講義メモ
・p.52「type07.cs」から

p.50 decimal型（再掲載）

・decimalは10進数のことで、実数を内部的2進数で扱うfloat/double型に比べて、誤差が出づらい
・ただし、1個に128ビット用いるのでメモリの消費量に注意
・初期化に用いる実数値も同じ扱いをする必要があるので、末尾にMまたはmを付けて示す必要がある
　例：decimal d = 3.14M; //decimal型扱いの値で、decimal型の変数を初期化
・扱える値の範囲はdecimal.MinValueからdecimal.MaxValueで、float型より狭いが、long型より広い
・符号なしdecimal型は無い

```
//p.52 type07.cs
using System;
class type07
{
    public static void Main()
    {
        decimal total; //decimal型の変数の定義
        Console.Write("借入金額---");
        decimal a = decimal.Parse(Console.ReadLine()); //string型⇒decimal型変換
        Console.Write("利息(%)---");
        decimal p = decimal.Parse(Console.ReadLine()); //string型⇒decimal型変換
        decimal r = p / 100M; //decimal型の変数の演算なので100もdecimal型扱いにする
        total = a * (1m + r); //decimal型の変数の演算なので1もdecimal型扱いにする(1Mで
もOK)
        Console.WriteLine("1期間後の元利合計は{0:c}です", total); //通貨書式で表示
        a = total; //利息を加えた結果を次の元金とする
        total = a * (1m + r); //decimal型の変数の演算なので1もdecimal型扱いにする(1Mで
もOK)
        Console.WriteLine("2期間後の元利合計は{0:c}です", total); //通貨書式で表示
        a = total; //利息を加えた結果を次の元金とする
        total = a * (1m + r); //decimal型の変数の演算なので1もdecimal型扱いにする(1Mで
もOK)
        Console.WriteLine("3期間後の元利合計は{0:c}です", total); //通貨書式で表示
        a = total; //利息を加えた結果を次の元金とする
        total = a * (1m + r); //decimal型の変数の演算なので1もdecimal型扱いにする(1Mで
もOK)
        Console.WriteLine("4期間後の元利合計は{0:c}です", total); //通貨書式で表示
    }
}
```

p.53 文字型

・char型：1文字分の情報＝Unicodeなので2バイト(16ビット)を持つデータ型
・文字リテラル：プログラム中で文字を記述するもの。'(シングルコーテーション)で囲む。
・初期化の書式：char 変数名 = '文字'; //例：char c = '猫';(C#は、ひらがな、漢字でもOK)
・Console.Write/WriteLineは文字型にも対応しているので、文字や文字型の変数をそのまま指定できる。

p.54 type08.cs

```
// type08.cs
using System;
```

```
class type08
{
    public static void Main()
    {
        char a = '猫', b = 'で', c = 'わ', d = 'か', e = 'る',
            f = 'C', g = '#', h = 'プ', i = '口', j = 'グ',
            k = 'ラ', l = 'ミ', m = 'ン', n = 'グ'; //文字型の変数の初期化
        Console.Write(a); //改行せずに表示
        Console.Write(b);
        Console.Write(c);
        Console.Write(d);
        Console.Write(e);
        Console.Write(f);
        Console.Write(g);
        Console.Write(h);
        Console.Write(i);
        Console.Write(j);
        Console.Write(k);
        Console.Write(l);
        Console.Write(m);
        Console.Write(n);
        Console.WriteLine(); //改行
    }
}
```

p.54（Unicode番号による指定）

・文字リテラルにおいて、\uで始まる16進数を'（シングルコーテーション）で囲むことで、Unicode番号による指定が可能
・例：char c = '猫'; ⇒ 例：char c = '\u732B'; //「猫」の文字コードは732B
・なお、「\u」の代わりに16進数を意味する「\x」をつけても同じ結果になる
・また、10進数に変換した結果の前に「(char)」をつけても同じ結果になる（シングルコーテーションは不要）
・この「(型)」をキャストといい、型変換をサポートとする。文字はOKだが、文字列には非対応なのでparseを用いる
・例：char c = '猫'; ⇒ 例：char c = (char)29483; //「猫」の文字コードは732B=10進数29483
・Console.Write/WriteLineのフォーマット指定子(p.28)は文字型にも対応しているので、type08.csのように1文字ずる列記する必要はない

p.54 type09.cs

```
//p.54 type09.cs
using System;
class type09
{
    public static void Main()
    {
        char a = '\u732B';    //Unicode指定
        char b = '\x3067';    //16進数指定
        char c = 'も';        //文字リテラル
        char d = (char)12431; //10進数指定
        char e = '\u304B';    //Unicode指定
        char f = '\x308B';    //16進数指定
        Console.WriteLine("{0}{1}{2}{3}{4}{5}", a, b, c, d, e, f);
```

```
    }
}
```

