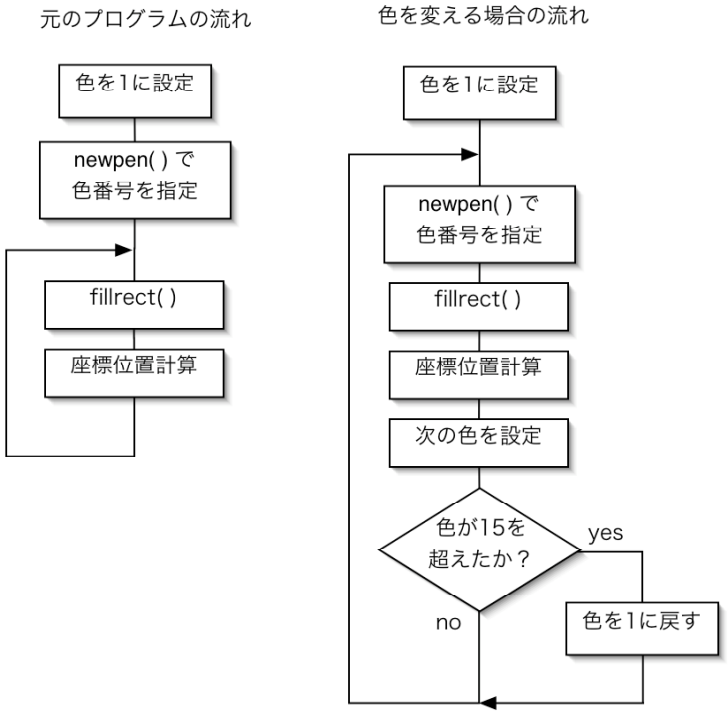


・ggetch( ) でキー入力を待つ。これが無いと「描画ウィンドウを開き、描いて、すぐ消える」ため、結果が見えない。（目に止まらない）
・gclose( ) で描画ウィンドウを閉じる。

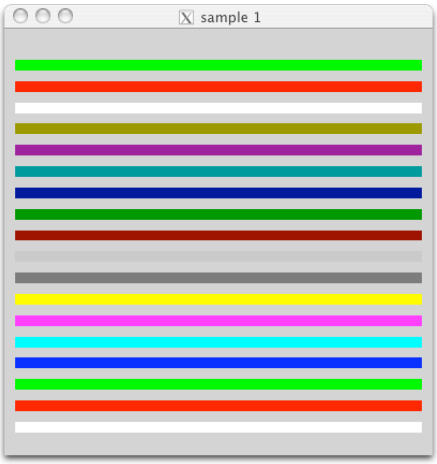
□ 棒の色を変える（条件判断のあるプログラム）

棒の色を一本ずつ変えるようにプログラムを変更します。
考え方：
・newpen( ) で色を変えるのですが、黒（色番号 0 ）は背景と同じ色で見えないので避ける。
・15 色（ 1 – 15 ）使いきったらまた 1 から始めれば良い。

右図を見て下さい。左側がサンプルプログラムそのまま、右側が棒の色を一本ずつ変える場合の処理の流れです。
（while による繰り返し条件判定の部分は省略しています。）



このように何かの条件によって処理の内容を変える場合は、if 文を使います。



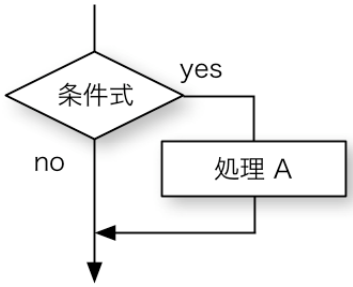
■ 条件分岐

何かの条件によって処理の内容を変える場合は、if 文などを使った条件分岐を設けます。

□ if 文

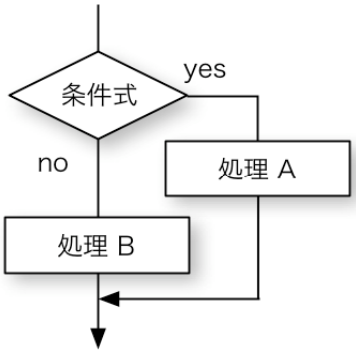
書式：

```
if ( 条件式 ) {
  条件が成立したときの処理 A ;
}
```



または

```
if ( 条件式 ) {
  条件が成立したときの処理 A ;
} else {
  条件が成立しなかったときの処理 B ;
}
```



上記の処理 A, B 部分には条件が成立した（またはしなかった）時の処理を何行でも書けますが、それらが 1 行しか無い場合に限り、{ } を用いず下記のように短く書くこともできます。

```
if ( 条件式 ) 処理 A ;
if ( 条件式 ) 処理 A ; else 処理 B ;
```

