

2023 年 5 月 11 日 (木) 実施

変数とデータ型

変数とは

プログラム中の命令を通じて、メインメモリ上にデータを格納する領域を確保し、必要に応じてその場所に格納されるデータを上書きすることが可能である。このような領域を**変数**という。C#言語では、変数の取り扱いに関して、次の様な特徴がある。

1. プログラム中で用いる**変数は必ず宣言**しておく。⇒ メインメモリ上にデータを格納する領域を確保せよ、という命令に相当する。
2. 変数の宣言時には、**データ型を指定**する。⇒ データ型毎にデータを格納する領域のサイズが固定されている。

例) `int x; // 整数 (integer) のデータ型の変数 x を宣言`
3. 変数にデータを格納するには、**代入**を行う。

例) `x=10; /* 変数 x に定数 10 を代入 (=の右辺を評価し、その値を左辺の変数に格納)
 なお、宣言時に初期化することも可能である。⇒ int x=4; */`
4. 代入式の左辺以外に変数名を用いると**参照**が行われ、変数に格納されたデータが取り出されて利用される。

例) `y = x*2; // 変数 x の中身を 2 倍した値を変数 y に代入する。`

データ型

C#言語には数値や文字のデータを扱う**値型**と、クラス等を扱う**参照型**とがある。主な値型には、次のものがある。

分類	型名	サイズ	内容
文字型	char	16 ビット	Unicode 文字 (U+0000 ~ U+FFFF)
整数型	byte	8 ビット	符号無し整数 (0 ~ 255)
	short	16 ビット	符号付き整数 (-32768 ~ 32767)
	int	32 ビット	符号付き整数 ($-2^{31} \sim 2^{31} - 1 = -2147483648 \sim 2147483647$)
	long	64 ビット	符号付き整数 ($-2^{63} \sim 2^{63} - 1$)
実数型	float	32 ビット	単精度浮動小数点数 有効桁数 7 桁
	double	64 ビット	倍精度浮動小数点数 有効桁数 15-16 桁
論理型	bool	1 ビット	論理値 (true または false)

値型の変数はデータを直接格納するのに対して、参照型の変数はデータ（オブジェクト）への参照を格納する。

C#言語では、1文字を格納するには、**char 型**の変数を用いる。その値は **Unicode** で表される文字コードの数値で、**16 ビットの符号無し整数**である。

```
例) char c; // 文字 (character) のデータ型の変数 c を宣言
char c1 = 'A'; // c1 の宣言時に初期値として文字 A を設定 (一重引用符で挟む)
char n1 = '神'; // n1 の宣言時に初期値として文字 神 を設定 (C#言語では全角文字も
1文字)
```

文字列の扱い

String クラス

C#言語では、文字列を扱う **String クラス**が用意されている。また、**String** の別名として、参照型の **string** も用意されている。文字列データは**文字列リテラル**（左右の二重引用符『"』で挟まれた中身）によって表される。

```
例) string s;
s = "CUC"; // string s = "CUC"; の様に、宣言時に初期化することも可能である。
```

書式指定文字列

書式指定文字列とは、内容が実行時に動的に決定される文字列で、**Format** メソッドを用いて、中括弧内に、実行時に他の値に置換される**プレースホルダー**を埋め込むことで、作成される。

```
例) s = string.Format("{0} × {1} の答えは {2}", i, j, (i * j));
```

この例では、プレースホルダーとして埋め込まれた {0}、{1}、{2} のそれぞれが、プログラムの実行時に **i**、**j**、**i * j** の値に置き換えられる。