アレンジ演習：type08a.cs

・type09.csを参考にして、フォーマット指定子を用いて短くしよう
　※「猫でもわかる」までOK

作成例

```
//アレンジ演習：type08.cs
using System;
class type08
{
    public static void Main()
    {
        char a = '猫', b = 'で', c = 'も', d = 'わ', e = 'か', f = 'る'; //文字型の
変数の初期化
        Console.WriteLine("{0}{1}{2}{3}{4}{5}", a, b, c, d, e, f);
    }
}
```

p.55 エスケープ文字

・すでに学習した改行を示す「\n」などは文字と同じ扱いのエスケープ文字とされる
・シングルクォーテーションや、ダブルクォーテーション、円マークなどを文字として用いる場合は「\"」「\'」「\\」を用いると良い
・文字列の中に「\n」を挿入して、表示時に改行させることできる

p.56 escape01.cs

```
//p.56 escape01.cs
using System;
class escape01
{
    public static void Main()
    {
        char n = '\n'; //文字変数を改行文字で初期化
        string str1 = "今日は"; //文字列変数
        string str2 = "よい天気です"; //〃
        Console.WriteLine(str1 + n + str2); //文字列と改行文字を連結すると文字列になる
        string str3 = "今日は\nよい天気です"; //途中に改行文字のある文字列
        Console.WriteLine(str3); //表示すると途中で改行する
    }
}
```

アレンジ演習：p.56 escape01a.cs

・1行追加して、文字列「"Let's Go"は\1000です」がこの通り表示できるように指定しよう

作成例

```
//アレンジ演習：p.56 escape01a.cs
using System;
```

```
class escape01
{
    public static void Main()
    {
        char n = '\n'; //文字変数を改行文字で初期化
        string str1 = "今日は"; //文字列変数
        string str2 = "よい天気です"; //〃
        Console.WriteLine(str1 + n + str2); //文字列と改行文字を連結すると文字列になる
        string str3 = "今日は\nよい天気です"; //途中に改行文字のある文字列
        Console.WriteLine(str3); //表示すると途中で改行する
        Console.WriteLine("\"Let's Go\"は\\1000です"); //【追加】「\"」「\\」が必要
    }
}
```

p.57 論理型

・C#はC言語とは異なり、真理値（真偽値）を表すための型としてbool型が提供されている
・また、bool型の変数に代入したり初期化などに利用できる論理型リテラルとして,true(真)、false(偽)が提供されている
・例: bool flag = true; //フラグをオンにする(立てる)
・なお、これに伴い、C/C++で可能な「非0を真とし、0を偽とする」ことはC#では禁止。
・bool型のNET型(p.42)は「System.Boolean」
・C#が提供する「変数名.GetType()」を用いると、その変数の.NET型情報を返してくれるので、これをConsole.WriteLineなどで表示することができる
・例: bool a; Console.Write(a.GetType()); //「System.Boolean」と表示
・C#が提供する「変数名.ToString()」を用いると、変数の内容を示す文字列を返してくれるので、これをConsole.WriteLineなどで表示することができる
・なお、bool型で値がtrueであれば「True」、falseであれば「False」となる。
・実は、ToString()は自動的に動作するもので「Console.Write(a); 」としても「True」が表示される

p.57 bool01.cs

```
//p.57 bool01.cs
using System;
class bool01
{
    public static void Main()
    {
        bool a = true;  //論理型の変数aを真で初期化
        bool b = false; //論理型の変数bを偽で初期化
        Console.WriteLine("a = {0}, b = {1}", a, b); //「a = True, b = False」と表示
        Console.WriteLine("aは{0}", a.GetType());//「aはSystem.Boolean」と表示
        Console.WriteLine("aは文字列にすると「{0}」", a.ToString()); //「True」と表示
        Console.WriteLine("bは文字列にすると「{0}」", b.ToString()); //「False」と表示
    }
}
```

