

INTERNET ACADEMY

Institute of Web Design & Software Services

C#4

インターネット・アカデミー

C# 目次

1. フォームアプリの基本

2. アプリ作成

フォームアプリとは

GUIを用いたアプリ（フォームアプリ）には**WindowsForms**を利用

WindowFormsの特徴

- ・ **直感的な画面作成** … GUIベースで簡単にデザインが可能。
- ・ **イベントドリブンモデル** … ボタンなどの操作に対する処理を記述。
- ・ **学習コストが低い**… 文法も容易で情報が豊富で学びやすい。

プロジェクトの作成

①プロジェクトの種類の選択

VisualStudio2022を起動し[スタート ウィンドウ] で **[新しいプロジェクトの作成]** を選択

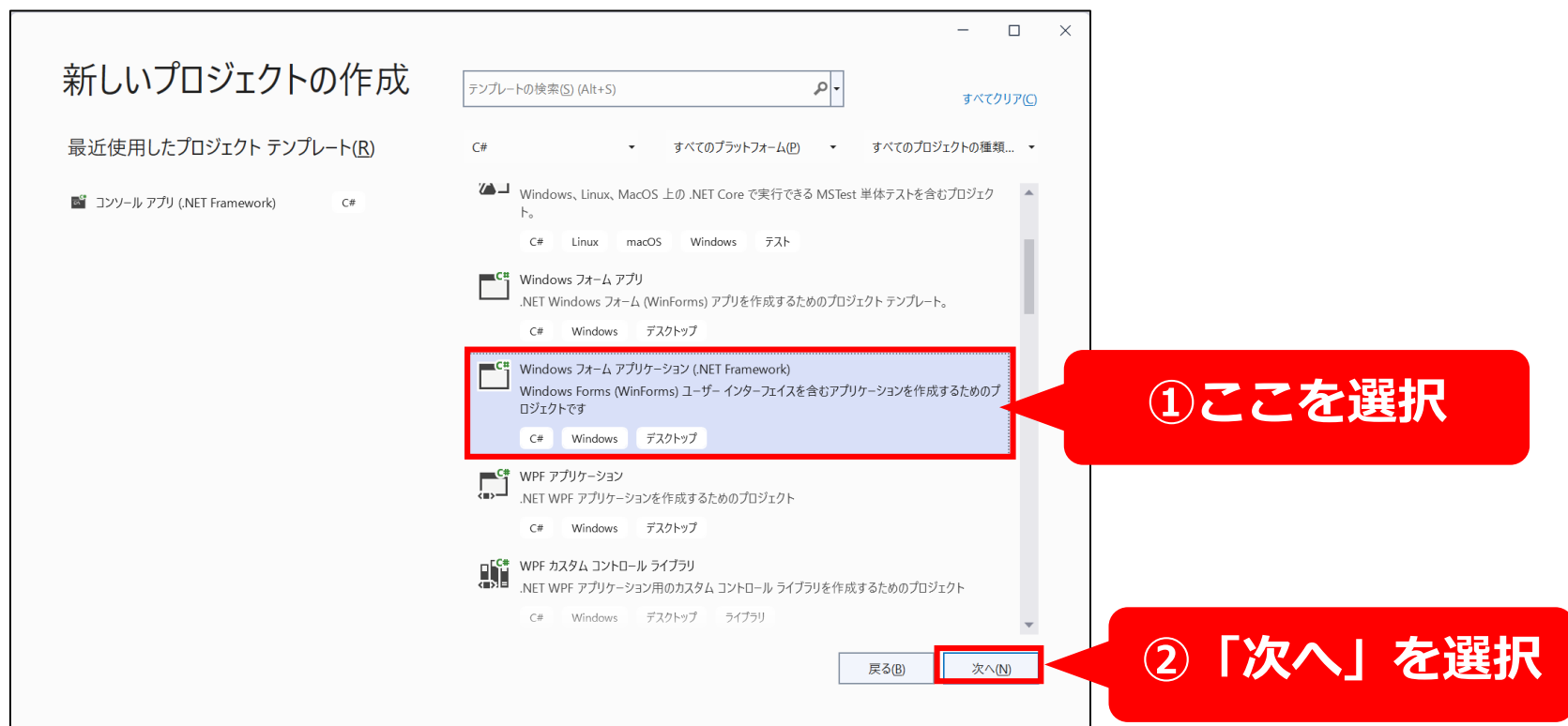


ここを選択

プロジェクトの作成

②アプリケーションの種類を選択

