講義メモ

・p.40「値型と参照型」から

提出フォロー：ミニ演習 **mini039.cs**

・int型の定数として TAX を10で初期化しよう（税率10%）
・これを用いて、500円の商品の税込み価格を表示しよう
　例「500円 税込み 550円」

作成例

```
//ミニ演習 mini039.cs
using System; //System名前空間の利用を宣言
class Mini037 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        const int TAX = 10; //税率を定数で定義して用いる
        int price = 500; //価格
        Console.WriteLine("{0}円 税込み {1}円", price, price * (100 + TAX) / 100);
    } //メソッド定義の終了
} //クラス定義の終了
```

p.40 値型と参照型

・int型は大きさが32ビットで整数用なので、32ビット分のメモリを確保して、そこに格納すれば良い
・このような「型の大きさが確定している」型を値型という
・対して、string型は0文字以上の文字列を扱うので、データの大きさがわからない
・そこで、別の場所にデータを格納し、その開始位置の情報（リファレンス、参照）を変数の値として持つ手法で解決する。
・このような「参照を持つ」型を参照型といい、string型やユーザ定義型などが該当する
・なお「string str = “ABC”;」という1行は下記のように実行される
　① メモリのどこかに文字列”ABC”が格納される（ちなみにこれはプログラム開始時に行われ終了まで有効）
　② 変数用の領域に、参照（番地情報）の大きさの変数strが用意される
　③ ①の参照（番地情報）が②に代入される

p.41 データ型

・値型には、数値型、文字型、論理型がある
・数値型には整数型と実数型（小数点を持つ値のための型）
※コンピュータにおいては整数と実数は扱いが異なり、整数は「その値」実数は「およその値（誤差がある前提）」

p.41 データ型：**.NET**型

・値型には2つの名前があり、int、doubleなどのC言語から引き継がれた古い型名と、C#の動作の基盤である .NET フレームワーク(p.8)で定めた型名（.NET型）がある
・一般的には古い型名が使われることが多いため、テキストも講義も基本的に古い型名を用いる
・しかし、C#システムからのヘルプ表示やエラー通知では.NET型が使われるので、確認しておこう
・例： 古い型名 int ⇒ .NET型 System.Int32
・.NET型における整数型は型の名前の中にビット数が入っているので、型の大きさがわかりやすい
　※これを理由として古い型名を使わない/推奨しないチームルールの場合もある

※なお、C言語ではint型などのビット数が不定（処理系依存）なので、C#で固定に改良されている
・.NET型の「System.」はプログラム冒頭で「using System;」を記述してあれば省略してOK
※チームルールで「using System;」を使わない/推奨しない場合もある

p.43 整数型

・int型は32ビットで負の数も格納可能なので、その範囲として、2^{32}＝4,294,967,296通りの情報から、半分を負の数用に、残りを0と正の数に用いる。よって、int型（.NET型Int32）で扱える値の範囲は -2,147,483,648 から 2,147,483,647 となる
・C#システムではint型がもっとも効率よく動作するようになっているが、0に近い値しか用いない場合、使わない領域が増えて無駄になる。そこで、int型より扱える値の範囲が狭い型が提供されている
・sbyte型（.NET型SByte）8ビットなので 2^8＝256通りの情報で、-128から127まで
・short型（.NET型Int16）16ビットなので 2^{16}＝65536通りの情報で、-32,768から32,767まで
・そして、int型では扱えない大きさの値を扱う型も提供されている
・long型（.NET型Int64）64ビットなので 2^{64}通りの情報で、 -9,223,372,036,854,775,808から 9,223,372,036,854,775,807まで
・また、プログラムにおいては負の数を持たないデータもあるので、intおよび上記を0以上専用にした符号なし（unsigned）型も提供されている。整数側の範囲が倍になり、負の数が紛れ込むことを防止できる。
・byte型（.NET型Byte）8ビットなので 2^8＝256通りの情報で、0から255まで
・ushort型（.NET型UInt16）16ビットなので 2^{16}＝65536通りの情報で、0から65,535まで
・uint型（.NET型UInt32）32ビットなので 2^{32}＝4,294,967,296通りの情報で、0から4,294,967,295まで
・ulong型（.NET型UInt64）64ビットなので 2^{64}通りの情報で、 0から18,446,744,073,709,551,615まで

