算術演算子

計算

プログラムの中心は変数やリテラルなどの演算にあります
必要な情報を集め、目的の値を算出するために様々な計算を行います

プログラムで変数やリテラルの計算を行うには**算術演算子**を使用します
算術演算子は + や - などの数学で使用する記号と同じなので直感的にわかりやすいです
ただし、乗算は *、除算は /、剰余は %、で求めます

演算子は通常1つ以上の**オペランド**を持ちます
オペランドは変数やリテラルなどの操作される対象の値のことを指します
演算子とオペランドからなりたつ文を**式**と呼びます

演算子	説明
+	加算、文字列の結合 オペランドの単純な値（単項演算子）
-	減算 オペランドの符号否定（単項演算子）
*	乗算
/	除算
%	剰余

オペランドを一つしか要求しない演算子を**単項演算子**と呼び
オペランドを二つ要求する演算子を**2項演算子**と呼びます
算術演算子は全てが2項演算子で、右辺と左辺にそれぞれ計算する式を指定します
ただし、符号指定で + と - 演算子は単項演算子として使うこともありましたね

演算子には**計算優先順位**があり、算術演算子は数学と同じ順序を持ちます
つまり、乗除さんは加減算よりも先に演算されます
さらに同一の算術演算子が多項式に複数ある場合は左から計算される決まりになっています
これを**左結合**と呼びます
計算の優先順位を変えたい場合は、数学同様に ( ) で囲みます

```
class Test {
 static void Main() {
  System.Console.WriteLine(10 + 5 * 2);
  System.Console.WriteLine((10 + 5) * 2);
 }
}
```

この結果は 20 と 30 の二つの結果が表示されます
最初の演算は 5 * 2 が最初に計算され、次に 10 + 10 が計算されます
これは、数学同様に乗除算が優先的に計算されるからです
次は 10 + 5 を ( ) で囲むことで計算順位を優先化させることによって
15 * 2 という計算になり最終的に 30 という結果を表示するようになっています

演算の結果はメソッドなどに渡すか、変数に代入するなどの受け皿が必要です
演算の結果新しい値が生成されない場合、これは開発者のミス以外のなにものでもないからです
（演算の結果、何らかの値が変化しないのならばその意味が無い）

代入演算子は全ての演算子の中で**最も優先順位が低い**ので
最終的に値を代入する変数はもっとも左辺に配置し、演算式を右辺に配置します

```
class Test {
 static void Main() {
  double a1 = 0.7 , a2 = 0.4 , x;
  x = 1.0 - (1.0 - a1) * (1.0 - a2);
  System.Console.WriteLine(x);
 }
}
```

例えば、稼動率 70% と 40% の二つのシステムを並列に運用し
一方でも正常に動作していればシステム全体が運用できるという場合の稼動率の演算です
この二つの並列システムの稼動率 82% という結果を変数 x に格納しています

例えば、これを利用すればさらに別の並列システムとの稼動率の演算に x を使うことができます
このように、演算の結果を変数に残すことで別の演算にその変数を使うこともできます

また、+ 演算子を文字列に用いると文字列を結合させることができます
本来文字列の結合は値の結合のそれとは処理の上でまったく異なりますが
C# コンパイラは Java 同様に文字列を + 演算子で結合して返すことができます
一辺、または両辺が文字列の時は両辺を文字列として結合します

```
class Test {
 static void Main() {
  string str = "Kitty on your lap";
  str = str + "¥nTokyo mew mew";
  System.Console.WriteLine(str + 1999);
 }
}
```

この結果は次のようになります

Kitty on your lap
Tokyo mew mew1999

最初に str に代入した文字列にさらに別の文字列を結合し
WriteLine() で 1999 という数値を文字列として結合して出力しています
文字列と数値を加算することで数値を文字列に自動変換させることができるので
単純に数値を文字列に変換したい時も ""と空文字を使って実現できます

因みに、算術演算は左から結合する左結合と書きましたが
例えば代入演算子は**右結合**なので注意する必要があります
代入演算子も連続して記述することができ、これを**多重代入**とも呼びます

variable1 = variable2 = variable3 = ...

