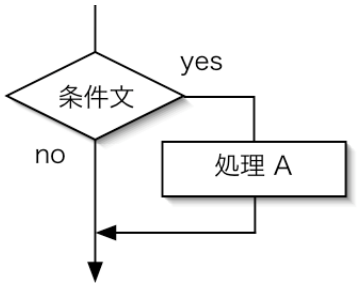
コンピュータ基礎実習上級 教材 (#7)

■ 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。

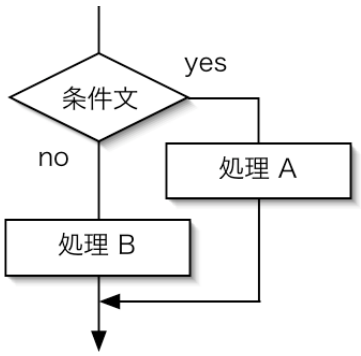
□ if 文

書式：
if (条件文) {
　条件が成立したときの処理 A;
}

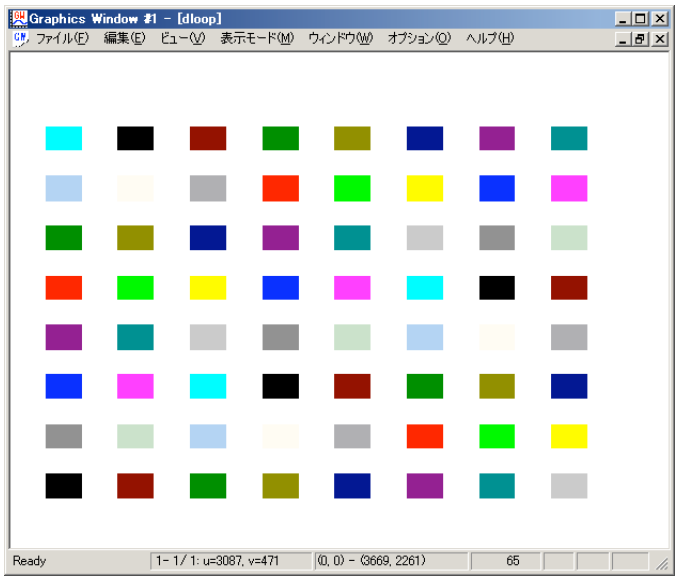


または

if (条件文) {
　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;
};



● 課題 1.



次々と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。色番号は 0 から 18 まで（19 は白）ですので、今回は「色番号が 18 になったら 0 に戻す」という条件分岐を設けると良いでしょう。

```
c++;
if( c == 19 ) {
  c=0;
};
```

うまくできた人は「偶数の色番号の場合は四角ではなく円を描く」ようにしてください。

参考：
(x1, y1)-(x2, y2) に色 c の四角を描く
```
GWsrect(x1, y1, x2, y2, c);
```

(x1, y1)-(x2, y2) に色 c の円を描く
```
GWscircle(x, y, x+10.0, y+10.0, c);
```

> 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。）

□ 条件式の結果

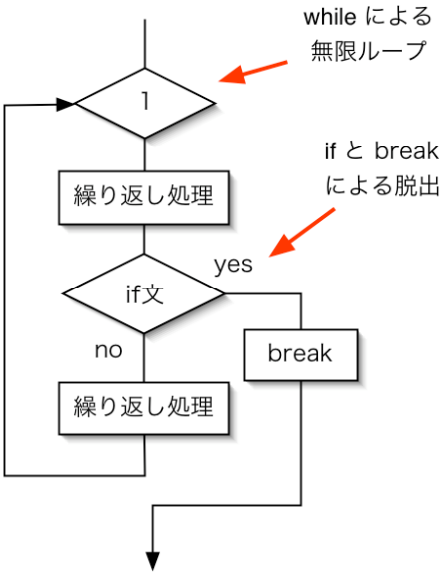
C 言語では条件式の結果として、条件が成立すれば（真の場合は）1 が、不成立の場合（偽の場合）は 0 が得られます。つまり a が 5 の時 if(a<10) {…} は if(1) {…} と同じことになります。
これを利用して while(1) { …… } のように無限に繰り返すループを作る場合もあります。

■ アニメーション

サンプルプログラムを実行してください。(下はその抜粋)

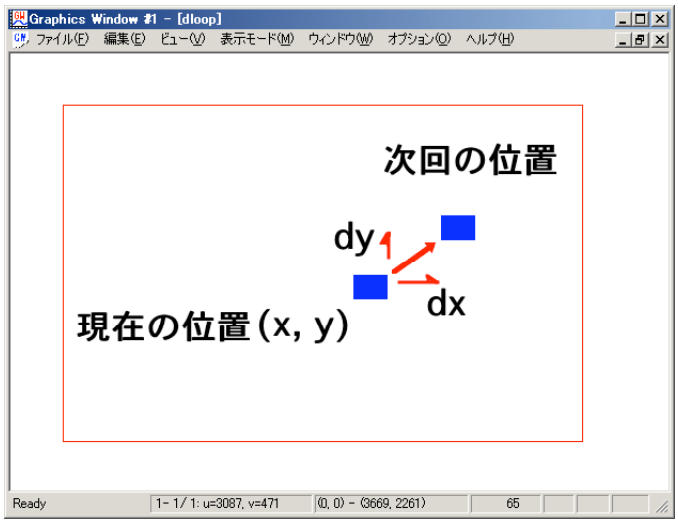
```
x=80.0; y=20.0; /* 最初の位置 */
dx=2.0; dy=0.4; /* ステップごとの進行速度 */

while(1) {
  /* 赤い枠 */
  GWxrect(10.0, 20.0, 160.0, 160.0, 13);
  if( (x >= 150.0) || (x <= 10.0) ) {
    /* 左右の壁に当たったら方向転換 */
    dx= -1.0 * dx;
  };
  x+=dx; /* x 軸方向にワンステップ進む */
  if(y >= 150.0) {
    break; /* while ループから脱出 */
  };
  y+=dy; /* y 軸方向にワンステップ進む */
  GWsrect(x, y, x+10.0, y+10.0, 16);
  GWsleep(20);
  GWclear(-1);
};
return 0;
```



注目して欲しいポイント:
- while(1) による無限ループ
- break によるループからの脱出
- GWclear による画面の消去
- GWsleep による待ち

ループごとの処理の内容:
- ループごとに x, y の座標位置を dx, dy を加算することで計算
- 左右の壁に当たったら跳ね返るように見せる
- dx の符号を逆転することでこれを実現する



● 課題 2.

このプログラムを修正して、y 軸方向にも跳ね返るようにしてください。

注意:
いつまでも終了しないプログラムとなるので、return は不要。終了するときは描画ウィンドウを強制的に閉じてしまえばよい。