p.43 整数型：**MaxValueとMinValue**

・C#が提供している型を現わす構造体（第11章）には型の最大値を返すプロパティ（属性情報）MaxValueと、最小値を返す MinValueがある
・型名がそのまま構造体の名前になっているので「型名.MaxValue」「型名.MinValue」でプログラムの中で型の最大値、最小値を扱える

p.44 type01.cs

```
//p.44 type01.cs すべての整数型の最大値と最小値を表示
using System;
class type01
{
   public static void Main()
   {
      Console.WriteLine("sbyte: {0}〜{1}", sbyte.MinValue, sbyte.MaxValue); //符号あり8ビット
      Console.WriteLine("short: {0}〜{1}", short.MinValue, short.MaxValue); //符号あり16ビット
      Console.WriteLine("int:   {0}〜{1}", int.MinValue, int.MaxValue); //符号あり32ビット
      Console.WriteLine("long:  {0}〜{1}",long.MinValue, long.MaxValue); //符号あり64ビット
      Console.WriteLine(); //空行を開ける
      Console.WriteLine("byte:  {0}〜{1}", byte.MinValue, byte.MaxValue); //符号なし8ビット
      Console.WriteLine("ushort:{0}〜{1}", ushort.MinValue, ushort.MaxValue); //符号なし16ビット
      Console.WriteLine("uint:  {0}〜{1}", uint.MinValue, uint.MaxValue); //符号なし32ビット
      Console.WriteLine("ulong: {0}〜{1}", ulong.MinValue, ulong.MaxValue); //符号なし64ビット
   }
}
```

アレンジ演習：**p.44 type01.cs**

・表示結果を3桁カンマ区切りにしよう
・型名を.NET型にしよう
・（できれば）符号ありの4行において「～」を中央に揃えよう

作成例

```
//アレンジ演習：p.44 type01.cs すべての整数型の最大値と最小値を表示
using System;
class type01
{
    public static void Main()
    {
        Console.WriteLine("SByte: {0,26:#,#}～{1,0:#,#}", SByte.MinValue, SByte.MaxValue);
        Console.WriteLine("Int16: {0,26:#,#}～{1,0:#,#}", Int16.MinValue, Int16.MaxValue);
        Console.WriteLine("Int32: {0,26:#,#}～{1,0:#,#}", Int32.MinValue, Int32.MaxValue);
        Console.WriteLine("Int64: {0,26:#,#}～{1,0:#,#}", Int64.MinValue, Int64.MaxValue);
        Console.WriteLine();
        Console.WriteLine("Byte  :{0}～{1,0:#,#}", Byte.MinValue, Byte.MaxValue);
        Console.WriteLine("UInt16:{0}～{1,0:#,#}", UInt16.MinValue, UInt16.MaxValue);
        Console.WriteLine("UInt32 {0}～{1,0:#,#}", UInt32.MinValue, UInt32.MaxValue);
        Console.WriteLine("UInt64:{0}～{1,0:#,#}", UInt64.MinValue, UInt64.MaxValue);
    }
}
```

p.45 整数型：**.Parse(**文字列**)**メソッド

・数字の列になっている文字列は、そのままでは計算などには使えない
・よって、数字型のデータに型変換する必要があり、これを実現するのが「型名**.Parse(**文字列**)**メソッド」
・たとえば、文字列”123”を整数123にするには、int n = int.Parse(“123”); とすれば良い
・文字列はstring型の変数に代入できるので、string str = “123”; int x = int.Parse(str); とできる
・この手法は、数値を文字列で得てしまう処理から数値を受け取る場合に必須
・具体的にはp.34の「Console.ReadLine()」で数値を入力したい場合に用いる（文字列として受け取る必要があるので、受け取ってから数値型に変換して用いる
・なお「型名.Parse(文字列)メソッド」に、その型の数値に変換できない文字列を渡すと、実行時エラーになり「ハンドルされていない例外: System.FormatException: 入力文字列の形式が正しくありません」と表示されてプログラムが異常終了する（対処法は第13章にて）

