講義メモ

・p.69「オブジェクト型とボックス化」から

提出フォロー：アレンジ演習：**enum01.cs**

・列挙子 MyMonth.Aprと、MyMonth.Mayをint型にキャストせずに表示してみよう
⇒ “Apr”、”May”が表示される。これはConsole.WriteLine()が「文字列表現を用いる」としているためで、内部的に「文字列表現を返す」処理を呼ぶため（詳細は後述）

作成例

```
//アレンジ演習：p.68 enum01.cs
using System;
class enum01
{
    enum MyMonth
    {
        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);
        Console.WriteLine("Mayは{0}月", (int)MyMonth.May);
        Console.WriteLine("Aprは{0}月", MyMonth.Apr); //【追加】
        Console.WriteLine("Mayは{0}月", MyMonth.May); //【追加】
    }
}
```

p.69 オブジェクト型とボックス化

・7章で説明する「クラス」によってプログラマがオリジナルの型を定義して利用できる（後述）
・この時、その基本構造としてC#が提供するのがobject型（.NETではSystem.Object型）
・基本構造なので、他の全ての型の変数やリテラルを代入できる

アレンジ演習：**p.69 object01.cs**

・float型の変数fを適当な値で初期化し、object型の変数oに代入できることを確認しよう

作成例

```
//アレンジ演習：p.69 object01.cs
using System;
class object01
{
    public static void Main()
    {
        object o; //基本構造としてC#が提供する型
        byte b = 50;
        int i = 10;
        long l = 55555;
```

```
        double d = 0.00254;
        float f = 3.14f; //【追加】
        o = b; //byte型からの代入が可能
        Console.WriteLine(o); //代入された値が保持されている
        o = i; //int型からの代入が可能
        Console.WriteLine(o); //代入された値が保持されている
        o = l; //long型からの代入が可能
        Console.WriteLine(o); //代入された値が保持されている
        o = d; //double型からの代入が可能
        Console.WriteLine(o); //代入された値が保持されている
        o = f; //【追加】float型からの代入が可能
        Console.WriteLine(o); //【追加】代入された値が保持されている
    }
}
```

p.70 オブジェクト型とボックス化（ボックス化とボックス化解除）

・object型の変数に代入することをボックス化（ボックシング）といい、汎用的で便利だが、元の型情報が失われてしまう。
・object型の変数に代入後、元の型の変数に代入することをボックス化解除（アンボックシング）といい、元の型情報を型キャストで与える必要がある
・なお、型キャストせずに代入しようとすると文法エラーになり、元と違う型に型キャストすると実行時エラーになるので注意

アレンジ演習：**p.71 object02.cs**

・float型の変数fを適当な値で初期化し、object型の変数oに代入してから、oからfに再代入できることを確認しよう（ただし、アンボックシングが必要）

作成例

```
//アレンジ演習：p.71 object02.cs
using System;
class object02
{
    public static void Main()
    {
        object o;
        int i, a = 100;
        o = a; // ボクシング
        i = (int)o; //アンボクシング
        Console.WriteLine("i = {0}", o);
        //【以下追加】
        float f = 3.14f;
        o = f; // ボクシング
        f = (float)o; //アンボクシング
        Console.WriteLine("f = {0}, o = {1}", f, o);
    }
}
```

p.73 3.15 文字列型

※ C言語にはない型で、C++、Java、C#などでそれぞれ異なる考え方で実装しているので注意
・string型(.NETではSystem.String型)として提供されており、p.40で説明の通り、値型である数値型、文字型、論理型とは異なり、参照型で扱いが大きく異なる
・値型は大きさが固定だが、string型は0文字以上の文字列を扱えるので、見かけ上の大きさは不定
・ただし、p.40 図3.2のとおり、実際には別の場所に文字列を置き、その位置情報を持つ形式
・string型は内部構造を持っており、その中にメソッドとプロパティ(メソッドの一種、後述)があり、これを呼ぶことで文字列を上手く扱える
・Lengthプロパティ:「string型変数.Length」とすることで文字数が得られる(2バイト文字も1文字)
・Copyメソッド:「string型変数① = string.Copy(string型変数②)」とすることで、文字列をコピーできる
※ 文字列は長さがバラバラなので、単に代入するとコピーにならない(後述)
・IndexOfメソッド:「string型変数.IndexOf(文字)」とするとことで、この文字があれば何文字目かが得られる(ただし、先頭文字を0文字目とする)
・IndexOfメソッド:「string型変数.IndexOf(文字列)」とするとことで、この文字列が含まれていれば何文字目からかが得られる(ただし、先頭文字を0文字目とする)
・なお、後述する配列と同じ仕組みを持っており、string型変数[整数n] とすることで、先頭文字を0文字目とするn文字目の文字が得られる

