講義メモ
・p.62「Dynamic型」から

提出フォロー：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

作成例

```
//アレンジ演習：p.56 escape01b.cs
using System;
class escape01
{
    public static void Main()
    {
        var n = '\n'; //【変更】文字変数を改行文字で初期化(char型)
        var str1 = "今日は"; //【変更】文字列変数(string型)
        var str2 = "よい天気です"; //【変更】〃(string型)
        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
        var str3 = "今日は\nよい天気です"; //【変更】途中に改行文字のある文字列(string
型)
        Console.WriteLine(str3); //表示すると途中で改行する
        Console.WriteLine("\"Let's Go\"は\\1000です"); //「\"」「\\」が必要
    }
}
```

p.62 dynamic型

・var型は初期化が必要で、初期化に用いた値の型によって型が決まる
・よって、型が未定義状態にはならない。
・型を未定義状態にしておいて、代入によって決まるようにするのがdynamic型
・VS2022での利用には、ソリューションエクスプローラーで「参照」を右クリックし「参照の追加」で参照マネージャを起動。「Microsoft CSharp」のチェックをオンにして「OK」

p.63 dynamic01.cs

```
//p.63 dynamic01.cs
using System;
class Dynamic01 {
  public static void Main() {
    dynamic x = 10, y = "abc", z; //dynamic型の変数は初期化しなくても良い
    z = 1.25; //代入によって型を定めることもできる
    Console.WriteLine("x ---- {0}", x.GetType()); //System.Int32(int)
    Console.WriteLine("y ---- {0}", y.GetType()); //System.String(string)
    Console.WriteLine("z ---- {0}", z.GetType()); //System.Double(double)
  }
}
```

アレンジ演習：p.63 dynamic01.cs

・初期化も代入もしていないdynamic型の変数にGetType()を行うとどうなるか確認しよう
・また、初期化または代入の後のdynamic型の変数に、型の違う値を代入するとどうなるか確認しよう
・加えて、nullを代入し、表示とGetType()を行うとどうなるか確認しよう

```
//アレンジ演習:p.63 dynamic01.cs
using System;
class Dynamic01 {
  public static void Main() {
    dynamic x = 10, y = "abc", z; //dynamic型の変数は初期化しなくても良い
    //Console.WriteLine("z ---- {0}", z.GetType()); //【追加】初期化も代入もしてないのでエラーになる
    z = 1.25; //代入によって型を定めることもできる
    Console.WriteLine("x ---- {0}", x.GetType()); //System.Int32(int)
    Console.WriteLine("y ---- {0}", y.GetType()); //System.String(string)
    Console.WriteLine("z ---- {0}", z.GetType()); //System.Double(double)
    //【以下追加】
    x = "ABC"; //初期化済みのdynamic型の変数に型が異なる値を代入できる
    Console.WriteLine("x ---- {0}", x.GetType()); //System.String(string)に代わっている
    z = 'Z'; //代入によって型を変えることもできる
    Console.WriteLine("z ---- {0}", z.GetType()); //System.Char(char)に代わっている
    y = null; //nullを代入できるが…
    Console.WriteLine("y ---- {0}", y); //なにも表示されない
    Console.WriteLine("y ---- {0}", y.GetType()); //異常終了する
  }
}
```

p.64 スコープ

・スコープは「視野」のことで、C#では変数などの有効範囲のこと
・「{」から「}」までをブロックという
・なお、classの「{」がクラスのブロックの開始で、Main()の「{」がMainのブロックの開始であり、最後の2つの「}」でそれぞれのブロックの終わりを示す
・なお、必要に応じて、Main()の「{」から「}」のブロックの中に、さらにブロックを何重にも作成できる
・変数などの有効範囲は自分が定義されたブロックの中のみ
・C/C++などとはC#ではスコープが異なっても同じ名の変数は定義できない(移植時に注意)

p.65 型変換

・その型で扱える範囲が広いことを「型が大きい」といい、反対を「型が小さい」という
・「型が大きい」変数へ「型が小さい」変数や値を代入すると、大きい方に合わせた形で解釈され、無条件に代入可能。
  例: short s = 5; int i = s; //「型が小さい」short型から「型が大きい」int型への代入なのでOK
・これは値の大きさとは無関係で、例えば、long型の2はshort型に代入可能に思われるがエラーになる
・この場合は、型キャストを行って強制的に型変換させることも可能
・書式: (型)式や値
  例: long s = 5; short i = (short)s; //変数sの値をshort型に変換してから初期値とする
※ 型キャストが正しいかどうかはプログラマの責任で行う(コンパイラは警告しないことがある)。
※ 文字列型から整数型や実数型への型変換はキャストでは不可なので、parseメソッドを用いること(p.45)