C# 用の [Windows フォーム アプリケーション (.NET Framework)] テンプレートを選択



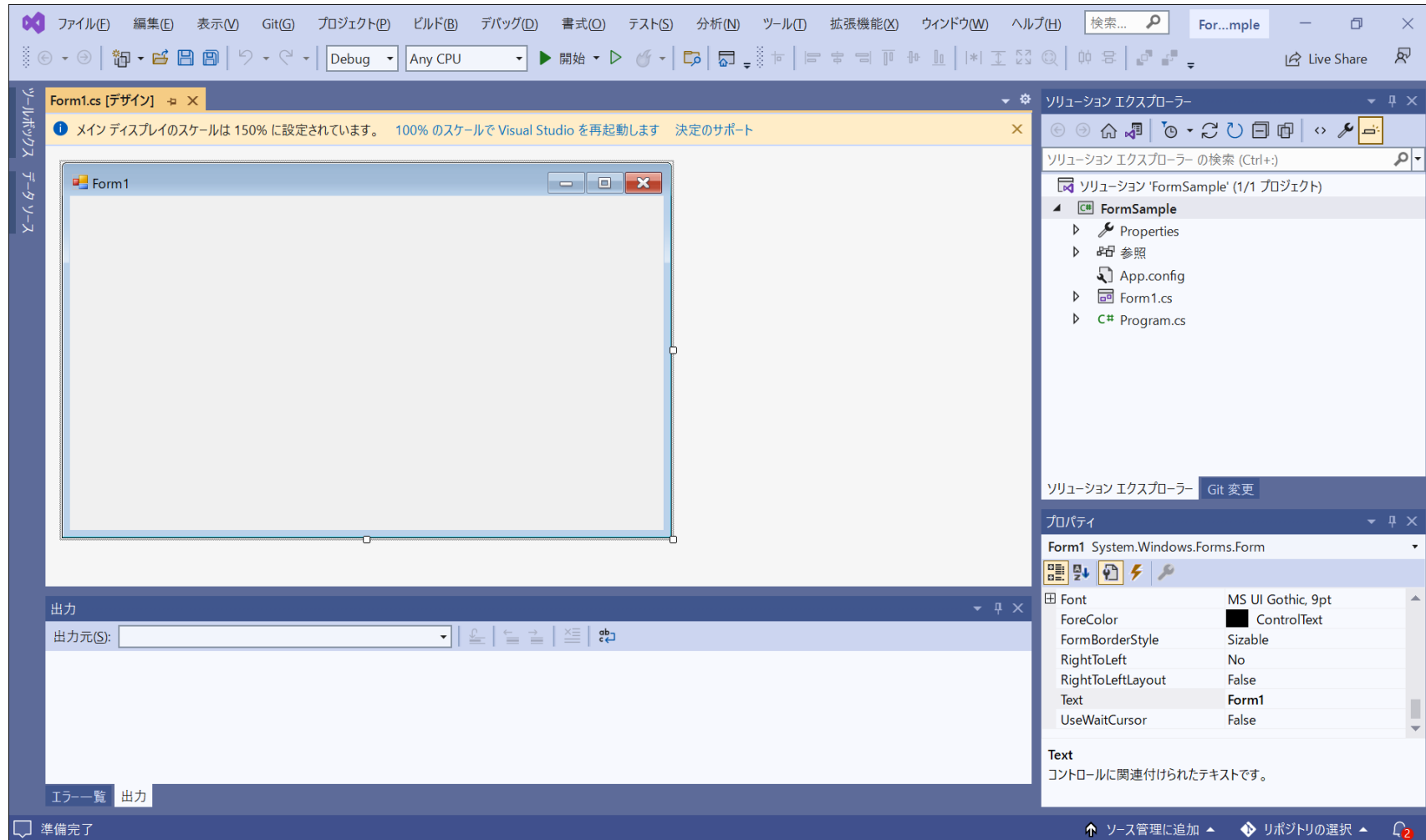
プロジェクトの作成

③プロジェクト名の入力

プロジェクト名を入力し、[作成](N) を選択

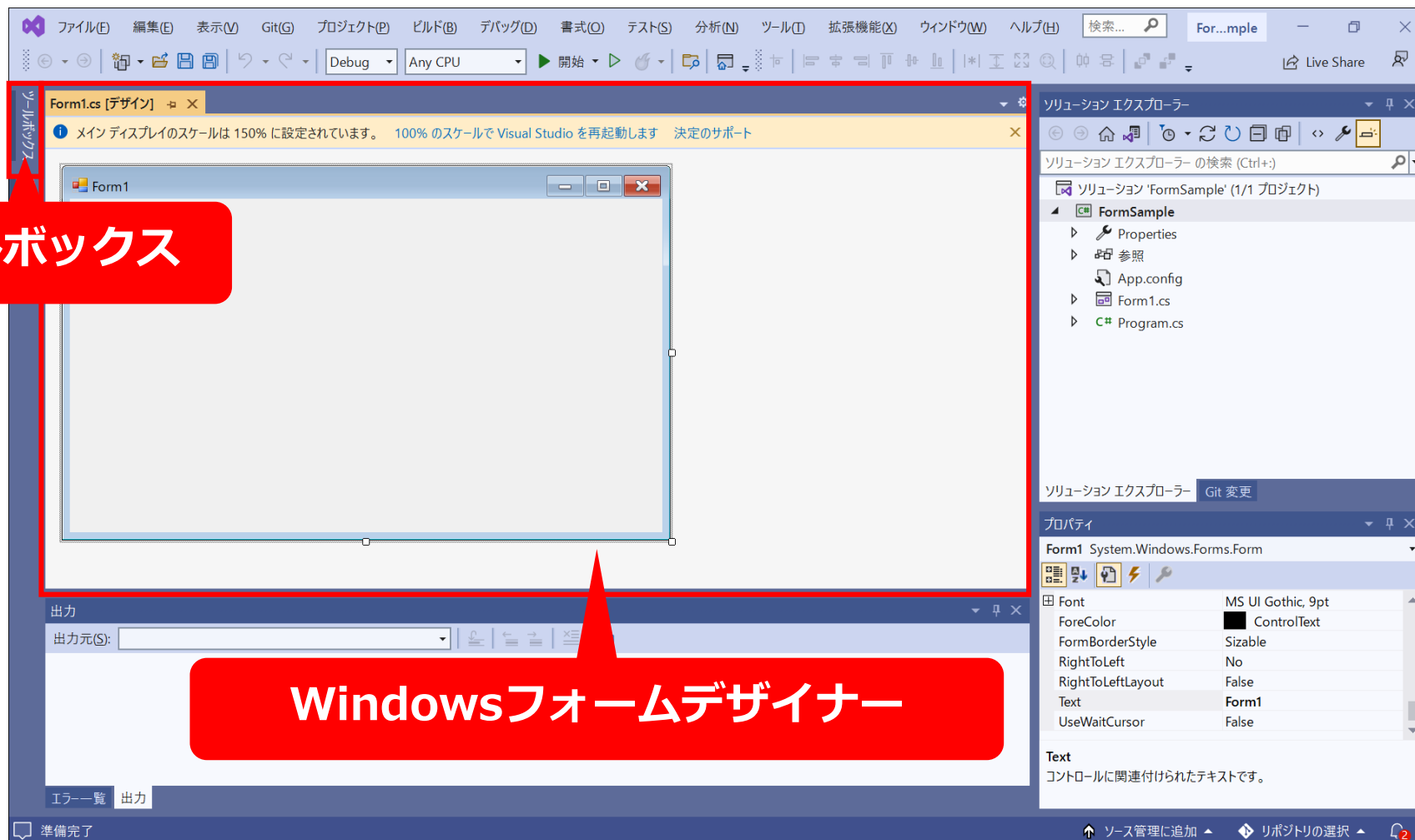
プロジェクトの作成

④プロジェクトの完成



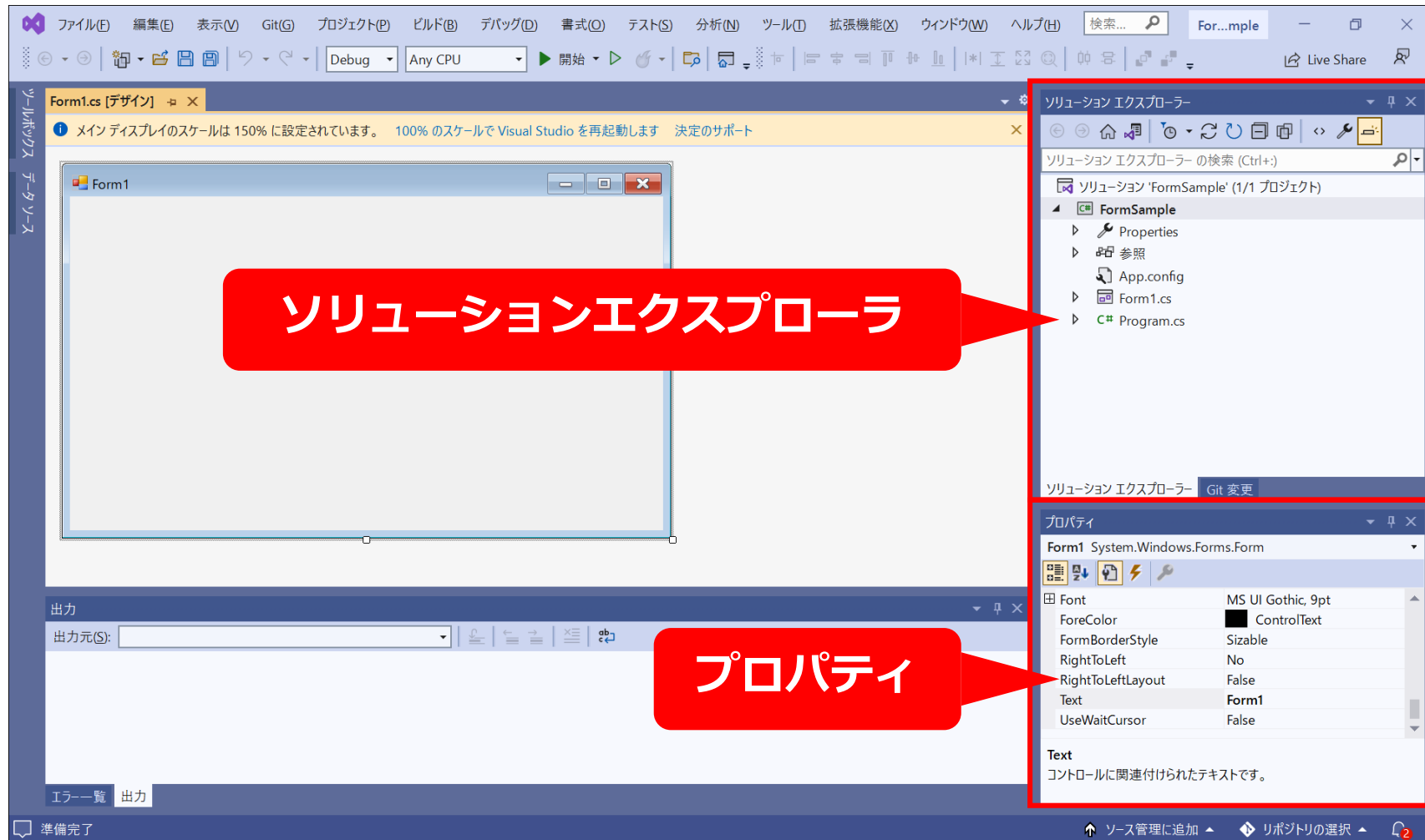
プロジェクトの作成

