オブジェクト指向とゲームプログラミング

● 基礎編 ― 第７回　構造化プログラミング ●

構造化プログラミング

構造化プログラミングには、「プログラムの構造化」と「データの構造化」があります。プログラムの構造化とは、以下の要素からなります。

①順次・選択・反復の制御構造のみを使う
②プログラムを小さな機能の単位(モジュール)に分割し、ほかからの影響を防ぐ
③複雑な機能を分割し、構造を階層化する

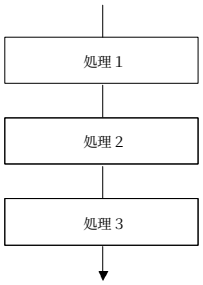
構造化プログラミングといえば、中心的な意味は①となります。なぜ、構造化プログラミングが必要なのかというと、プログラムが複雑になるからです。実用的なプログラムは、1000行以上というのは普通で、数万行、数十万行になります。ということは、「式」「文」「関数」が数万、数十万と使われていることになります。
複雑なプログラムを作るうえで、バグがより少ないよう、またバグがある場合は、修正しやすいようなプログラムの品質が求められます。その答えの１つが構造化プログラミングというわけです。

構造化プログラミングの基本構成要素

構造化プログラミングのもともとの意味は「順次・選択・反復」のみを使用したプログラミングとなりますが、Ｃ言語ではどう実装するかを説明します。

順次

「順次」とは、プログラムの要素が順番に処理されていくことです。たとえば、処理１の次に処理２を行って、その次に処理３を行って…というような処理の方法です。これは、普通にプログラムを書いていけば、ほとんどの場合順次になります。




```
・・・・
処理１;
処理２;
処理３;
処理４;
・・・・
```

選択

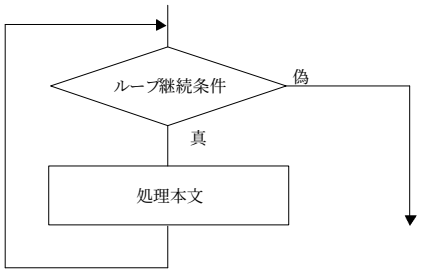
「分岐」ともいいます。条件を判断して、処理の流れを変えることです。「条件が合っていたら処理をする」場合と、「条件が合っていたら処理１をし、合っていなかったら処理２をする」場合があります。また、派生的なものとして、変数の値を調べていくつかの場合に分岐することもあります。
選択にはif文とswitch文があります。if文は２分岐、switch文は多分岐を行います。

反復

「繰り返し」ともいいます。プログラムの一部分を繰り返し実行することです。１回ごとに何らかの条件をチェックして、条件が成立している間は繰り返しを続けます。
反復には、while文、for文、do-while文があります。

while

while文は、以下のように処理が行われます。ループに入る前に条件判定を行い、条件が成り立つ間処理を反復します。



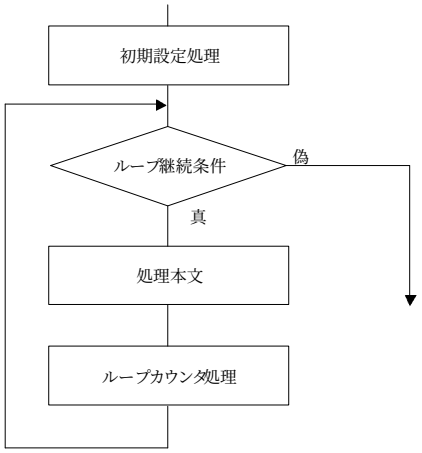
while文の書式は以下のようになります。

```
while(ループ継続条件) {
        ⋮
    処理本文
        ⋮
}
```

「ループ継続条件」は、ループを継続させるための条件を記述します。この条件が真の間、ループが継続されます。ループに入る前に判定が行われるので、条件によってはループに入らない場合があります。

for

for文は、以下のように処理され、条件が真の間(成り立つ間)処理を反復させることができます。



for文は、おもに回数を指定したループを行うときに使用します。回数を管理するために、ループカウンタという変数を使用します。for文の書式は以下のようになります。

```
for(初期設定処理; ループ継続条件; ループカウンタ処理) {
        ⋮
    処理本文
        ⋮
}
```