アレンジ演習:**p.73 string01.cs**

・string型変数strに含まれない文字'も'が何文字目にあるかを得ようとすると、どんな値が得られるか確認しよう
⇒ -1が得られるので「この文字はない」と判断できる

作成例

```
//アレンジ演習: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);
        //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);
        //文字列strをmystrにコピー
        mystr = String.Copy(str);
        Console.WriteLine("mystr = {0}", mystr);
        //文字列の検索
        int n = str.IndexOf('は');
        Console.WriteLine("文字列に'は'が出てくるのは{0}番目の文字", n + 1);
        n = str.IndexOf("よい");
        Console.WriteLine("文字列に「よい」が出てくるのは{0}文字目から", n + 1);
        //【以下追加】
        n = str.IndexOf('も');
        Console.WriteLine("含まれない文字をIndexOf()で探すと{0}になる", n);
    }
}
```

p.75 3.16 is演算子とas演算子

※ 演算子なので4章の範疇であり、正誤情報もあり誤解を招く内容になっているため講座では割愛
https://www.sbcr.jp/support/12979/

p.78 練習問題1 ex0301.cpp ヒント

・1年の秒数なので、日数×1日の時間数×1時間の分数×1分の秒数で得られる（うるう年の考慮は不要）

作成例

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

p.78 練習問題2 ex0302.cpp ヒント

・p.50 type05.cs を参考にして、円の半径の入力用の変数rをコンソールからの入力で初期化しよう
・そして、変数rの2乗に円周率 Math.PI をかけて円の面積を得て表示しよう

作成例

```
//p.78 練習問題2 ex0302.cpp
using System;
class ex0302
{
    public static void Main()
    {
        Console.Write("半径：");
        double r = double.Parse(Console.ReadLine()); //半径を得る
        Console.WriteLine("面積は{0}", r * r * Math.PI); //面積を表示
    }
}
```

第4章　演算子

p.79 4.1 式と演算子

・+、-、*、/ のような計算を表す記号や単語を演算子という。なお「=」や「.」も演算子。
・演算子があつかう対象をオペランド（項）といい、演算子によってオペランドの数が異なる
・オペランドと演算子による表現を式という
・式を計算したり解釈することを「評価する」といい、その結果を「評価」という
・例えば式「1 + 2」を評価すると、評価として「3」が得られる
・演算子「=」による代入も演算で、評価することによって代入が行われる。なお評価は代入値になる
・例えば式「a = 5」を評価すると、評価として「5」が得られる

アレンジ演習：**p.80 expression01.cs**

・式「a = 7 + 2」の評価も表示してみよう
⇒ 先に「7 + 2」が評価されて「9」が得られ「a = 9」になり、代入値の9が評価になる

作成例

```
//アレンジ演習：p.80 expression01.cs
using System;
class expression01
{
    public static void Main()
    {
        int a = 0;
        Console.WriteLine("a = {0}", a);
        Console.WriteLine("(a = 7)の値は{0}", a = 7); //代入値が評価になる
        Console.WriteLine("(a = 7 + 2)の値は{0}", a = 7 + 2); //【追加】代入値が評価になる
    }
}
```

p.81 4.1 式と演算子（演算子の優先順位）

・上のアレンジ演習でわかるように、+演算子は、=演算子より優先される
　つまり「a = 7 + 2」が「a = 7」から評価されることはなく、先に「7 + 2」が評価される
・これは演算子の優先順位が定められているからで詳細はp.97で後述