p.66 cast01.csについて

・このプログラムは実行しても何も表示されない

アレンジ演習:p.66 cast01.cs

・aの値をbyte型の範囲を超える値にすると、どうなるか確認しよう
・また、キャストによって、元の変数の型は変わらないことを確認する処理を追記しよう

```
//アレンジ演習:p.66 cast01.cs
using System;
class cast01 {
    public static void Main() {
        long a = 500; //aの値をbyte型の範囲(0～255)を超えるものにしても
        byte b;
        b = (byte)a; //byte型へのキャストは可能だが
        Console.WriteLine("b = {0}", b); //244になってしまう(異常終了はしない)
        Console.WriteLine("aの型は{0}", a.GetType()); //System.Int64(long)で変わら
ない
    }
}
```

p.66 列挙型

・一連の整数値に名前をつけて扱う仕掛けが列挙で、グループの名前にあたるのが列挙名、グループのメンバーにあたるのが列挙子。
・定義書式: enum 列挙名 { 列挙子,… } //int型の列挙
・「列挙名:型」とすることで、sbyte、byte、short、ushort、uint、long、ulong型の列挙を定義することもできる列挙子
・列挙子には順に0から+1ずつ番号が振られるが「列挙子 = 整数」で任意の値に設定できる
・例:enum tuka { ichiendama = 1, goendama = 5, juendama = 10 };
・利用側は「列挙名.列挙子」で値を扱うことができる
・例:Console.WriteLine("５円玉は{0}", (int)tuka.goendama);
・C#やUnityが提供する特別な数値において、列挙型が用いられることが多い
 例:ConsoleColor 列挙型 { Black, DarkBlue, DarkGreen, DarkCyan, … }を色名して利用(p.106)
・列挙子のデータ型は列挙型になるので、整数としてあつかう時は型キャストすること
 例:Console.WriteLine("Blackの値は{0}", (int)ConsoleColor.Black);

p.68 enum01.cs

```
//p.68 enum01.cs
using System;
class enum01
{
    enum MyMonth { //月名の列挙の定義
        Jan = 1, Feb, Mar, Apr, May, Jun, Jul,
        Aug, Sep, Oct, Nov, Dec
    };
    public static void Main() {
        Console.WriteLine("Aprは{0}月", (int)MyMonth.Apr); //型キャストすると4になる
        Console.WriteLine("Mayは{0}月", (int)MyMonth.May); //型キャストすると5になる
    }
}
```

p.69 オブジェクト型とボックス化

・クラス型は7章で、配列型は6章で、デリゲート型は12章で説明します

p.73 文字列型

・文字列の扱いはC、C++、C#、Javaでそれぞれ異なるので注意。

・C#ではstring型で扱うが、値型ではなく参照型になっている（p.40）
・値型の変数は型に合わせた大きさで、中に値を持つ。
・文字列は0文字以上で文字数無制限なので、型に合わせた大きさが取れない。よって、別途配置しておいて、その開始位置（メモリアドレス）を変数値とすることで解決している
・よって、文字列変数「string s」への代入式「s = "ABC"」は下記のように実行される
 ① 文字列"ABC"がメモリの空いている場所に配置される
 ② 配置場所のメモリアドレスが、変数sに代入される
※ p.73にインスタンス(オブジェクト)の説明があるが、7章で扱う
・参照型においては、型によって便利な仕掛けであるプロパティやメソッドが提供されている
・string型の場合：
 ・Lengthプロパティ：文字数を返す。例：Consle.Write(s.Length) ⇒ 3
 ・IndexOf(文字)メソッド：その文字が何文字目にあるかを返す（先頭は0）。例：
Consle.Write(s.IndexOf('C') ⇒ 2
 ・IndexOf(文字列)メソッド：その文字列が何文字目からあるかを返す（先頭は0）。例：
Consle.Write(s.IndexOf("BC") ⇒ 1
・また、6章で説明する配列の構文を利用して、文字列の何文字目かの文字を得られる（先頭は0）。例：
Consle.Write(s[2]) ⇒ C
・また、8章で説明する静的メソッドを持っており、文字列変数② = String.Copy(文字列変数①); とすることで、文字列のコピーができる