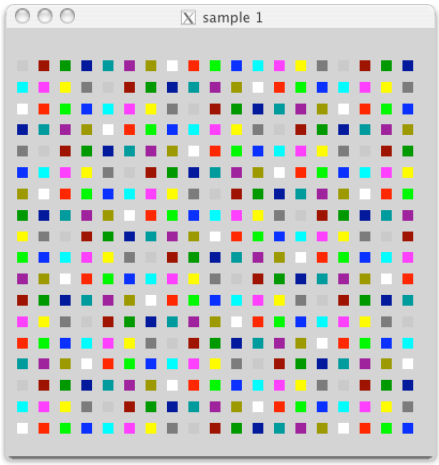
□ 課題 1.

if 文を使って色を変えるようにプログラムを修正して実行してください。
今回は「色番号が 16 になったら 1 に戻す」という条件分岐を設けることになるので、例えば右のような if 文を用いれば良いでしょう。

```
c++;
if( c == 16 ) {
  c=1;
}
```

□ 課題 2.

if 文を使って小さな四角で画面を埋めるようにして下さい。



ヒント：
・まず x 座標を変化させながら小さな四角を描く。
・x が右端に達したら x を 10.0 に戻し、y を変化させる。
・y が上端に達したら終了。

最後の y が上端に到達したら終了、という処理は元のサンプルプログラムに組み込まれているため、今回は考慮しなくても良い筈。

注意：
if 文を使って同じ処理をさせる場合でも、さまざまな書き方がある点に注意。プログラムにはさまざまな書き方があり、唯一の解というものは無い。

■ 条件式　(if 文や whie( ), for( ) などで使えます。)

□ 比較などを行う関係演算子

数値などの大小比較について以下のような演算子が利用できます。

演算子	意味	利用例
==	等しい	`if (a==b) { ... }`
!=	等しくない	`if (a!=b) { ... }`
>	左辺が大きい	`if (a>b) { ... }`
>=	左辺が等しいか大きい	`if (a>=b) { ... }`
<	左辺が小さい	`if (a<b) { ... }`
<=	左辺が等しいか小さい	`if (a<=b) { ... }`

==, != を等値演算子、>, < などを関係演算子と呼んでいます。
これらより算術演算子のほうが優先度が高いため、
if(a < b-1) という記述は
if(a < (b-1)) と同じとみなされます。
（左から順に処理されてまず a < b が先に処理されるようにはなりません。）

□ 複数の条件を並べる論理演算子

複数の条件を並べて判定したい場合は以下のように書きます。

演算子	意味	利用例
&&	AND （両方の条件が成立したら）	`if (a==b && a<100) { ... }` （a と b が等しく、かつ a が 100 未満なら真）
||	OR （どちらかの条件が成立したら）	`if (a==b || a<100) { ... }` （a と b が等いか、または a が 100 未満なら真）
!	NOT （条件の反転）	`if (! a==b) { ... }` （a と b が等しくなければ真）

&& や || を論理演算子と呼んでいます。否定の ! は否定演算子と呼ばれています。
これらは関係演算子よりさらに低い優先度が設定されているので、
if(a < b && c > d) は
if((a < b) && (c > d)) として処理されます。
また、
if(a < b-1 && c + 2 > d -5) は
if((a < (b-1)) && ((c + 2) > (d -5)) として処理されます。

（なるべくバグを発生させない、プログラマの勘違いを誘発させないようにするために、暗黙の優先順位に依存した複雑な論理式を書くより、( ) を明示的に使ってわかりやすい記述を心がける方がよいでしょう。）

■ break , continue, 無限ループ

□ break 文

ループを途中で打ち切りたい、何か条件が成立すればループから脱出したいような場合には break 文が便利です。for, while, do~while によるループの中で break 文を実行すると、その時点でループを打ち切ります。(switch 文でも break を使いますが、これについては次週)

通常、break は if 文と組み合わされて利用されるでしょう。一つ簡単な例を示します。
以下の左側プログラムは「1～10 までの整数の中で 3 の倍数を表示する」ループです。これを break を使って「1～10 までの整数の中で最初に見つけた(最小の) 3 の倍数だけを表示する」ように変更しました(同右側)。実行結果から、break によって最初に倍数を見つけた時点でループから脱出して処理を終了しているのが分かるでしょう。

```
for(i=1; i<=10; i++) {
  if( i%3 == 0 ) {
    printf("I found it! (%d)\n", i);
  }
}
```

