論理演算

論理積

プログラムで詳細にビットを操作する場合は、算術演算子ではなく
ビット単位で演算する**論理演算**を使用します

論理積 AND は、二つのオペランドのビット列を比較し
双方が 1 であれば 1 を、一方でも 0 であれば 0 を出力します

1 AND 1 = 1
0 AND 1 = 0
1 AND 0 = 0
0 AND 0 = 0

これが論理積と呼ばれる演算です
C# では、論理積は **&** 演算子を使います

expr1 & expr2

expr1 と expr2 にはそれぞれ論理積を求める式を指定します
& はオペランドを二つ受け取る2項演算子で、双方のオペランドの論理積を返します
二つのオペランドは、各ビット（桁）が論理積で比較されます

```
class Test {
 static void Main() {
  int var = 0x3C7C;
  System.Console.WriteLine(var & 0xF0F0);
 }
}
```

このプログラムは、変数 var と 0xF0F0 の論理積を出力します
論理積はビットの**フィルタリング**に使用することができます

0011 1100 0111 1100
1111 0000 1111 0000

0011 0000 0111 0000

上のプログラムの論理積は、このように計算されています
右辺のオペランドで 0xF0F0 というリテラルを与えることによって
F は全てのビットが 1 なので、相手のビット列をそのまま出力しますが
0 は全てのビットが 0 なので、相手のビット列に関係なく 0 が出力されます
この性質を利用して、論理積で特定のビットを削除したりすることができます

論理和

論理積の他に、代表的な論理演算が **論理和 OR** です
論理和は、二つのビットを比較し一方でも 1 ならば 1 を出力します
逆に言うと双方が 0 の時のみ 0 を出力するのが論理和です

1 OR 1 = 1
1 OR 0 = 1
0 OR 1 = 1
0 OR 0 = 0

これが論理和による演算です。論理和は | 演算子を使います

expr1 | expr2

expr1 と expr2 にはそれぞれ論理和を求める式を指定します
& 演算子同様に二つのオペランドの論理和を返します

```
class Test {
 static void Main() {
  byte a = 1 , b = 2 , c = 4 , d = 8;
  int x = a | d , y = b | c;
  System.Console.WriteLine("a | d = " + x);
  System.Console.WriteLine("b | c = " + y);
 }
}
```

このプログラムは各種ビットに意味を持たせたとする a ～ d の変数を用意しています
変数 x と y は意味を持った数値を複数組み合わせてシステムに要求するという場合
このように論理和を使ってビットをセットするという方法が頻繁に用いられます
変数 x に代入する時の a | d の論理和の動きを見てみましょう

0000 0001
0000 1000

0000 1001

答えは 9 になりますが、加算ではなく論理和で求めていることが重要です
加算は a + a とすると値が増えつづけ意味のない値になりますが
論理和であれば意味付けされている定数からはみ出すようなことはありません

論理否定

整数を単純に否定する場合は、リテラルや数値変数の前に - 単項演算子を付加しました
これは2の補数が求められ 0 からオペランドを引いた結果が得られました

論理否定 NOT は全てのビットを単純に反転させます
これによって1の補数を求めたり、与えられてビット列を否定できます

NOT 1 = 0
NOT 0 = 1

NOT はビットを反転させるだけなのでオペランドは1つしか要求しません
論理否定は **~** 演算子を使います

~ expr

expr にはビットを反転させたい式を指定します
ビットを反転させるので ~ 演算子で1の補数を求めることができます

```
class Test {
 static void Main() {
  sbyte var = 0x07;
  System.Console.WriteLine(~var + 1);
 }
}
```

以前、コンピュータは負数を2の補数で表現すると説明しました
これは、減算処理を加算処理で行えるというメリットがあるからです

このプログラムでは ~ 演算子で var のビットを反転させ1の補数を求め
これに1加算することによって2の補数、つまり var の負数を表現しています

排他的論理和

プログラム処理の中では、複数の選択の中から常に一つだけを選択するという
排他的な制御がまれに見かけられます

排他的論理和 XOR は、双方のビットが同じ場合は 0、異なる場合は 1 を出力します
すなわちビットが排他的であれば 1 となる論理演算なのです

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

論理和は一方でも 1 ならば 1 を返しましたが
排他的論理和は一方が 1 の時のみ 1 を返すという仕様になっています
排他的論理和は **^** 演算子を使います

expr1 ^ expr2

expr1 と expr2 には排他的論理和を求める式を指定します

```
class Test {
 static void Main() {
  int var = 0xA5;
  System.Console.WriteLine(var ^ 0xF0);
 }
}
```

このプログラムは、排他的論理はを使って**部分否定**しているプログラムです
論理否定はビット全体を否定しましたが、排他的論理和を使えば部分否定ができます

1010 0101
1111 0000

0101 0101

2進数 0101 0101 は10進数 85 なのでこれが出力されたはずです
上位4ビットだけが部分的に否定されていることが確認できますね