本日の課題

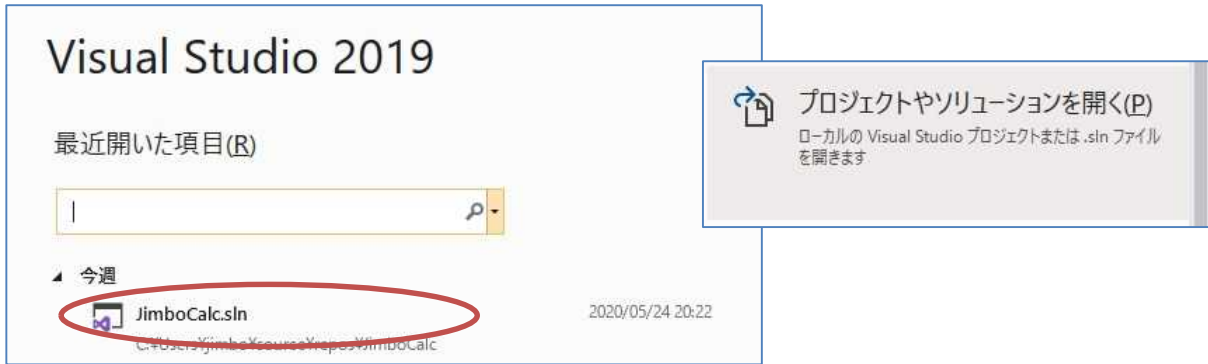
今回の授業では、文字列を格納する変数について、変数の宣言や代入、参照をどの様に記述していくのかを学ぶ。

手順

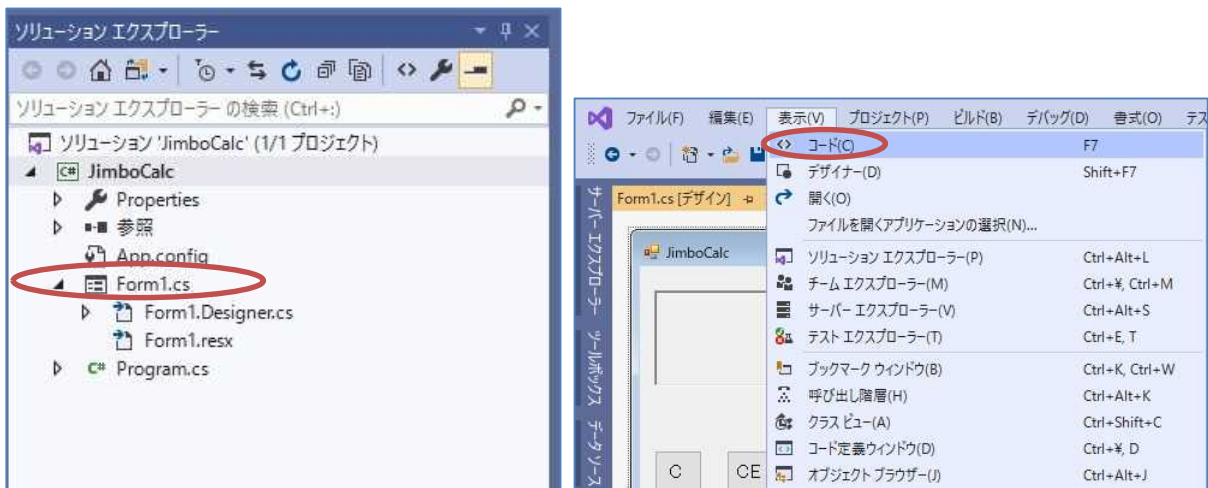
1) ソリューションを開く

『最近開いた項目』から **JimboCalc.sln** を選ぶか、それが表示されていないならば、『プロジェクトやソリューションを開く』で **JimboCalc.sln** を探して開く。

(図は次のページ)