p.81 4.2 算術演算子

・+、-、*、/ のような計算を表すえ演算子を算術演算子という
・算術演算子にはオペランドの数が1つのもの（単項演算子）、2つのもの（2項演算子）があり、同じ演算子でもオペランドの数が異なると内容が変わってしまうので注意
・また、オペランドの型によっても処理内容が変わることがあるので注意
・2項+演算子①：2つのオペランドが共に数値であれば、その和を評価とする
　例：1 + 2 ⇒ 3
・2項+演算子②：2つのオペランドのどちらかが文字列であればの連結結果の文字列を評価とする
　例：1 + ”x” ⇒ ”1x”
・なお、①②以外は文法エラーになるが、文字型は数値型として扱われる

アレンジ演習 **p.81 add01.cs**

・文字’a’と1を2項+演算子に渡すと得られる評価と、その型を表示してみよう

作成例

```
//アレンジ演習：p.81 add01.cs
using System;
class add01
{
    public static void Main()
    {
        Console.WriteLine(3 + 6); //9
        Console.WriteLine(3.0 + 6); //9(表示において小数点以下を省略)
```

```
        Console.WriteLine("3.5" + 6); //3.56(連結になる)
        Console.WriteLine(3.5 + "6"); //3.56(連結になる)
        Console.WriteLine(); //空白行の表示
        Console.WriteLine("(3.0 + 6)の型は{0}",
            (3.0 + 6).GetType()); //System.Double(実数9.0になる)
        Console.WriteLine("文字列3.5 + intの6の型は{0}",
            ("3.5" + 6).GetType()); //System.String(連結で文字列になる)
        Console.WriteLine("doubleの3.5 + 文字列6の型は{0}",
            (3.5 + "6").GetType()); //System.String(連結で文字列になる)
        //【以下追加】
        Console.WriteLine('a' + 1); //98('a'の文字コード97に1加算した結果)
        Console.WriteLine("('a' + 1)の型は{0}",
            ('a' + 1).GetType()); //System.Int32(加算でint型になる)
    }
}
```

p.82 4.2 算術演算子（型の異なる数値どうしの演算結果の型）

・型の異なる数値どうしの算術演算では、大きい方（表現できる範囲の広い方）に合わせてから計算し、大きい方の型の評価になるのが基本
　例：「3.0 + 6」はdouble型とint型なので、6をdouble型にして「3.0 + 6.0」としてから加算し9.0となる
　※ただし、System.WriteLine/Writeは実数値の「.0」を省略するので表示は「9」になる
・p.82 下の7つの法則は丸暗記不要で、必要に見直す程度でOK
・decimal型は例外で、他の型とは扱いが異なる

p.83 4.2 算術演算子（-演算子と/演算子）

・2項-演算子：2つのオペランドが共に数値であれば、左オペランドの値から右オペランドの値を差し引いた結果を評価とする
　例：1 - 3 ⇒ -2
・なお、どちらかが数値でなければ文法エラーになるが、文字型は数値型として扱われる
・単項-演算子：右オペランドの値の符号を反転した結果を評価とする
　例：aに10が代入されていたら、-a ⇒ -10
・2項/演算子①：2つのオペランドが共に整数であれば、左オペランドの値を右オペランドの値で割った結果の整数部を評価とする（つまり、小数点以下切り捨て）
　例：10 / 4 ⇒ 2（2.5の整数部になる）
　※なお、この時、右オペランドがゼロだと異常終了するので注意
・2項/演算子②：2つのオペランドが共に数値でどちらかまたは両方が実数であれば、左オペランドの値を右オペランドの値で割った結果の実数を評価とする（なお、割り切れない場合は誤差が発生する）
　例：10.0 / 4 ⇒ 2.5（実数÷整数）
　※なお、この時、右オペランドがゼロだと無限大となる（異常終了しない）

アレンジ演習：**p.83 division01.cs**

・「10 / 0.0」の評価と、その型を表示してみよう
　⇒「∞」でSystem.Doubleになる（ちなみに -10 / 0.0 は -∞ になる）
・「10 / 0」の評価を表示しようとしてみよう（文法エラーになることを確認しコメントアウト
・int型変数nを0で初期化し「10 / n」の評価を表示しようとしてみよう（異常終了することを確認しよう）
　⇒ コンソールにException(例外)の発生が表示され、0ではないコードで終了＝異常終了となる