p.47 type02.cs

```
//p.47 type02.cs
using System;
class type02
{
    public static void Main()
    {
        Console.Write("整数を入力してください---");
        int x = int.Parse(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("今の数字を2倍すると{0}ですね。", x * 2);

        Console.Write("あなたの年齢を入力してください---");
        ushort age = ushort.Parse(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("あと{0}年で100歳ですね", 100 - age);
```

```
    }
}
```

アレンジ演習：**p.47 type02.cs**

・台形の上辺の長さ、下辺の長さ、高さをコンソールから入力したら面積を表示する処理を追加しよう
・なお、3つの値の型は UInt32とする

作成例

```
//アレンジ演習p.47 type02.cs
using System;
class type02
{
    public static void Main()
    {
        Console.Write("整数を入力してください---");
        int x = int.Parse(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("今の数字を2倍すると{0}ですね。", x * 2);

        Console.Write("あなたの年齢を入力してください---");
        ushort age = ushort.Parse(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("あと{0}年で100歳ですね", 100 - age);
        //【以下追加】
        Console.Write("台形の上辺の長さ：");
        UInt32 upper = UInt32.Parse(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.Write("台形の下辺の長さ：");
        UInt32 lower = UInt32.Parse(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.Write("台形の高さ：");
        UInt32 height = UInt32.Parse(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("この台形の面積は{0}", (upper + lower) * height / 2);
    }
}
```

p.47 type02.cs 補足：メソッドで得た値を他のメソッドに渡して初期化に用いる

・このプログラムでは「メソッドで得た値を他のメソッドに渡して初期化に用いる」ことを：
  int x = int.Parse(Console.ReadLine());
  この1行で行っている。これは定番処理なので、分解する必要はないが、理解の為に分解すると、下記のようになる（2つの変数を省略できている）
  ① string str; //入力用の文字列型変数の宣言
  ② str = Console.ReadLine(); //コンソールから数字列を入力
  ③ int x; //変換結果用の整数変数の宣言
  ④ x = int.Parse(str); //コンソールからえた数字列をint型に変換

p.45 整数型：**Convert**クラスの**To**型名**(**文字列**)**メソッド

・.NET型については、Parseの代わりに「To型名(文字列)メソッド」も利用できる
・このメソッドはConvertクラスにあるので「Convert.To型名(文字列)」の形式で呼び出すこと
・たとえば、文字列”123”をUInt32型123にするには、int n = Convert.ToUInt32.Parse(“123”); とすれば良い

p.47 type03.cs

```
//p.47 type03.cs
using System;
class type03
{
    public static void Main()
    {
        Console.Write("整数を入力してください---");
        int x = Convert.ToInt32(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("今の数字を2倍すると{0}ですね。", x * 2);

        Console.Write("あなたの年齢を入力してください---");
        ushort age = Convert.ToUInt16(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("あと{0}年で100歳ですね", 100 - age);
    }
}
```

