無条件ジャンプ

ラベルとgoto

これまでプログラムは上から下へ逐次実行してきました
本来の手続き型言語のプログラムは「逐次」「判断」「繰り返し」で制御できますが
場合によっては何らかの要因によって無条件に指定位置に制御を移したい場合があります

C# はステートメントにラベルプレフィックスをつけることができます
ラベルはステートメントに識別子を与えたものです

identifier : statement

identifier にはラベルの識別子を指定し、statement には名前をつけるステートメントを指定します
無条件ジャンプを行う場合、このラベルを識別子に用いてプログラムの制御を移します
無条件にプログラムの流れを変えるには goto ステートメントを使用します

goto identifier;

identifier には、ジャンプ先のステートメントのラベルを指定します
goto はプログラムの流れを無視して無条件に指定ラベルへ移行します

```
class Test {
 static void Main() {
  int count = 0;

  start: count++;
  if (count > 100) goto end;
  System.Console.WriteLine("count = " + count);
  goto start;

  end: System.Console.WriteLine("End of goto loop");
 }
}
```

これは、start ラベルから goto start までの間
count が 100 になるまで繰り返すループを goto で再現したプログラムです
実際には、後記する専用のループ処理ステートメントがあるため
goto ステートメントでこのような処理を記述することは批難にあたる行為です

goto は無条件ジャンプであり、基本的に障害がないように思えますが
実はブロック内にはジャンプできないという条件があります
ブロック内で宣言されたラベルは、ブロックの外からは見えないという規約があるからです
（この概念を「スコープ」と呼びます。後ほど詳しく紹介します）

```
class Test {
 static void Main() {
  goto inner;
  {
   inner: System.Console.WriteLine("Kitty on your lap");
  }
 }
}
```

このプログラムの goto はブロックの内部の inner ラベルにジャンプしようとしていますが
ブロックの外からはラベルが見えないため、コンパイルエラーが発生します

逆に、ブロック内からブロックの外へのジャンプは可能です

```
class Test {
 static void Main() {
  int count = 0;
  start: {
   count++;
   if (count > 100) goto end;
   System.Console.WriteLine("count = " + count);

   goto start;
  }
  end: System.Console.WriteLine("End of goto loop");
 }
}
```

さっきのプログラムの start ラベルを単一のブロックステートメントにまとめたプログラムです
ブロックの内部から外部のラベルにジャンプしていることがわかりますね

因みに、ラベルは独自の宣言空間を持っています
変数やクラス名などと名前が重複しても問題はありません
例えば、次のようなプログラムの一部分があったとしても問題はありません

```
int label;
goto label;
label: label = 10;
```

このような可読性の低い識別子をつけるべきではありませんが
ラベルは独自の宣言空間を持つため、変数 label と衝突することはないです

goto はプログラムの状況に関係なく無条件にジャンプします
しかし、現代の構造化プログラムの時代においてgoto は禁句といってもよい
これは、1968年に Dijkstra という人物が goto の問題点を指摘した本を出版したことからはじまり
現代では goto のような無条件分岐はプログラムの制御の追跡を困難にすると考え
多くのプログラマが goto を使わないgoto レスプログラムを信仰しています

---

switch と goto

これまで、C 言語などでも goto は存在していましたが
多くのプログラマは使用することはありませんでしたし、Java ではその姿を消しています

しかし C# では goto は無条件分岐以外に重要な利用方法があります
それは switch 文でフォールスルーを行いたい場合です
C# では goto 文を使ってジャンプする case を選択することができるのです

goto case constant-expression;
goto default;

case をつけた constant-expression を指定する goto 文は
constant-expression と同じ定数式を持つ case にジャンプします
goto default は switch 文の default 区にジャンプすることができます

goto case や goto default は switch 文に包含されている必要があります
また、constant-expression と switch の制御型に互換性がない場合や
指定した定数式の case ラベルが存在しない場合もエラーになります

```
class Test {
 static void Main() {
  int x = 0;
  string str = "";

  switch(x) {
  case 0:
   str = "Kitty on your lap¥n";
   goto case 1;
  case 1:
   str += "Tokyo mew mew¥n";
   goto case 2;
  case 2:
   str += "Di Gi Charat";
   goto default;
  default:
   System.Console.WriteLine(str);
   break;
  }
 }
}
```

このプログラムは、case 0 が実行され、さらに下の case 文にフォールスルーしていきます
最終的には全ての case と default を実行してプログラムは終了します
goto case の定数を変更することで、移行する順番を柔軟に変更するという威力を持ちます

前のページへ　戻る　次のページへ