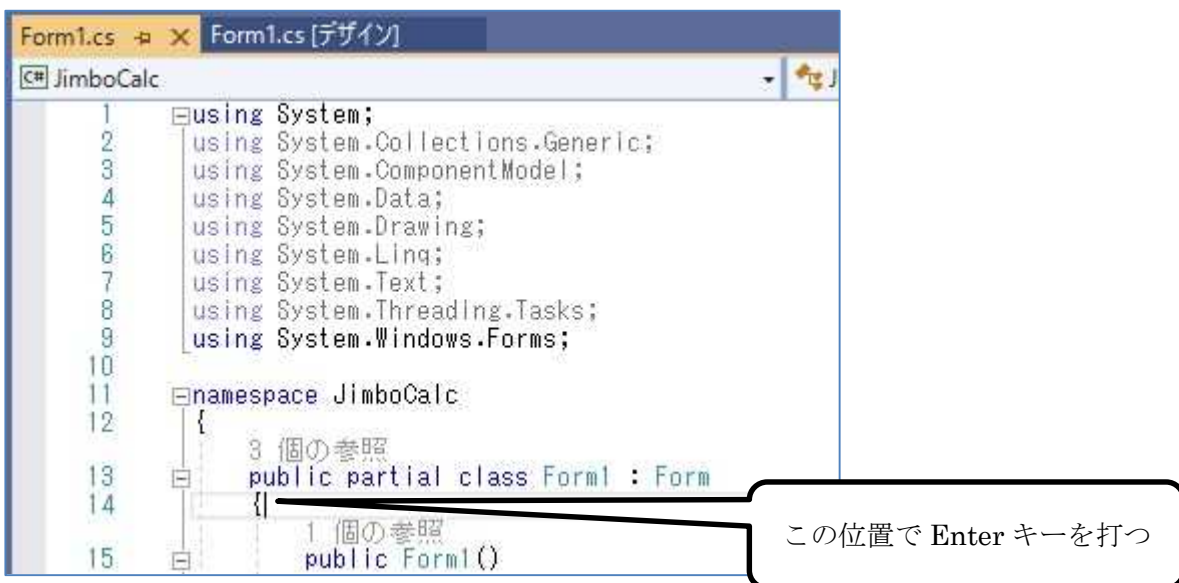


ソリューションを開いてもフォームが表示されない場合は『ソリューションエクスプローラー』で『Form1.cs』をダブルクリックしてフォームデザイナーを開く。また、『表示』タブの『コード』を選択して、コードエディタを開く。



2) コーディング

まず、文字列を格納する 2 つの変数 `degits`, `formula` を宣言し、空の文字列リテラルで初期化する。変数は `Form1.cs` 内の全体で有効なものとするため、クラスのフィールドで宣言する。



```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11 namespace JimboCalc
12 {
13     3 個の参照
14     public partial class Form1 : Form
15     {
16         string degits = "";
17         string formula = "";
18     }
19     1 個の参照
20     public Form1()
    
```

```

public partial class Form1 : Form
{
    string degits = "";
    string formula = "";
}
    
```

次に、フォームデザイナーに戻って、フォームをダブルクリックする。



『Form1_Load』というイベントハンドラが作成されるので、ラベルの Text プロパティを空の文字列リテラルに設定する。 (図は次のページ)

```

Form1.cs*  Form1.cs [デザイン]*
JimboCalc  JimboCalc.Form1
64  private void button8_Click(object sender, EventArgs e)
65  {
66      label1.Text = button8.Text;
67      label2.Text = label1.Text;
68  }
69
70  | 個の参照
71  private void button9_Click(object sender, EventArgs e)
72  {
73      label1.Text = button9.Text;
74      label2.Text = label1.Text;
75  }
76  | 個の参照
77  private void button10_Click(object sender, EventArgs e)
78  {
79      label1.Text = button10.Text;
80      label2.Text = label1.Text;
81  }
82  | 個の参照
83  private void Form1_Load(object sender, EventArgs e)
84  {
85  }
86  }
87
88
    
```

```

82  | 個の参照
83  private void Form1_Load(object sender, EventArgs e)
84  {
85      label1.Text = "";
86      label2.Text = "";
87  }
88
    
```

```

private void Form1_Load(object sender, EventArgs e)
{
    label1.Text = "";
    label2.Text = "";
}
    
```

更に、前回作成した button1 のイベントハンドラの中身を書き換える。

```

21
22  | 個の参照
23  private void button1_Click(object sender, EventArgs e)
24  {
25      degits += button1.Text;
26      formula += button1.Text;
27      label2.Text = degits;
28      label1.Text = formula;
29  }
    
```

```

private void button1_Click(object sender, EventArgs e)
{
    degits += button1.Text;
    formula += button1.Text;
    label2.Text = degits;
    label1.Text = formula;
}
    
```