p.73 string01.cs

```
//p.73 string01.cs
using System;
class string01
{
    public static void Main()
    {
        string str = "今日はよい天気です";
        string mystr;
        char c;
        // Lengthプロパティで文字列の長さを調べる
        Console.WriteLine("strは長さ{0}です", str.Length); //9
        //文字型変数cに文字列strの5番目の文字を代入
        c = str[4];
        Console.WriteLine("文字列の5番目の文字は「{0}」です", c); //「い」
        //文字列strをmystrにコピー
        mystr = String.Copy(str);
        Console.WriteLine("mystr = {0}", mystr); //「今日はよい天気です」
        //文字列の検索
        int n = str.IndexOf('は'); //下記では先頭が0なので1加えて1番目からにしている
        Console.WriteLine("文字列に'は'が出てくるのは{0}番目の文字", n + 1); //3
        n = str.IndexOf("よい"); //下記では先頭が0なので1加えて1番目からにしている
        Console.WriteLine("文字列に「よい」が出てくるのは{0}文字目から", n + 1); //4
    }
}
```

p.75 is演算子とas演算子

・まだ学習していない参照型のみを対象としているので、説明は割愛します

p.78 練習問題1 ex0301.cs ヒント

・Main()メソッドの中は、Console.WriteLine("1年は{0}秒", 計算式); の1行のみで良い

作成例

```
//p.78 練習問題1 ex0301.cs
using System;
class ex0301
{
    public static void Main()
    {
        Console.WriteLine("1年は{0}秒", 365 * 24 * 60 * 60);
    }
}
```

p.78 練習問題2 ex0302.cs ヒント

①Console.ReadLine()で円の半径を文字列(string型)で得る ⇒ p.34
②文字列を実数(double型)に変換して半径を得る ⇒ p.45
③面積を「半径×半径×Math.PI」で得て表示する

作成例

```
//p.78 練習問題2 ex0302.cs
using System;
class ex0302
{
    public static void Main()
    {
        string s; //入力用
        Console.Write("半径:"); s = Console.ReadLine(); //文字列を読み込む
        double r; //半径
        r = double.Parse(s); //実数に変換
        Console.WriteLine("半径{0}の円の面積は{1}", r, Math.PI * r * r); //表示
    }
}
```

作成例・ショートバージョン

```
//p.78 練習問題2 ex0302.cs
using System;
class ex0302
{
    public static void Main()
    {
        Console.Write("半径:");
        double r = double.Parse(Console.ReadLine()); //半径を文字列で読み込み実数に
変換
        Console.WriteLine("半径{0}の円の面積は{1}", r, Math.PI * r * r); //半径と面
積を表示
    }
}
```

第4章 演算子

p.79 式と演算子

・加算などの演算を示す記号や単語を演算子といい、1つまたは2つまたは3つの情報を得て1つの結果を返す
・この情報をオペランド（項）といい、1つだと単項演算子、2つだと2項演算子という。
・演算子には、計算結果を返す算術演算子、比較結果を返す関係演算子(p.88)、論理演算を行う論理演算子(p.89)などがある
・「=」も演算子で、代入演算子で、2項=演算子という。
・演算子を含む式は評価されて1つの結果を返す（算術演算子の場合は計算が評価）
・これを式の評価（式の値）という
・よって式「d = 5 + 2」は、2項+演算子による「5 + 2」の評価である「7」を、2項=演算子によってdに代入している。また、その評価は代入値である「7」となる。
・これでわかるように、2項+演算子の方が2項=演算子より優先されている（詳細はp.97）

p.80 expression01.cs

```
// p.80 expression01.cs
using System;
class expression01 {
    public static void Main() {
        int a = 0;
        Console.WriteLine("a = {0}", a);
        //代入式を指定するとその評価(値)を表示できる(代入では代入値)
        Console.WriteLine("(a = 7)の値は{0}", a = 7); //7となる
    }
}
```

p.81 算術演算子

・すでに学習した2項+演算子、2項-演算子、2項*演算子、2項/演算子で加減乗除算が可能
・加えて、2項%演算子(剰余算 p.84)、単項++演算子(インクリメント p.85)、単項--演算子(デクリメント p.85)がある
・2項+演算子は加算と結合の2つの演算が可能で、これを演算子のオーバーロード(多重定義)という
・2項+演算子は、2項の両方が加算可能であれば加算し、どちらかまたは両方が文字列ならば連結する

提出：p.78 練習問題2

次回予告：p.81「add01.cs」から再開します。