ブーリアン

比較演算

リテラル及び変数の型にはと呼ばれる特殊な論理値があります
ブーリアンは、true（真）または false（偽）のいずれかの定数を持ちます

C 言語においてブーリアンは数値型として表現してきました
C++ では true 及び false という定数が標準で定義されていますが、実装レベルは同じです
しかし C# では Java のようにブーリアン型として独立しています

ブーリアン型変数は bool で宣言された変数で
このリテラルや変数は条件分岐で使用され、プログラムの流れに影響します

```
class Test {
 static void Main() {
  bool bl = true;
  System.Console.WriteLine(bl);
  bl = false;
  System.Console.WriteLine(bl);
 }
}
```

これがブーリアンリテラル及び変数の実体です
コンソールにブーリアンの文字列表現 True と False が表示されます

ブーリアンは他の型と異なり数値表現がないので算術演算はできません
したがって、**比較演算**などを行いその結果を true か flase で格納したります
このようなブーリアンを返す演算を**関係演算**と総称します

演算子	説明
==	等しければ true
!=	等しくなければ true
<	小さければ true
>	大きければ true
<=	同じか小さければ true
>=	同じか大きければ true

これを利用すれば、数値を比較してその結果を得ることができ
これは後ほど紹介する条件分岐などに利用することができるのです

```
class Test {
 static void Main() {
  int var = 100;
  System.Console.WriteLine("var == 500 : " + (var == 500));
  System.Console.WriteLine("var != 500 : " + (var != 500));
  System.Console.WriteLine("var < 500 : " + (var < 500));
  System.Console.WriteLine("var <= 100 : " + (var <= 100));
  System.Console.WriteLine("var > 100 : " + (var > 100));
  System.Console.WriteLine("var >= 500 : " + (var >= 500));
 }
}
```

それぞれの比較演算の結果がコンソールに出力されます

C# は C言語と異なり文字列を string というデータ型で保有します
（Cは、連続した char 型の配列として文字列を表現します）
C# は文字列も == と != 演算子で比較演算することが可能な言語です

文字列が等しいと判断される条件は同じ長さ、同じ位置に同じ文字が格納されいる状態
つまり論理的に文字列が等しければ等しいとされます。直感的でわかりやすいですね
また、双方が null と呼ばれる特殊なリテラルの場合も等しいと判断されます

```
class Test {
 static void Main() {
  string str1 = "Kitty on your lap";
  string str2 = "Kitty on your lap";
  System.Console.WriteLine(str1 == str2);
 }
}
```

二つの文字列型変数は Kitty on your lap という文字列を保有しています
これらは文字の長さもそれに位置する文字も全て同じなので true を返します

ブール論理演算

前回の論理演算はブーリアンにも適応します
ブーリアンは true と false しかありませんが
true を 1、false を 0 として前回の関係と照らし合わせるとよいでしょう

例えば true & false は false になり、true | false は true というようになります
関係演算の中で「双方が真ならば」とか「一方でも真ならば」というような処理に使用します

```
class Test {
 static void Main() {
  int x = 100;
  System.Console.WriteLine((x < 200) & (x > 50));
  System.Console.WriteLine((x == 50) | (x == 200));
 }
}
```

このプログラムは変数 x に対して複数の条件で関係演算を行っています
最初は x が 200 より小さく、**かつ** 50 よりも大きいといか？
次は x が 50 と等しい、**または** 200 と等しいか？という演算です

また、ブーリアンの論理否定は **!** 演算子を使用します
この演算子は右辺のオペランドが true なら false に、false なら true にします

! expr

expr には否定する論理値をあらわす式を指定します
これを用いれば、関係演算の中で排他的論理和などを表現できます

```
class Test {
 static void Main() {
  bool var1 = true , var2 = false;
  System.Console.WriteLine((var1 | var2) & !(var1 & var2));
  System.Console.WriteLine((var1 | var1) & !(var1 & var1));
  System.Console.WriteLine((var2 | var2) & !(var2 & var2));
 }
}
```

このプログラムは、変数 var1 と var2 を定義し
それらの **論理和と否定論理積の論理積** を求めます
こうすることで、関係演算の中で XOR 排他的論理和を求めることができます

このように、否定演算子を用いることで否定論理和や否定論理積を作ることができます
もちろん、他だ単純にブーリアンを反転させる否定として使うことも多いです

条件付き論理演算子

ブーリアンの論理演算の場合は必ずしも両方のオペランドを評価する必要がない場合があります
例えば、論理積を求めるのに左オペランドが false ならば右オペランドが何であれ
必ず式の評価が false になることは間違いありません

論理和も同様に、左オペランドが true ならば評価は右オペランドに関係なく ture です
この場合、右オペランドは評価する必要がないならば評価しないほうが
当然、実行する命令ステップが減少するためプログラムは高速になります

これを可能にする演算子が**条件付き論理演算子**と呼ばれるものです
条件付論理積は **&&**、条件付論理和は **||** を使用します

expr1 && expr2
expr1 || expr2

expr1 と expr2 には、それぞれ論理積、論理和を求めたい式を指定します
これらは、左辺のオペランドを最初に評価し
左辺だけで評価できれば場合は即座に結果を返し、右オペランドを評価しません
左辺だけでは判定できない場合は右辺も評価します
それ以外は、通常の論理積、論理和と変わりはありません

```
class Test {
 static void Main() {
  int var = 0;
  System.Console.WriteLine(false && 5 < (var = 10));
  System.Console.WriteLine(true || 5 < (var = 10));
  System.Console.WriteLine(var);
 }
}
```

このプログラムでは、論理積や論理和を求める際に
右オペランドで変数 var に値を代入してから比較演算を行います
しかし、左オペランドだけで評価できる場合、右オペランドは評価されません
このプログラムを実行すると、次の結果が得られます

False
True
0

右の代入演算が行われていないことが、最後の 0 で確認できます
論理演算子を & や | に変えたり、左オペランドのリテラルを反転させると
右オペランドが評価され var に値 10 が格納されることを確認できます