アレンジ演習：**p.47 type03.cs**

・円の半径をコンソールから入力したら円周と面積を表示する処理を追加しよう
・なお、半径の値の型は UInt64とし、円周率は3.14とする

作成例

```
//p.47 type03.cs
using System;
class type03
{
    public static void Main()
    {
        Console.Write("整数を入力してください---");
        int x = Convert.ToInt32(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("今の数字を2倍すると{0}ですね。", x * 2);

        Console.Write("あなたの年齢を入力してください---");
        ushort age = Convert.ToUInt16(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("あと{0}年で100歳ですね", 100 - age);
        //【以下追加】
        Console.Write("円の半径：");
        UInt64 r = Convert.ToUInt64(Console.ReadLine()); //文字列を得て整数に変換した値で初期化
        Console.WriteLine("円周は{0}、体積は{1}", 2 * 3.14 * r, 3.14 * r * r);
    }
}
```

p.47 type03.cs 補足

・「int x = Convert.ToInt32(Console.ReadLine());」と型名の表現を同時に用いるのは避けた方が良い
「**UInt32** x = Convert.ToInt32(Console.ReadLine());」とすることが推奨されることがある

p.48 浮動小数点数型

・実数のコンピュータ内における表現方法は複数あり、最も柔軟で利用しやすい代わりに誤差がでやすいのが浮動小数点数型
・浮動小数点数型は下記の2つの型がある
　・float型（.NET型Single）32ビットで単精度なので誤差が大きくなりやすい
　・double型（.NET型Double）64ビットで倍精度なので誤差を小さくできる
・誤差を減らすと変数のサイズが増加してしまうので、誤差を許容できる場合は、変数のサイズが小さくなるfloat型を選ぶと良い
　※ Unityでは座標計算において小数点以下を用いないので、基本的に実数はfloat/Single型の利用が多い
・なお、浮動小数点数型はどちらも符号有りで、符号なし型は存在しない
・浮動小数点数型も「型名.MaxValue」「型名.MinValue」でプログラムの中で型の最大値、最小値を扱える
・なお、これらをそのままConsole.WriteLineなどで表示する指数表記になる

p.49 type04.cs

```
//p.49 type04.cs 浮動小数点数型の最大値と最小値を表示
using System;
class type04
{
    public static void Main()
    {
        Console.WriteLine("float: {0}～{1}", float.MinValue, float.MaxValue);
        Console.WriteLine("double:{0}～{1}", double.MinValue, double.MaxValue);
    }
}
```

アレンジ演習：**p.49 type04.cs**

・1.0f / 7.0f でfloat型の変数を初期化、1.0 / 7.0 でfloat型とdouble型の変数を初期化して表示することで、精度の違いを確認する処理を追加しよう
※ 1.0f / 7.0fの「f」はfloat型扱いの数値であることを示すサフィックス（p.59で解説）

作成例

```
//アレンジ演習：p.49 type04.cs 浮動小数点数型の最大値と最小値を表示
using System;
class type04
{
    public static void Main()
    {
        Console.WriteLine("float: {0}～{1}", float.MinValue, float.MaxValue);
        Console.WriteLine("double:{0}～{1}", double.MinValue, double.MaxValue);
        //【以下追加】
        float f = 1.0f / 7.0f; //単精度変数fを1/7で初期化
        Console.WriteLine("float : {0}", f);
        double d = 1.0 / 7.0; //倍精度変数dを1/7で初期化
        Console.WriteLine("double: {0}", d);
    }
}
```

p.49 Math.Powメソッド

・WrireLine/Write/ReadLineメソッドのあるConsoleクラスや、To○メソッドのあるConvertクラスのように、C#が提供する便利なクラスとしてMathクラスがある
・Mathクラスは算術演算系のメソッドなどを提供するクラスで、べき乗を返すPowメソッドがある
・p.49の枠の中はメソッドの形式を示す書式で
  ・「public static」で「Math.Pow」形式でどこからでも呼べる事を示す
  ・「double」で実行結果（戻り値）が倍精度実数型で返される事を示す
  ・「(double x, double y)」で倍精度実数型の2値を受け取ることを示す（xとyは特に意味はなく、省略される場合もある）
・よって、Math.Pow(実数①, 実数②) で呼び出し、結果をdouble型変数に代入すれば良いことがわかる
  例： double oct = Math.Pow(2.0, 3.0); // 2^3＝8が代入される

提出：アレンジ演習：**p.49 type04.cs**

次回予告：「p.50 type05.cs」の解説とアレンジ演習から