リテラル

数値

コンピュータは計算が得意であるということは、誰もが知ってる常識です
当然、プログラムのソースの大部分は計算と制御で埋め尽くされるでしょう

プログラムでは、数値や文字列を直接記述したものを**リテラル**と呼びます
データには様々な**型**があり、代表どころでは整数、浮動小数、文字などがあります
数値を使った演算は主に整数型のリテラルで表現することができます

C#の整数型は、0 ～ 9 を使った10進数と、0 ～ F を使った16進数を使用できます
10進数はそのまま、16進数の場合は数値の前に **0x** プリフィックスを付加する必要があります
なお、16進数のA～Fは小文字で指定しても問題はありません

```
class Test {
 static void Main() {
  System.Console.WriteLine(1983);
  System.Console.WriteLine("-ビル・ゲイツが Windows を発表-");

  System.Console.WriteLine(0x7C1);
  System.Console.WriteLine("-Microsoft が Windows をリリース-");
 }
}
```

このプログラムでは WriteLine() メソッドに 1983 という10進整数値と
7C1 という16進整数値（10進数の 1985)を渡しています
WriteLine() メソッドは、この数値を文字列に変換しコンソールに出力します

ただし、16進数も10進数も機械語では2進数なので
これらはコンパイラが扱うものであり、機械語レベルでは同じ扱いです
そのため、文字列に変換して表示した時は10進数で表示されてしまいます

実数値

プログラムが小数も扱う場合は、整数型ではなく実数型を使用します
実数リテラルは整数と違い**10進数しか扱えない**ので注意してください

実数型は、10進整数部と小数以下の10進指数部からなります
プログラムでは小数を**浮動小数点**と呼ばれる方法で表現します
基本的には、数学同様に **整数部.小数部** という形で表せます
ただし浮動小数は、アンダーフローや丸め誤差と呼ばれる誤差が発生するので
その扱いにはある程度のコンピュータ科学知識が必要になることがあります

実数型は、指数表記 e または E を使用することができます
指数表記は10の累乗を表すもので 4e3 で 4000 というように表すことができます

```
class Test {
 static void Main() {
  System.Console.WriteLine(42.195);
  System.Console.WriteLine(0.3e-2);
 }
}
```

このプログラムも、実数を問題なくコンソールに出力してくれることでしょう

文字と文字列

C# の文字は様々な国の言語を表現できる国際的な Unicode を使用しています
これは、近年 Java や Windows NT などにも積極的にとり入れられている重要な文字コードで
アプリケーション開発の国際化の時代に非常に貢献している技術です

Unicode は 2 バイト文字であり、C# は1文字に2バイトの容量を使用します
これは、C/C++ の ASCII に比べて大きく異なる点の一つです

文字は**シングルクォーテーション '**で囲むことで表現できます
文字列と違い、文字型は2バイトまでと定められているため1文字しか表せません

```
class Test {
 static void Main() {
  System.Console.WriteLine('A');
  System.Console.WriteLine('1');
 }
}
```

このプログラムは WriteLine() メソッドに文字を渡して表示します
注意してほしいのですが '1' というのは数値ではなく文字型としての 1 です
表示された時は 1 としかありませんが、データ型が違うというのは重要です
この後に紹介する計算などの時には、データ型の違いは重要な項目になります

文字列は、前回紹介したようにダブルクォーテーション " で囲むことで表現できました
しかし、改行などは文字列の中で表現できませんでした
改行やタブのような特殊な文字は**エスケープ文字 ¥**と呼ばれる文字で表現します

エスケープ文字は ¥ で始まりその後に何らかの文字などを付加します
これを**エスケープシーケンス**と呼び、次のものがあります

エスケープシーケンス	説明	Unicode
¥'	シングルクォーテーション	0x0027
¥"	ダブルクォーテーション	0x0022
¥¥	円マーク	0x005C
¥0	null	0x0000
¥a	アラート	0x0007
¥b	バックスペース	0x0008
¥f	改ページ	0x000C
¥n	改行	0x000A
¥r	キャリッジリターン	0x000D
¥t	水平タブ	0x0009
¥v	垂直タブ	0x000B
¥x16進数	16進エスケープシーケンス	

これらのエスケープシーケンスを文字、または文字列に含ませて出力すると
それぞれのエスケープシーケンスの意味する文字を出力することができます
これを使用すれば、文字列内に " を表示させることが出きるようになります

ただし、文字列において ¥ に続く文字は上のいずれかのエスケープシーケンスである必要があります

```
class Test {
 static void Main() {
  System.Console.WriteLine("Kitty on your lap¥nTokyo mew mew");
  System.Console.WriteLine("¥"Nekoneko Zoo¥"");
  System.Console.WriteLine("¥x005C¥x0009¥x005C");
 }
}
```

これをコンパイルして実行すると、次のようになります

```
Kitty on your lap
Tokyo mew mew
"Nekoneko Zoo"
¥       ¥
```

このように、改行コードやタブをエスケープシーケンスで表現できます
これは C/C++ の経験者であればお馴染みのものでしょう

C# は、上のように "" で囲まれた文字列を**標準文字列**と呼びます
これに対し、**逐語的文字列**と呼ばれる便利な文字列表現もあります

逐語的文字列は、標準文字列と違い " 以外の全ての文字を文字列内に指定できます
標準文字列では文字列中に改行することはできませんでしたが
逐語的文字列はタブ、改行、エスケープ文字などが全てそのまま解釈されます
ただし、逐語的文字列内で " を表現したい場合は ""と記述します

逐語的文字列は **@" ではじまり " で終わり**ます

```
class Test {
 static void Main() {
  System.Console.WriteLine(
@"¥Kitty on your lap¥
¥n Tokyo mew mew ¥t
""Di Gi Charat"""
  );
 }
}
```

これを実行すると、次のような文字列が出力されます

```
¥Kitty on your lap¥
¥n Tokyo mew mew ¥t
"Di Gi Charat"
```

これを見れば、エスケープ文字は解釈されずにそのまま解釈され
タブや改行も表現されていることが確認できます

前のページへ　戻る　次のページへ