ジャンプ文

ループのフロー制御

ループ制御によってプログラムはより複雑で高度なアルゴリズムを実現できます
しかし、複雑なアルゴリズムの場合は単純にループするだけでは実現できません

ジャンプステートメント と呼ばれるステートメントをループ内で使えば
ループの実行制御をより柔軟に行うことができるようになります
switch 文の各 case の最後に指定するステートメントも、このジャンプステートメントでした

switch で簡単に紹介しましたが、ループを抜け出すには **break** 文を使います
何らかの理由でループを抜け出したい場合、この break を指定して強制的に抜け出します

break;

break には何も指定する値などはありません
switch や ループステートメント内で使用すると、制御を抜け出します
無限ループなどを作ってしまった場合、この break で抜け出すなどの手段があります

```
class Test {
 static void Main() {
  int i = 0;
  while (true) {
   System.Console.WriteLine(i + "回目のループです");
   i++;
   if (i > 10) {
    System.Console.WriteLine("ループを抜け出します");
    break;
   }
  }
 }
}
```

このプログラムの while 文は true を指定しているため無限ループとなっています
プログラムはループ内でカウンタ変数を加算し、if でそれを調べ
指定した条件になればループを抜けるようにしかけています

逆にループの先頭に復帰するには **continue** 文を使います
この文はステートメントの途中でも現在の処理を中断しループの先頭に制御を戻します

continue;

continue も break 同様にとくに指定する値はありません
これ以上現在の処理を実行することに意味がなく、次の繰り返しに移りたい時に使用します

```
class Test {
 static void Main() {
  for (int i = 1 ; i <= 100 ; i++) {
   if ((i % 2) == 1) continue;
   System.Console.WriteLine(i + "回目のループです");
  }
 }
}
```

このプログラムは、カウンタ変数の2の剰余を求め 1 であれば
すなわちカウンタ変数が奇数であれば文字を表示せずに次のループに移行します
この結果、プログラムは偶数回目のループの時のみ文字を表示します

ループのネスト

ループの中に別のループを組み込むという複雑な処理を行うこともできます
これは、高度な配列処理によく用いられる方法です
このような、ある文の中に同じ文を組み込むことを**ネスト**（入れ子）と呼びます

ネストされたループでは、break や continue 文はどのように働くのでしょうか
実は、これらのジャンプステートメントは現在処理中のループに対して働きます

```
while(expression) {
 while(expression) {
  while(expression) {
   break;
  }
  break;
 }
 break
}
```

これは、ネストされた while と break 文の関係を色で表したものです
break は、それぞれ自分の置かれている位置にもっとも近いループに働きます

```
class Test {
 static void Main() {
  for (int i = 1 ; ; i++) {
   for (int j = 1 ; ; j++) {
    System.Console.WriteLine("第二層 " + j);
    if (j >= 5) break;
```

```
    }
    System.Console.WriteLine("第一層" + i);
    if (i >= 5) break;
   }

  }
}
```

ネストされたループを使ったプログラムの例です
break の効果を示すために条件式は省略して無限ループにしています

制御を返す

goto、break、continue 以外に、もう一つジャンプステートメントが存在します
それは、呼び出し元に制御を返す **return** 文です
このステートメントは、厳密にはループではなくメソッドに関連するステートメントです

return [expression];

expression には、呼び出し元に返す適切な値を指定します
このステートメントは**メソッドを終了**させ、呼び出し元に制御を返します
私たちが今扱っているメソッドは Main() メソッドです
Main() メソッド内で return を呼び出すと、その制御はこのプログラムを呼び出した場所に帰ります
この管理は、私たちではなく OS の仕事になります

私たちが今扱っている Main() メソッドは値を返すものではありません
そのため、単純に return と記述すればその時点でプログラムは終了します

```
class Test {
 static void Main() {
  int x = 0;
  switch (x) {
  case 0:
   System.Console.WriteLine("Kitty on your lap");
   return;
  case 1:
   System.Console.WriteLine("Tokyo mew mew");
   return;
  }
  System.Console.WriteLine("Di Gi Charat");
 }
}
```

switch のいずれかの case ブロックが実行されると
文字列を出力してそのままメソッドを終了させます
break であれば、switch 後の WriteLine() メソッドが実行されますが
return を指定して Main() メソッドを終了させるため、最後の WriteLine() は実行されません

---

break

現在の制御を抜け出して次の処理へ移行します

continue

現在の処理を中断し、ループの先頭に復帰します

return

return [expression];

現在のメソッドを終了し、呼び出し元に制御を返します

expression - 呼び出し元に返す適切な戻り値を指定します

前のページへ　　戻る　　次のページへ