④プロジェクトの完成



プロジェクトの作成

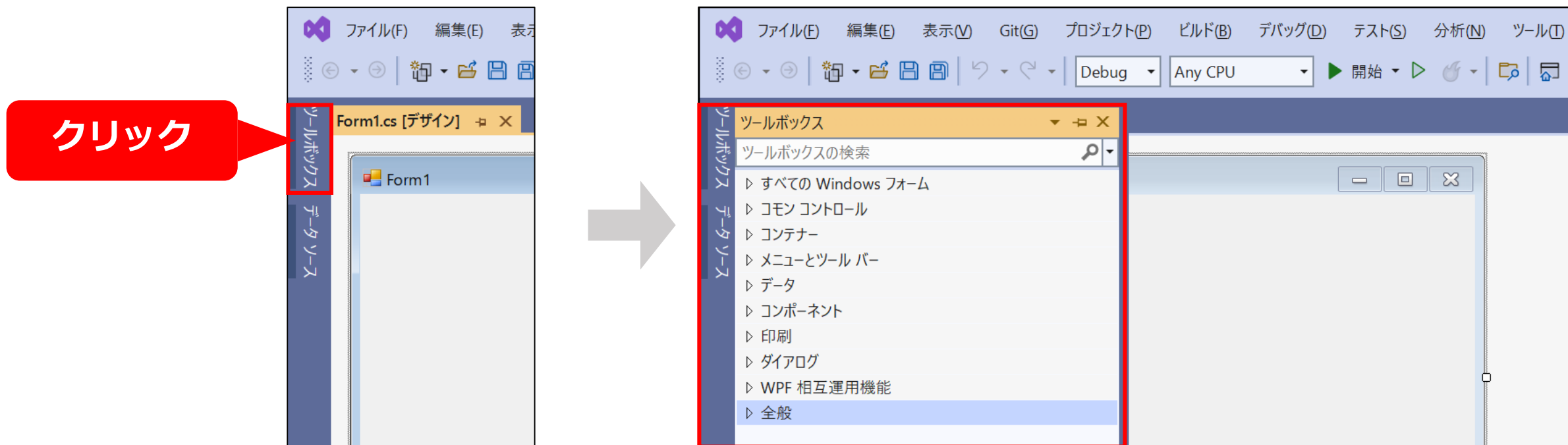
④プロジェクトの完成



Windowsフォームデザイナーとツールボックスの使い方

ツールボックス

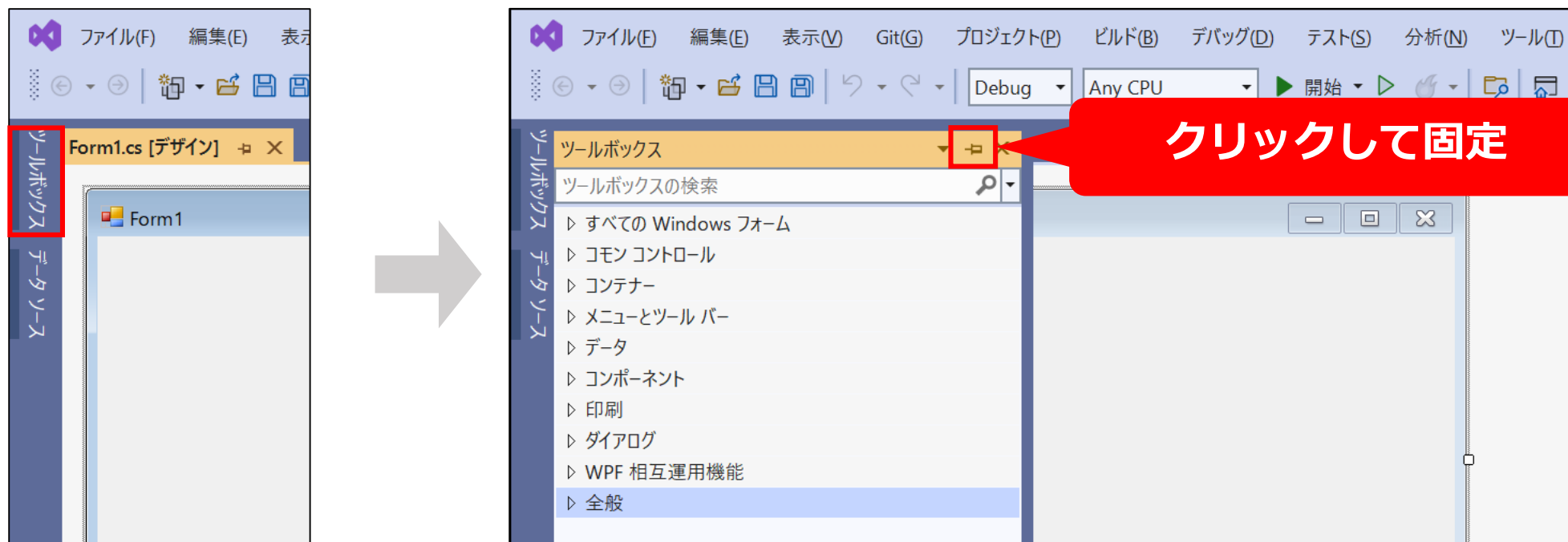
フォームに配置する様々な**フォームコントロール**がジャンル別に選択できる



Windowsフォームデザイナーとツールボックスの使い方

ツールボックス