実行結果:
```
I found it! (3)
I found it! (6)
I found it! (9)
```

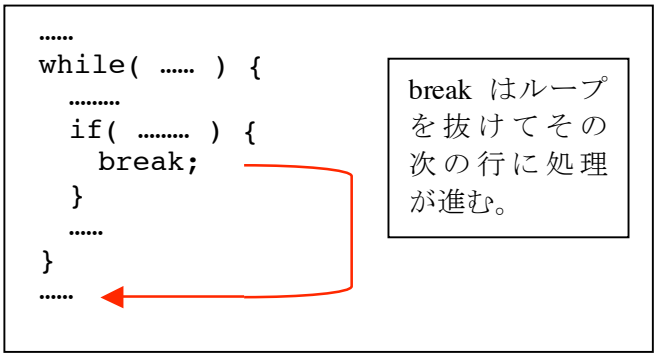
```
for(i=1; i<=10; i++) {
  if( i%3 == 0 ) {
    printf("I found it! (%d)\n", i);
    break;
  }
}
```

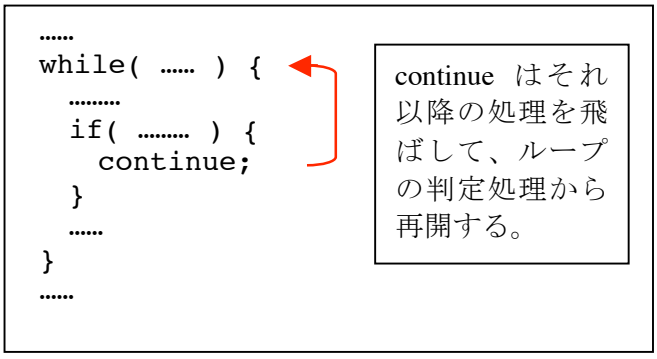
実行結果:
```
I found it! (3)
```

もし break が多重ループの中にあった場合は、その break が含まれる最も内側のループから脱出します。

□ continue 文

continue 文はループ内の処理をそこで止め、次回のループの繰り返し判定処理から続けるものです。(for の場合は判定の前に繰り返し毎処理を行いますので、そこから続けることになります。)

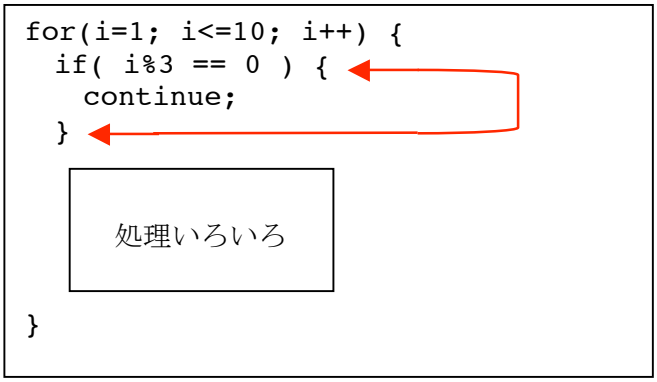


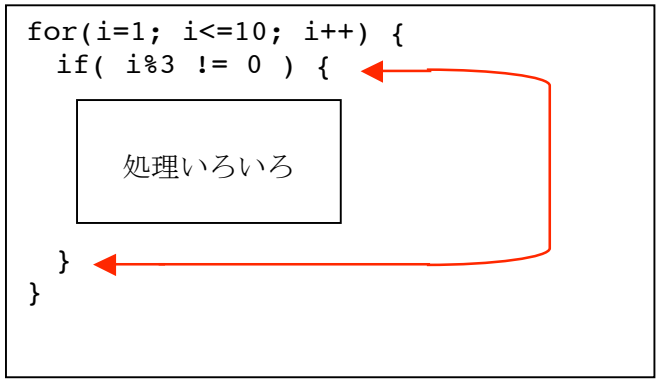


再開は次回ループのための判定処理からです。while や for の場合は上右図のように処理が上に戻るイメージになりますが、do~while の場合は判定がループ末尾にありますから、下まで飛ばすような形になります。

continue を使うケースはそう多くなく、ループの最初に「y が負なら処理不要」といった除外条件をそれと分かりやすく書くために使われる場合などに有効です。

以下に例を示しておきます。左右共に同じ振る舞いをするプログラムですが、continue を使って記述した左の方が、冒頭の if 文のことを考慮しなくてはいけない範囲（矢印でマークした部分）が短くなっています。右側のように記述した場合、if 文の終わりが遠くなり、何十行も間に入ってしまうと分かりにくくなってしまいます。





□ 無限ループ

C 言語では条件式の結果として、条件が成立すれば（真の場合は）1 が、不成立の場合（偽の場合）は 0 が得られます。つまり a が 5 の時 if(a<10) {…} は if(1) {…} と同じことになります。
これを利用して while(1) { …… } のように無限に繰り返すループを作る場合もあります。（条件式が 1 つまり常に真なので、この while ループは終わることがない。）

while, for, do ~ while などのループからは break によって脱出することができるため、無限ループはよく if 文及び break 文と組み合せた形で使われます。

以下に while(1) を使った無限ループのプログラム例を示します。もし y が 100.0 を越えたら、という条件で break する部分に注目して下さい。