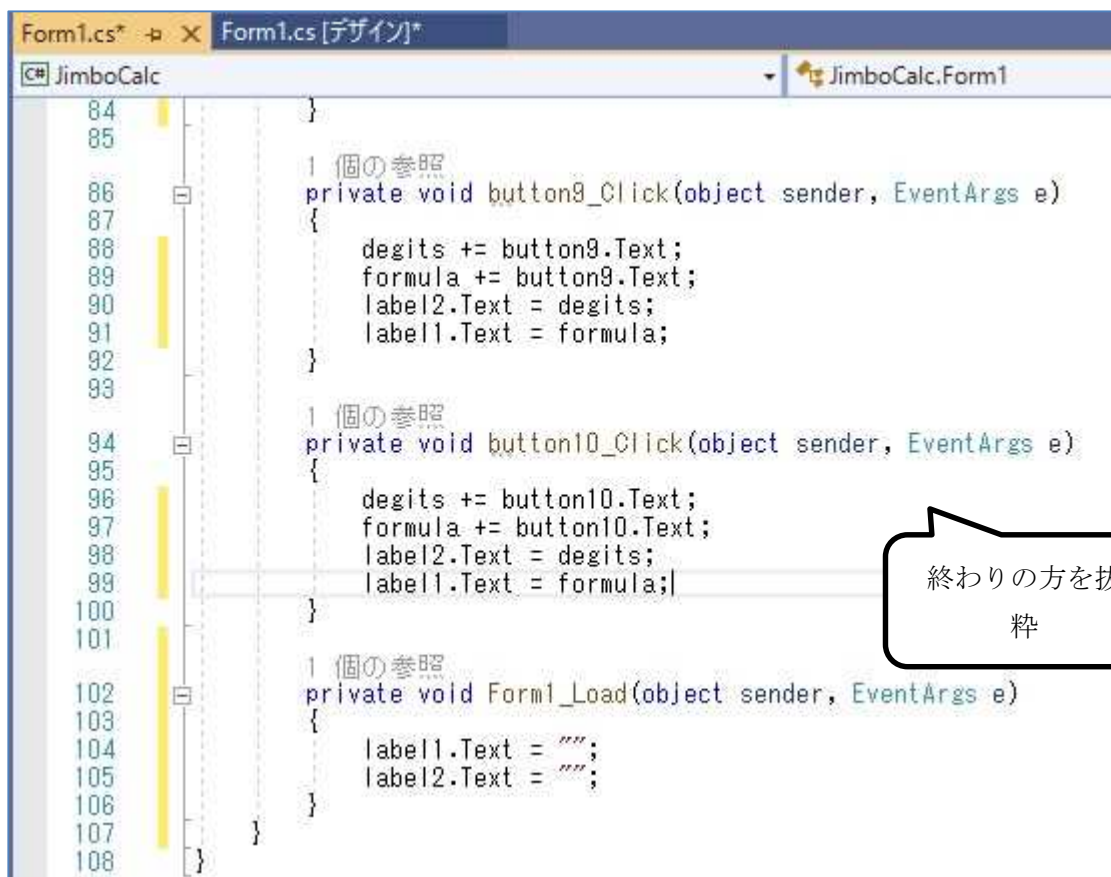
ここでは、ボタンの Text プロパティの値を `degits` という名前の変数の中身に連結して代入している。『+=』は複合演算子で、次の2つの文は等価である。

```
degits += button1.Text;
degits = degits + button1.Text;
```

2つ目の文の右側の `degits` では参照が行われ、変数に格納されたデータが取り出されて利用される。文字列リテラル同士に『+』演算子が適用されると文字列の連結が行われる。

また、ボタンの Text プロパティの値を `formula` という名前の変数の中身にも連結して代入している。続く2つの文では、`label1` 及び `label2` の Text プロパティのそれぞれに `degits` 及び `formula` を参照して得られたデータを代入している。

更に、フォームデザイナー上で『1』から『9』までのボタン (`button2` から `button10`) のイベントハンドラの中身を同様に書き換える。その際、`buton2_click` ならば、`button2.Text` を用いるといったことに注意する。



ここで、『Form1.cs の保存』ボタンを押してから、『開始』ボタンを押して、プログラムを実行して、『0』から『9』までのボタンを順に押して行って、動作を確かめる。

(最終的な図は次のページ)



これに相当する図(スクリーンショット)を自分のアプリの実行時に作成して課題提出用にする。(Alt キーを押しながら PrtScr キーを打って画面のイメージを取り込み, アクセサリのペイントでファイルとして保存する。)

提出物 :

- 1) フォームのデザインファイル **Form1.Designer.cs** をメールに添付して提出する。
- 2) コードエディタで編集したソースファイル **Form1.cs** をメールに添付して提出する。
- 3) 実行の最終結果のスクリーンショット **第 3 回実行結果.jpg** (.png も可) をメールに添付して提出する。
- 4) 質問を記述したファイル **Prog2_Questions_3rd.txt** に解答を書き込んで保存し, メールに添付して提出する。