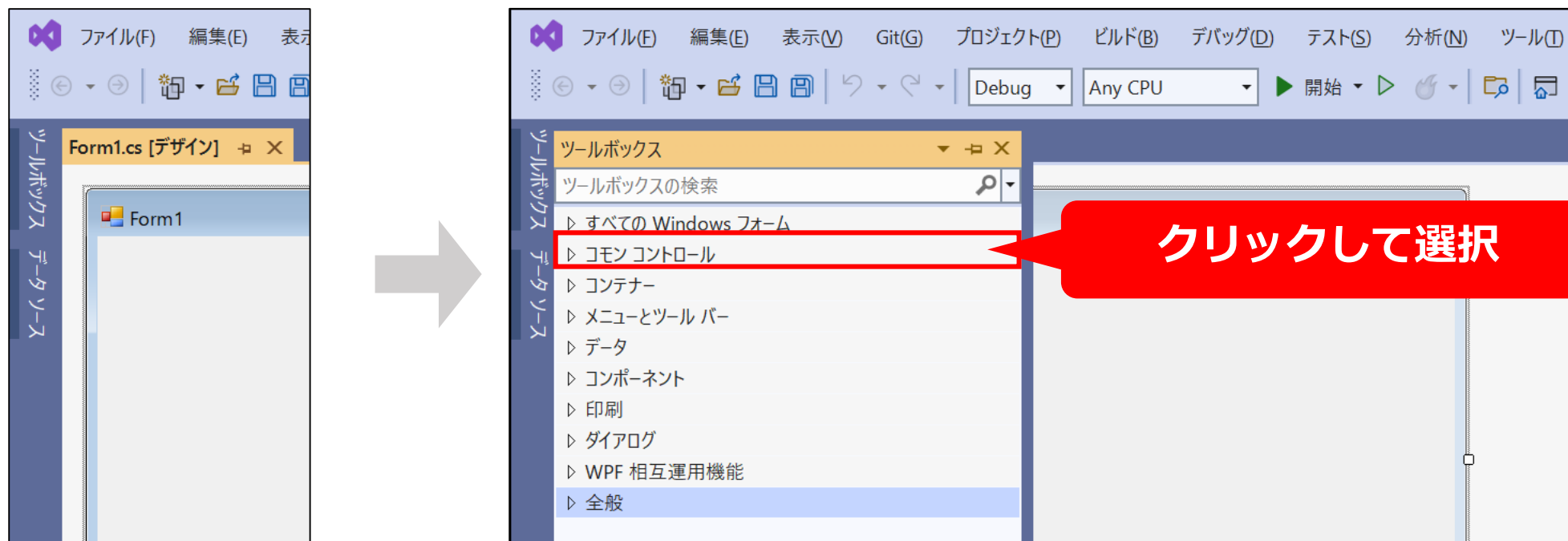
フォームに配置する様々な**フォームコントロール**がジャンル別に選択できる



Windowsフォームデザイナーとツールボックスの使い方

ツールボックス

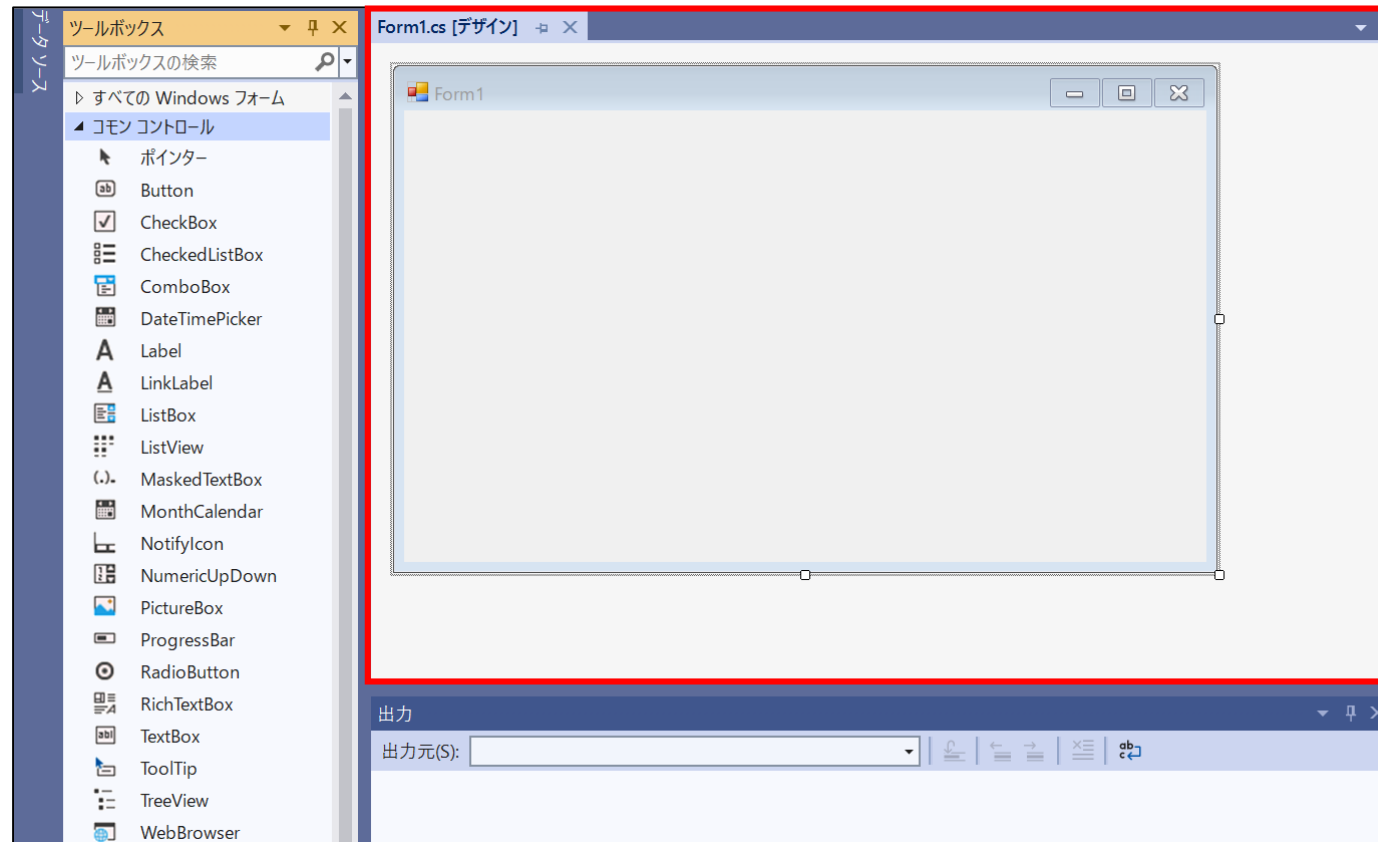
フォームに配置する様々な**フォームコントロール**がジャンル別に選択できる



Windowsフォームデザイナーとツールボックスの使い方

Windowsフォームデザイナー