作成例
//アレンジ演習：p.83 division01.cs

```
using System;
class division01
{
    public static void Main()
    {
        Console.WriteLine("10 / 3 = {0}", 10 / 3); //整数商で3
        Console.WriteLine("(10 / 3)の型は{0}",
            (10 / 3).GetType()); //System.Int32（int型）
        Console.WriteLine("10 / 3.0 = {0}", 10 / 3.0); //実数商で3.3…
        Console.WriteLine("(10 / 3.0)の型は{0}",
            (10 / 3.0).GetType()); //System.Double（double型）
        //【以下追加】
        Console.WriteLine("10 / 0.0 = {0}", 10 / 0.0); //実数商で∞
        Console.WriteLine("(10 / 0.0)の型は{0}",
            (10 / 0.0).GetType()); //System.Double（double型）
        //Console.WriteLine("10 / 0 = {0}", 10 / 0); //文法エラー
        int n = 0;
        Console.WriteLine("10 / n = {0}", 10 / n); //実行時エラー
    }
}
```

p.84 剰余演算子

・割った結果の余りを返す演算子で2項%演算子となる
　※この「%」にはパーセント（百分率）の意味は全くないので注意
・C/C++の剰余演算子は整数専用だが、C#では実数にも利用可能
・2項%演算子：2つのオペランドが共に数値であれば、左オペランドの値を右オペランドの値で割った余りを評価とする
　※ただし、右オペランドの値が整数0だと文法エラーになり、値が0である変数や式だと異常終了する

アレンジ演習：**p.84 mod01.cs**

・「10 % 0.0」の評価と、その型を表示してみよう
　⇒ NaN（Not a Numberの略で非数のこと）でSystem.Doubleになる
・「10 % 0」の評価を表示しようとしてみよう（文法エラーになることを確認しコメントアウト
・int型変数nを0で初期化し「10 % n」の評価を表示しようとしてみよう（異常終了することを確認しよう）

作成例

```
//アレンジ演習：p.84 mod01.cs
using System;
class mod01
{
    public static void Main()
    {
        Console.WriteLine("10 % 3 = {0}", 10 % 3); //1
        Console.WriteLine("13.53 % 2 = {0}", 13.53 % 2); //1.53
        Console.WriteLine("13.53 % 2.5 = {0}", 13.53 % 2.5); //1.03
        //【以下追加】
        Console.WriteLine("10 % 0.0 = {0}", 10 % 0.0); //NaN
        Console.WriteLine("(10 % 0.0)の型は{0}",
            (10 % 0.0).GetType()); //System.Double（double型）
```

```
        //Console.WriteLine("10 % 0 = {0}", 10 % 0); //文法エラー
        int n = 0;
        Console.WriteLine("10 % n = {0}", 10 % n); //実行時エラー
    }
}
```

p.85 インクリメント演算子

・多くのプログラムにおいて「変数の値をカウントアップ」することは多い
・これを式にすると「変数 = 変数 + 1」になり、違和感があるので、専用の単項++演算子が提供されている
・単項++演算子：オペランドが数値型の変数であればその値を1加算（インクリメント）した結果にする
・よって、式「変数 = 変数 + 1」は「変数++」と記述できる
・なお、単項++演算子は「++」を変数の前に書く「前置」と、後に書く「後置」があり、どちらも変数をインクリメントするが、式の評価が異なるので注意
　※単独で用いる（評価を用いずインクリメントだけ行う）場合は「変数++」と「++変数」のどちらでもOK
・前置単項++演算子：オペランドが数値型の変数であればその値を1加算（インクリメント）した結果にし、その結果を評価とする（つまり、評価する前にインクリメント）
・後置単項++演算子：オペランドが数値型の変数であればその値を評価としてから、1加算（インクリメント）した結果にする（つまり、評価した後にインクリメント）

提出：アレンジ演習：**p.86 increment01.cs**

・さらに「b++」した場合の、評価と変数bの値も表示してみよう

次回予告：**p.85** インクリメント演算子（デクリメント演算子）から