アレンジ演習:p.57 bool01a.cs

・2行追加して「論理型変数 + 文字列」が可能かどうか、可能であれば型と結果を表示しよう

```
//アレンジ演習:p.57 bool01.cs
using System;
```

```
class bool01
{
    public static void Main()
    {
        bool a = true;  //論理型の変数aを真で初期化
        bool b = false; //論理型の変数bを偽で初期化
        Console.WriteLine("a = {0}, b = {1}", a, b); //「a = True, b = False」と表
示
        Console.WriteLine("aは{0}", a.GetType());//「aはSystem.Boolean」と表示
        Console.WriteLine("aは文字列にすると「{0}」", a.ToString()); //「True」と表示
        Console.WriteLine("bは文字列にすると「{0}」", b.ToString()); //「False」と表示
        Console.WriteLine(a + "型です"); //【追加】「True型です」と表示
        Console.WriteLine((a + "型です").GetType()); //【追加】「System.String」と表示
    }
}
```

p.58 リテラル

・プログラム中に記述された値のこと、整数リテラル、実数リテラル、文字リテラル、文字列リテラル、論理リテラルがある。
・リテラルの型は表記法とサフィックス(接尾語)で決まる。
・整数リテラルはサフィックスがなければint型→uint型→long型→ulong型(値の大きさによる)。
 L/lを付けるとlong型、U/uを付けるとuint型、UL/ulを付けるとulong型。
・実数リテラルはサフィックスがなければdouble型。F/fを付けるとfloat型、M/mを付けるとdecimal型。
・文字リテラルはchar型
・文字列リテラルはstring型
・論理リテラルはbool型

p.59 Object.GetTypeメソッド

・前に「Object.」とつくものは、自動的に利用可能になっているメソッドなどを示す
・GetTypeメソッドの対象はリテラルや式でも良い
・なお、単項-演算子は優先度が低いのでカッコで囲み「(-10).GetType()」のように指定しすれば良い

p.60 literal01.cs

```
//p.60 literal01.cs
using System;
class literal01
{
    public static void Main()
    {
        string format = "{0}の型は.NET型で{1}です"; //共通で用いるフォーマット指定
        Console.WriteLine(format, "100",     100.GetType()); //System.Int32(int)
        Console.WriteLine(format, "100U",   100U.GetType());
//System.UInt32(uint)
        Console.WriteLine(format, "100L",   100L.GetType());
//System.Int64(long)
        Console.WriteLine(format, "100UL", 100UL.GetType());
//System.UInt64(ulong)
        Console.WriteLine(format, "1.25",   1.25.GetType());
//System.Double(double)
        Console.WriteLine(format, "1.25F", 1.25F.GetType());
//System.Single(float)
```

```
        Console.WriteLine(format, "1.25M", 1.25M.GetType());
//System.Decimal(decimal)
        Console.WriteLine(format, "10F",     10F.GetType());
//System.Single(float)
        Console.WriteLine(format, "10D",     10D.GetType());
//System.Double(double)
        Console.WriteLine(format, "10M",     10M.GetType());
//System.Decimal(decimal)
        Console.WriteLine();
        Console.WriteLine(format, "-10D", (-10D).GetType());
//System.Double(double)
    }
}
```

p.61 暗黙の型指定

・初期化によって型を確定できる場合、型名の代わりに「var」を指定すると、型が自動的に推定される
・例: var p = 3; //pはint型になる
・例: var f = true; //fはbool型になる
・なお、無を表すnullを初期値に出来るが、nullでは型を確定できないので対象外。
・どの型になったのかはGetType()メソッドで確認できる
※ varを積極的に利用するかどうかは実務ではチームルールで決めていることがある

【補足】p.62 var01.cs、p.63 dynamic01.csについて

・「public static int Main()」となっているが、この内容ではいつもの「public static void Main()」として良い
・合わせて「return 0;」は不要

```
//p.62 var01.cs
using System;
class var01
{
    public static void Main() //※テキストではintだがvoidで良い
    {
        var mytext = "猫でもわかるC#プログラミング 第"; //string型になる
        var no = 3; //int型になる
        var myc = '版'; //char型になる
        Console.WriteLine(mytext + no + myc); //連結してstring型になる
        Console.WriteLine("mytextの型は{0}で、noの型は{1}で、mycの型は{2}です",
            mytext.GetType(), no.GetType(), myc.GetType()); //String,Int32,Char
    }
}
```

提出:アレンジ演習:p.56 escape01b.cs

・4つある変数をすべてvar型にして動作を確認しよう

次回予告:p.62「Dynamic型」から再開します。