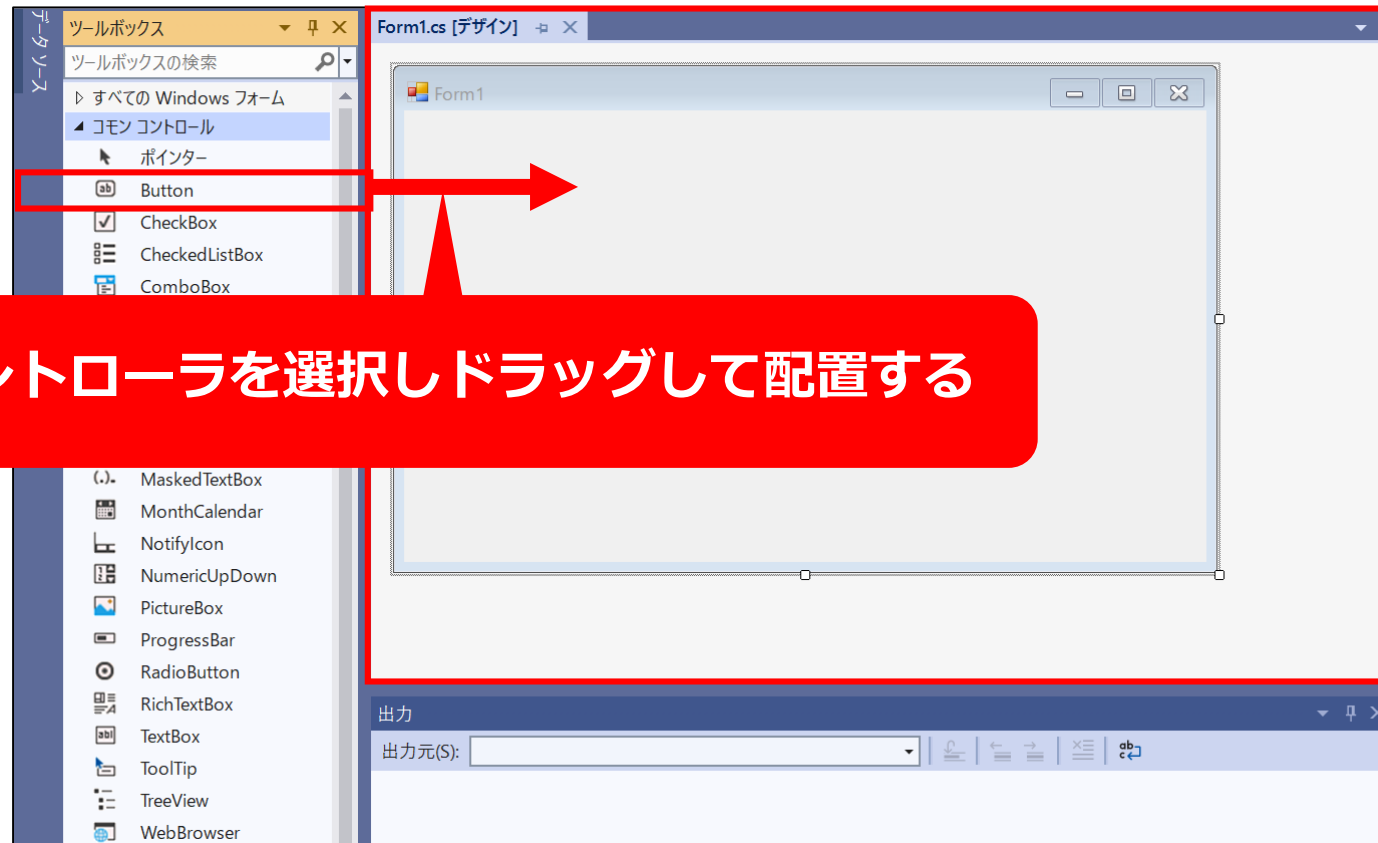
フォームへのデザインを行う部分。ボタンやテキストボックスなどを配置する



Windowsフォームデザイナーとツールボックスの使い方

Windowsフォームデザイナー

フォームへのデザインを行う部分。ボタンやテキストボックスなどを配置する



アプリ作成

テキストボックスとボタンを用いた簡単なアプリ

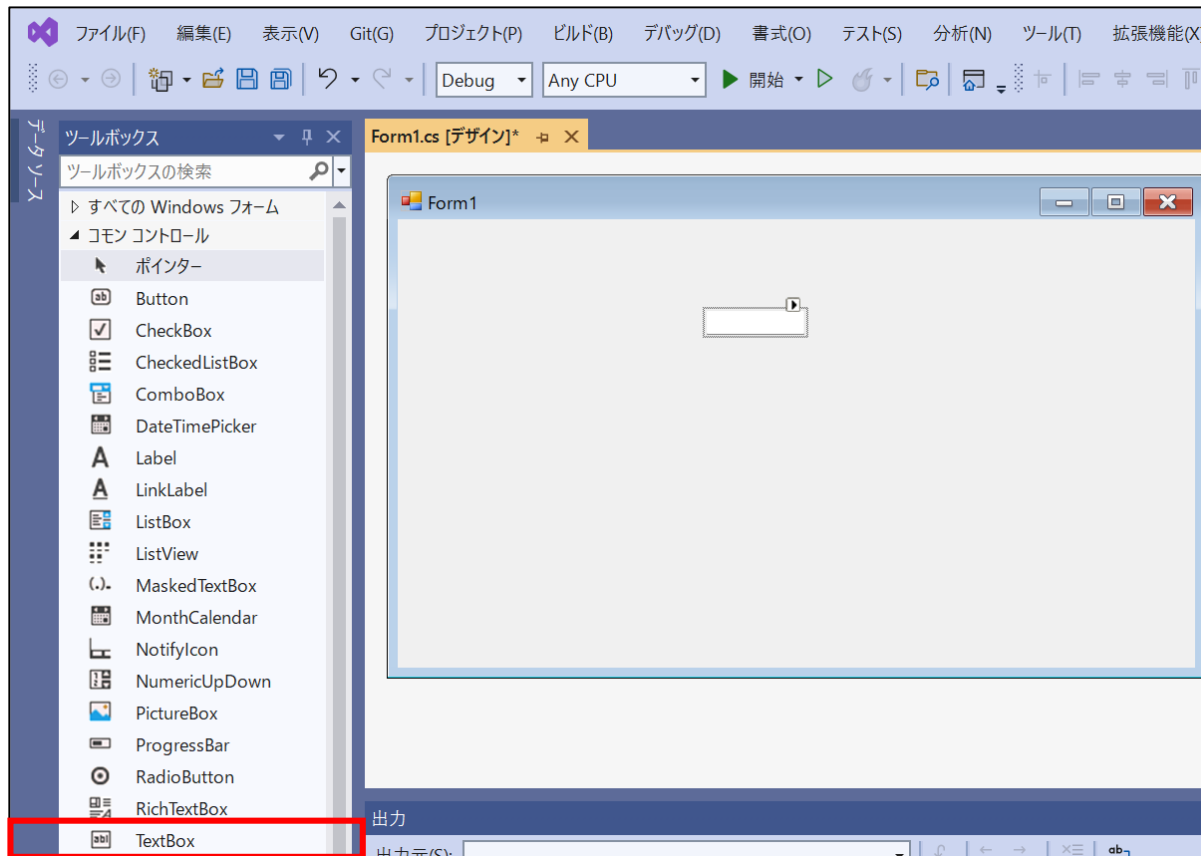
アプリケーションのイメージ