「初期設定処理」は、ループに入る前に一度だけ必ず実行されます。通常は、ループカウンタの宣言と初期化を行います。","で区切ることにより、複数の変数を宣言し、初期化することができます。

「ループ継続条件」は、ループを継続させるための条件を記述します。ループを終了させるための条件ではないことに注意してください。条件が真の間、ループが継続されます。この条件は、初期設定処理の後に判定が行われるので、条件によってはループに入らない場合があります。

「ループカウンタ処理」は、処理本文の後に実行され、ループカウンタの増減を行う処理を記述します。","（カンマ）で区切ることにより、複数のカウンタを処理することができます。

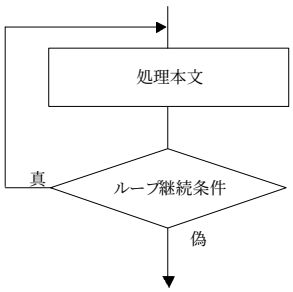
「初期設定処理」「ループ継続条件」「ループカウンタ処理」は、必要のないものは省略することができます。３つの処理すべてを省略してしまうこともできます。

以下のようにfor文を記述すると、iが0から9までの１０回、処理本文が実行されます。

```
for(int i = 0; i < 10; i++) {
    処理本文
}
```

do-while

do-while文は、ループの最後に条件判定を行い、真の間処理を反復します。



do-while文の書式は以下のようになります。while文の最後に、";"（セミコロン）が必要なことに注意してください。

```
do {
        ⋮
    処理本文
        ⋮
} while(ループ継続条件);
```

「ループ継続条件」は、ループを継続させるための条件を記述します。この条件が真の間、ループが継続されます。ループの最後に判定が行われるので、最低でも１回は処理本文が実行されます。

break

ループは、継続条件が真である限り続けられますが、breakで強制的に抜けることができます。
breakは、ループとswitch文の{}で囲まれたブロックの外側に抜け出すことができます。多重ループの場合は、breakを実行したループからのみ抜け出すことができます。多重ループの外側に一気に抜け出すことはできません。
たとえば、以下のプログラムでは、ループ継続条件（i < 10000）が偽になるか、jが65535より大きくなるとループが終了します。

```
for(int i = 0, j = 0; i < 10000; i++) {
    処理本文１

    if(j > 65535)
        break;

    処理本文２
}
```

以下のような場合、kが0になるとコメントの部分まで抜けます。多重ループの外側ではないことに注意してください。

```
int   k;
for(int i = 0; i < 10000; i++) {
    for(int j = 0; j < 2000; j++) {
        処理本文

        if(k == 0)
            break;
    }
    // breakはここに抜けます。
}
```

continue

continueは、ループ内の処理の一部をスキップすることができます。for文の場合は、continueが実行されると、そのループに関するループカウンタ処理が行われ、ループ継続条件の判定が行われます。while文とdo-while文では、そのループに関する継続条件の判定が行われます。

たとえば、以下のプログラムでは、jが65535より大きくなるとcontinueが実行され、処理本文２がスキップされます。

```
for(int i = 0, j = 0; i < 10000; i++) {
    処理本文１

    if(j > 65535)
        continue;

    処理本文２
}
```

以下のような場合は、continueは内側のループの「処理本文２」をスキップします。

```
int   k;
for(int i = 0; i < 10000; i++) {
    処理本文１
    for(int j = 0; j < 2000; j++) {
        if(k == 0)
            continue;
        処理本文２
    }
}
```

無限ループ

for文やwhile文を使って、無限に反復する「無限ループ」を確立することができます。それぞれ以下のようになります。

```
for(;;) {
    処理本文
}

while(1) {
    処理本文
}
```