この場合は、代入演算は右結合なので variable3 から処理されます
複数の変数を同時に同じ値に初期化する時などはこれを使用します

```
class Test {
 static void Main() {
  int var1 , var2 , var3;
  var1 = var2 = var3 = 100;

  System.Console.WriteLine("var1 = " + var1);
  System.Console.WriteLine("var2 = " + var2);
  System.Console.WriteLine("var3 = " + var3);
 }
}
```

全ての変数は 100 を表示します

2項数値昇格

式の中の計算は通常は同じデータ型どうしで行われるべきですが
時には異なるデータ型変数による演算が生じることもあります

異なる型どうしの演算はできないため、コンパイラはこのような演算を発見すると
一方のオペランドを暗黙に変換し共通の型にしてから演算します
この時の変換には必ず変換規則が割り当てられるので、プログラマはそれを知っておく必要があります

異なる型どうしが演算しようとすると、二つのオペランドのうち
サイズが**大きいほうのオペランド型**に他方を変換するのが一般的です
これは、サイズの昇格はデータを破壊することがないからです
逆にサイズを縮小すると、上位ビットを切り捨てることでデータが破壊される可能性があるため
少なくともプログラマが明示しない限り暗黙に変換するべきではないという理論があります

一方のオペランドが型が **decimal** であるか？ **YES→** もう一方のオペランドを **decimal** に変換 float、double ならエラー
↓**NO**
一方のオペランドが型が **double** であるか？ **YES→** もう一方のオペランドを **double** に変換
↓**NO**
一方のオペランドが型が **float** であるか？ **YES→** もう一方のオペランドを **float** に変換
↓**NO**
一方のオペランドが型が **ulong** であるか？ **YES→** もう一方のオペランドを **ulong** に変換 sbyte、short int、long ならエラー
↓**NO**
一方のオペランドが型が **long** であるか？ **YES→** もう一方のオペランドを **long** に変換
↓**NO**
一方のオペランドが型が **uint** であるか？ **YES→** もう一方のオペランドを **uint** に変換 sbyte、short、int なら **両辺を long** に変換
↓**NO**
両辺を int に変換

これが、C#の2項数値昇格のマップです
異なるデータ型の演算を行う場合はこれに注意する必要があります

```
class Test {
 static void Main() {
  double var1 = 3.24;
  int var2 = 5;
  System.Console.WriteLine(var1 + var2);
 }
```

```
}
```

このプログラムでは bouble 型と int 型の変数を加算していますが
2項数値昇格アルゴリズムのおかげでデータが壊れることなく予想した結果を得られます
int 型の var2 が double 型に昇格してその上で加算処理されていることがわかります

複合代入演算子

代入演算において、代入する変数と何らかの式が計算対象の場合
通常の代入演算子を使用することもできますが
これよりも**複合代入演算子**を使用することが推奨されます

通常の演算は、複数の式と演算の結果を変数に代入するものですが
場合によっては、この演算対象に代入する変数が関わることがあります
例えば、次のような計算などは演算に代入する変数も使用しています

variable = variable + expression...

variable に expression を加算したものを variable に代入する計算です
もちろんこれでも何の問題もないのですが
このような場合は複合代入演算を使うべきです

複合代入演算子は op= という形をとります
op には、何らかの演算子を指定します

variable op = expression

variable には変数、expression には variable の演算対象となる式を指定します
例えば variable に 4 を乗算したい場合 variable *= 4 と記述します

```
class Test {
 static void Main() {
  string str = "Kitty on your lap";
  str += "¥nTokyo mew mew";
  System.Console.WriteLine(str);
 }
}
```

このプログラムを実行すると、Kitty on your lap と Tokyo mew mew という文字が表示されます
この結果から、複合代入演算子によって str に文字列が結合したことがわかります

前のページへ　戻る　次のページへ