Windowsフォームデザイナーとツールボックスの使い方

テキストボックスの作成

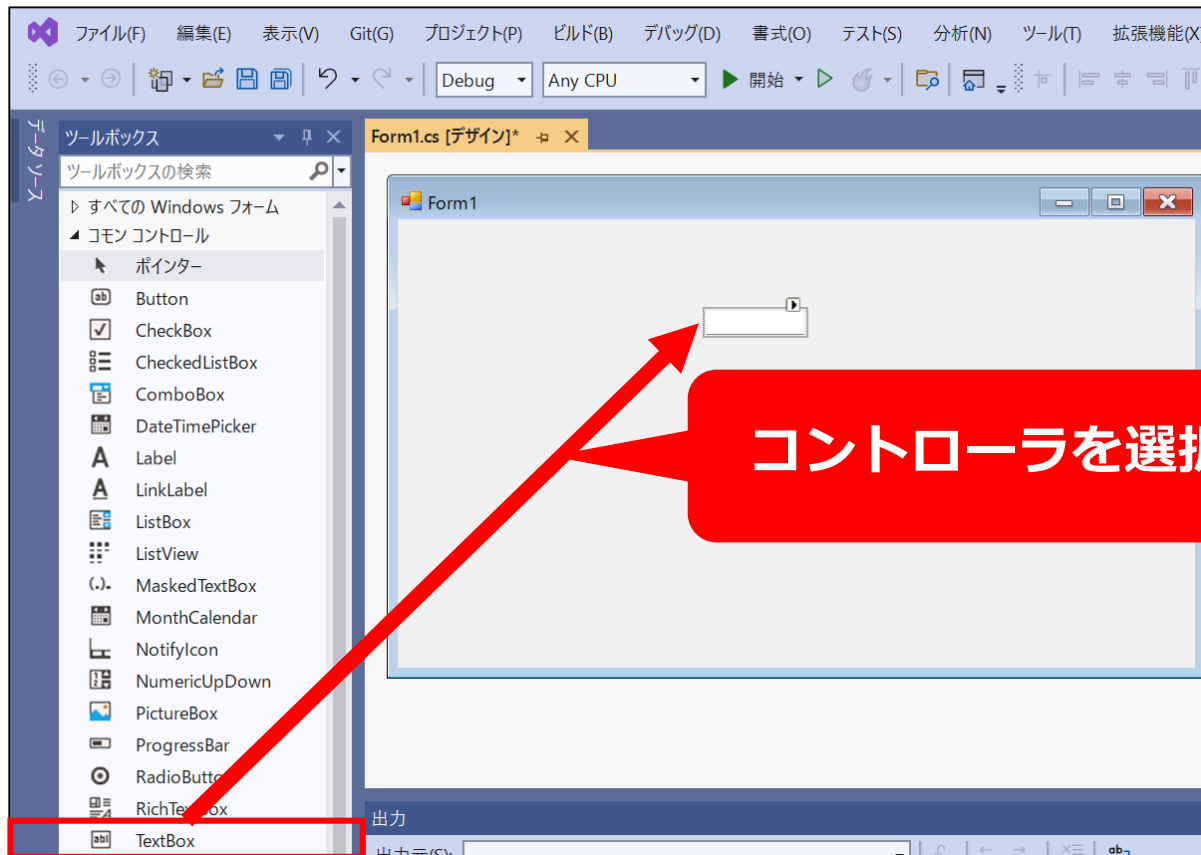
テキストボックスをドラッグしフォームに配置



テキストボックスの作成

フォームへ配置

テキストボックスをドラッグしフォームに配置

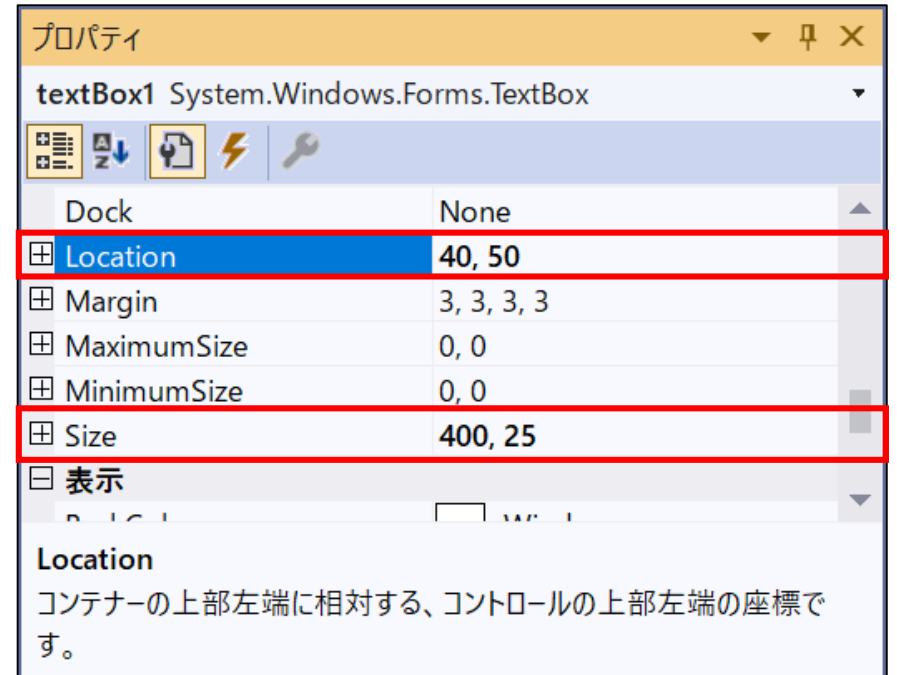
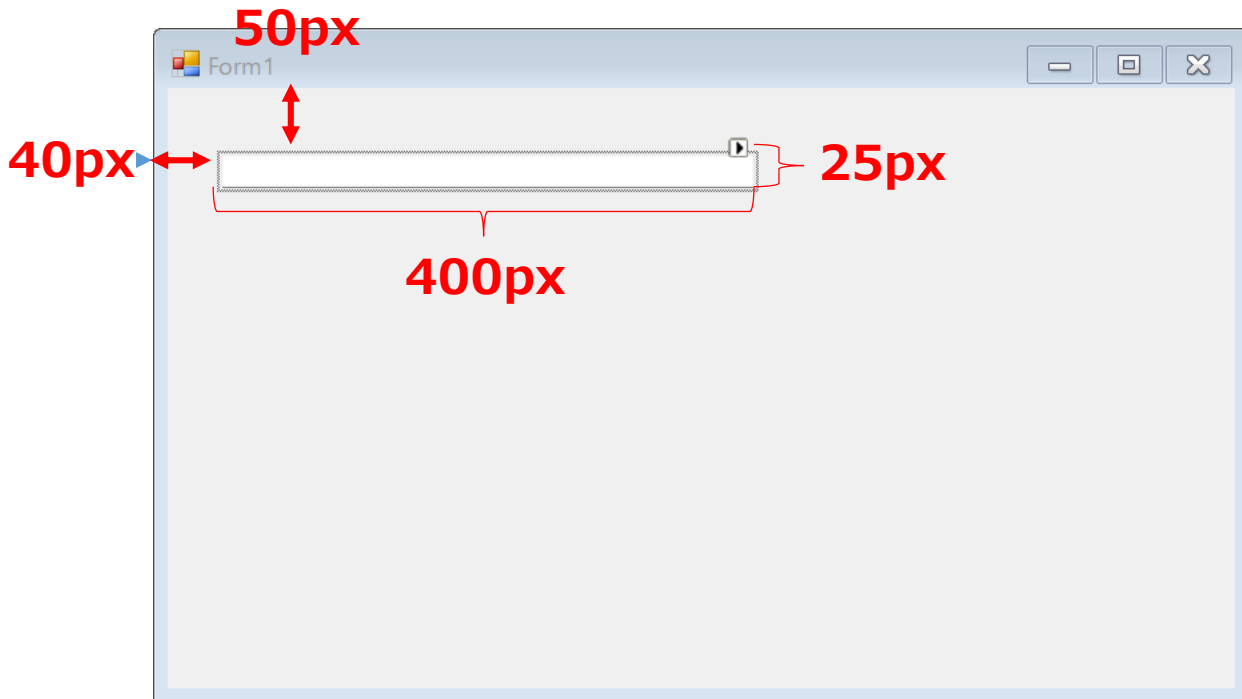