無限ループからの脱出は、breakで行います。

練習問題

1　以下の(１)～(４)の「処理(省略)」は、それぞれ何回実行されるか答えよ。

(１)
```
for(int i = 0; i < 100; i++) {
    処理(省略)
}
```

(２)
```
int   i = 0;
while(i < 100) {
    処理(省略)
    i++;
}
```

(３)
```
for(int i = 100; i < 0; i--) {
    処理(省略)
}
```

(４)
```
for(int i = 100; i > 0; i++) {
    処理(省略)
}
```

2　以下のプログラムを実行したとき、変数a，bの値を答えよ。

(１)
```
int   a = 0;
for(int i = 0; i < 100; i++) {
    if(i % 3 == 0)
        continue;
    a++;
}
```

(２)
```
int   b = 0, j = 0;
while(1) {
    if(j > 100)
        break;
    b += j++;
}
```

オブジェクト指向とゲームプログラミング

◇　総集編　シフト演算　◇

シフト演算

C言語の演算子には、算術演算子以外に、ビットを直接扱うものがあります。とくに、ビットを直接左右に移動させるシフト演算は、演算の高速化などに効果があります。
シフト演算には、左シフトと右シフトがあります。左シフトはビット列を左に、右シフトはビットを右にずらします。
たとえば、以下のようにビットを左右に移動させることができます。シフト演算では、あふれたビットは、捨てられます。

```
01011001 左に１シフト→ 10110010
01011001 右に３シフト→ 00001011
```

左シフト演算子は「<<」、右シフト演算子は「>>」と記述し、算術演算と同様に扱います。

```
x = x << 2;       // 変数xを左に２シフトし、その結果を変数xに代入する
y = y >> 8;       // 変数yを右に８シフトし、その結果を変数yに代入する
```

以下のように記述することもできます。

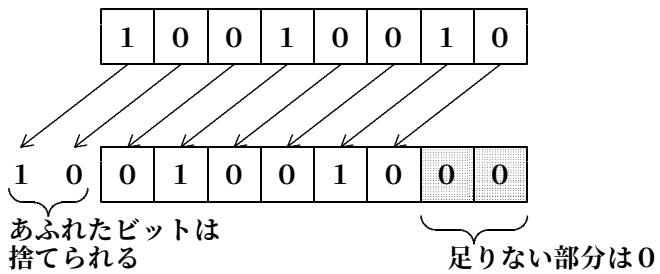
```
x <<= 2;          // 変数xを左に２シフトし、その結果を変数xに代入する
y >>= 8;          // 変数yを右に８シフトし、その結果を変数yに代入する
```

２進数の各桁は、重さが２なので、シフトはある重要な意味を持ちます。左に１つシフトするごとに、値は２倍となり、右に１回シフトするごとに、値は半分(２で割る)になります。

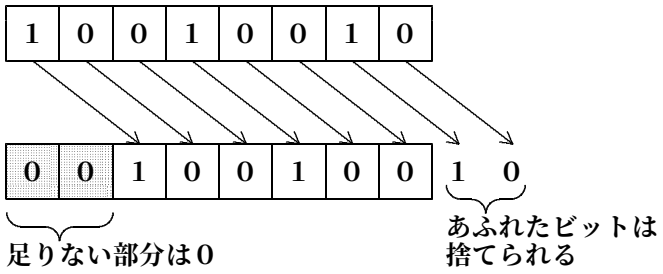
論理シフト

シフト演算には、論理シフトと算術シフトがあります。違いは、符号ビットを考慮するかどうかです。論理シフトの場合は、符号ビットを考慮せず、すべてのビットがシフトされます。あふれたビットは捨てられ、足りない部分は０で埋められます。

・論理左シフト



・論理右シフト



練習問題

1　以下の２進数を16進数に変換しましょう。

（１）$00101110_{(2)}$　　（２）$01111100_{(2)}$　　（３）$10000001_{(2)}$　　（４）$11111110_{(2)}$

2　以下の２進数で表される数値をunsigned char型の変数に代入したとき、実際に代入される値を10進数で答えましょう。

（１）$00101110_{(2)}$　　（２）$01111100_{(2)}$　　（３）$10000001_{(2)}$　　（４）$11111110_{(2)}$

3　以下の１０進数を８桁の２進数に変換しましょう。

（１）３１　　（２）２４６

4　以下のプログラムを実行したとき、変数a，bに代入される値を答えましょう。

```
unsigned char   a =   1 << 2;
unsigned char   b = 127 >> 5;
```