ループ制御

for 文制御

さて、逐次制御（順番に正しく実行する制御）と選択制御（分岐処理）の
二つを使ったプログラムを作れるようになりました
ここでは構造化プログラムのもう一つの制御であるループについて説明します

ループとはその名のとおりプログラムの一定の部分を
ある条件を満たすまで繰り返し実行するという重要な処理です
ループ制御は高度なアルゴリズムやグラフィックスシステムの建築に不可欠な存在なのです

一般的に C 言語プログラマの多くは for ステートメントを使用します
このステートメントは「初期化」「条件」「カウント」を同時に制御してくれます

for ([initializers]; [expression]; [iterators]) statement

initializers には、変数の宣言や初期化を指定します
C 言語ではこの部分で宣言することはできませんでしたが
C# は C++ や Java 言語と同様に宣言することもできます
initializers は、ループを開始する直前に一度だけ実行されます

expression には、ループをするための条件式を指定します
この条件式は statement を実行する前、ループするごとに評価されます
この式が true を返す間 for ステートメントは statement を繰り返し実行します

iterators は繰り返す度に実行するステップで、statement の実行を終了すると評価されます
ここで、expression で使われているカウンタ変数などをインクリメントする方法が一般的です

expression に指定する式はブーリアン型であれば何でも良いのですが
一般的にループ処理は、現在のループ回数を制御するカウンタ変数を用意して使用します
このカウンタ変数の初期化とループごとのカウンタの加算（通常はインクリメント）を
for ステートメントは同時にサポートします

```
class Test {
 static void Main() {
  for (int i = 1; i <= 10 ; i++)
   System.Console.WriteLine(i + "回目のループです");
 }
}
```

このプログラムを見れば、for の基本的な概念を理解することができるでしょう
for ステートメントはカウンタ変数 i をループ開始前に宣言して初期化します
このループは i が 10 と同じかそれ以下の間繰り返しつづけます
for の埋め込みステートメントである WriteLine() を実行すると制御は for に戻り
カウンタ変数 i をインクリメントした後、式を評価し、これを繰り返します

ループするごとに変数 i はインクリメントされ、いつか i は 10 を超えて
for の expression は false を返しループを抜けることになります

initializers や iterators には、ループとは無関係な変数の宣言や演算ができますが
通常は論理的にループと関連性のある処理を行うべき場所であるということは意識するべきです

ただし、注意しなければならないのですが
initializers で変数を宣言した場合は、ループの外からこの変数を見ることはできません
つまり、initializers で宣言した変数は for の埋め込みステートメント内だけで有効です
(この考えは「スコープ」と呼び、スコープについて詳しくは後記します)

```
class Test {
 static void Main() {
  for (int i = 1; i <= 10 ; i++)
   System.Console.WriteLine(i + "回目のループです");
  System.Console.WriteLine("ループは " + i + " 回目で終了しました");
 }
}
```

このプログラムは、for の埋め込みステートメント以外の場所から i を参照しています
しかし、変数 i は for ステートメントでしか宣言されていないためエラーになります

initializers と iterators はカンマ演算子で区切ることによって
同時に複数の変数の宣言や演算を行うことができるようになります
これは、例えばループ内で二つ以上の変数を条件にしたい時などに有効です

```
class Test {
 static void Main() {
  for (int x = 100 , y = 0 ; x != y ; x-- , y++)
   System.Console.WriteLine("x = " + x + " : y = " + y);
 }
}
```

このプログラムはループ内で x と y という二つの変数を扱っています
この二つの変数が同じ値になった時にループから抜けるというものです

また、for の各設定は**省略することも可能**です
条件式を省略した場合は常に true である（つまり無限ループ）ということを意味します
ただし ; は省略できない ので、例えば全てを省略する場合は **for (;;)** という書き方をします

---

while 文制御

for 文は非常に汎用的に使用できるループステートメントなので
多くの C や Java プログラマが好んで使用するステートメントです
しかし、これよりも簡素に記述することができる **while** ステートメントというものもあります

while (expression) statement

expression には、ループ条件となる条件式を指定します
ここが true を返す間 statement を繰り返し実行します
for 文同様に expression はループ毎に評価され false が返されるまで繰り返します
statement には、ループで実行したい埋め込みステートメントを指定します

この文は、for と異なりカウンタ変数の宣言やインクリメントは
statement 内で独自のサポートするなどの対策が必要になります

```
class Test {
 static void Main() {
  int i = 1;
  while(i <= 10) {
   System.Console.WriteLine(i + "回目のループです");
   i++;
  }
 }
}
```

上の for ステートメントで紹介した最初のプログラムのサンプルと
同一の処理を while ステートメントで記述してみたプログラムです
while 文は単純に for 文の for (;expression;) という形と同じであるといえます

---

do/while 文制御

for や while はループの開始前に式を評価します
通常はこれで問題はありませんが、特殊なケースにおいて面倒が発生します
それは、**1度は必ずステートメントを実行したい**時です

for や while 文の場合、式が false を返すと
ループを1度も実行せずに次へ制御を移す可能性があります
式の中に予測できない値があろうとも、1度以上繰り返し処理をする必要がある場合
当然、その場に合わせたアルゴリズムで解決することも可能ですが
アルゴリズムが複雑になったりすると、非効率でソースも読みづらくなります

そのような場合は **do** ステートメントを使用します
これは、**最初にステートメントを実行**し、その後に式を評価します

do statement while (expression);

statement は、ループするステートメントを指定します
expression には、ループの条件となる条件式を指定します

do ステートメントは statement を実行し、その後 expression を評価します
expression が true ならば do に制御を戻し繰り返し statement を実行します
expression が false ならば、ループを抜け出し次の処理へ制御を移します

```
class Test {
 static void Main() {
  int i = 1;
  do {
   System.Console.WriteLine(i + "回目のループです");
   i++;
  } while (i <= 10);
 }
}
```

このプログラムを改良して、例えば i を 100 で初期化してコンパイルして実行してください
それでも、必ず1度はループステートメントが実行されるのを確認できます

一般的には for や while 文が使用され do ステートメントが使用されることはめったにありません
繰り返し処理にどのステートメントを使用するかはプログラマの好みに分かれますが
通常はどの言語でも共通する for を使用することが多いといえるでしょう

---

for

for ([initializers]; [expression]; [iterators]) statement

ステートメントを条件が true の間繰り返します

initializers - ローカル変数宣言、またはステートメント式リストを指定します
expression - ブーリアンを指定します。ここが true の間 statement を実行します

iterators - 反復子です。ステートメント式リストを指定します
statement - expression が true の間実行する埋め込みステートメントを指定します

while

while (expression) statement
ステートメントを条件が true の間繰り返します
カウンタ変数やインクリメントは独自に対応する必要があります

expression - ブーリアンを指定します。ここが true の間 statement を実行します
statement - expression が true の間実行する埋め込みステートメントを指定します

do/while

do statement while (expression);
ステートメントを実行してから式を評価し
式が true であればステートメントの先頭に復帰します

statement - expression が true の間実行する埋め込みステートメントを指定します
expression - ブーリアンを指定します。ここが true であれば statement の先頭に復帰します