コントローラを選択しドラッグして配置する

テキストボックスの作成

テキストボックスの位置とサイズの調整

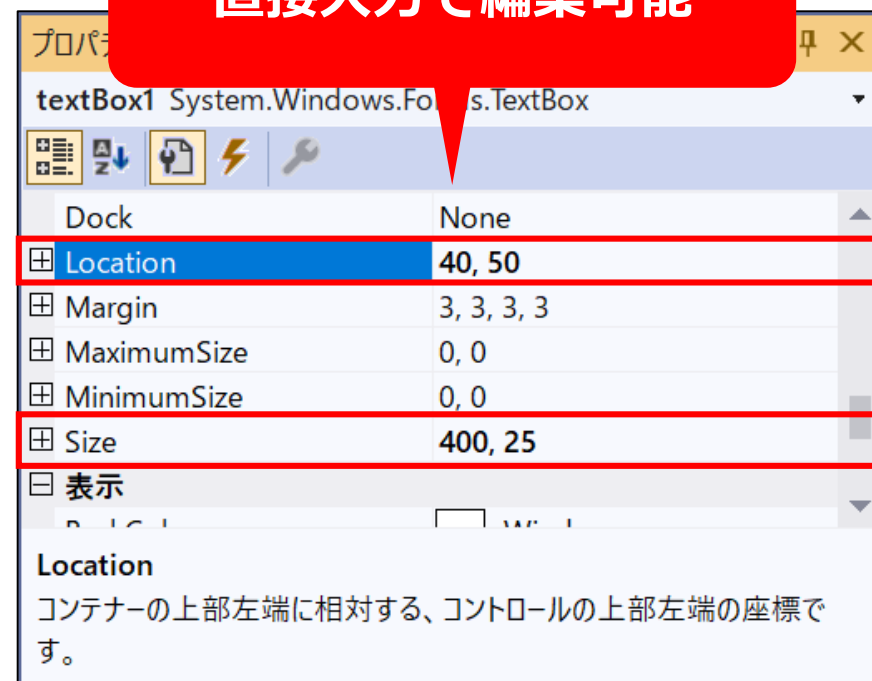
プロパティを利用して位置（Location）とサイズ（Size）の調整



テキストボックスの作成

テキストボックスの位置とサイズの調整

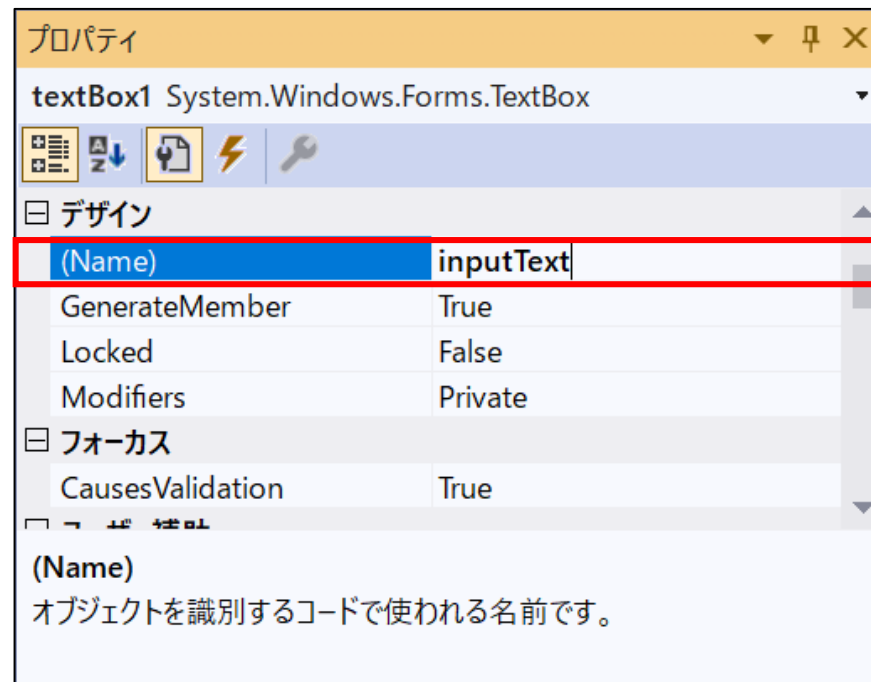
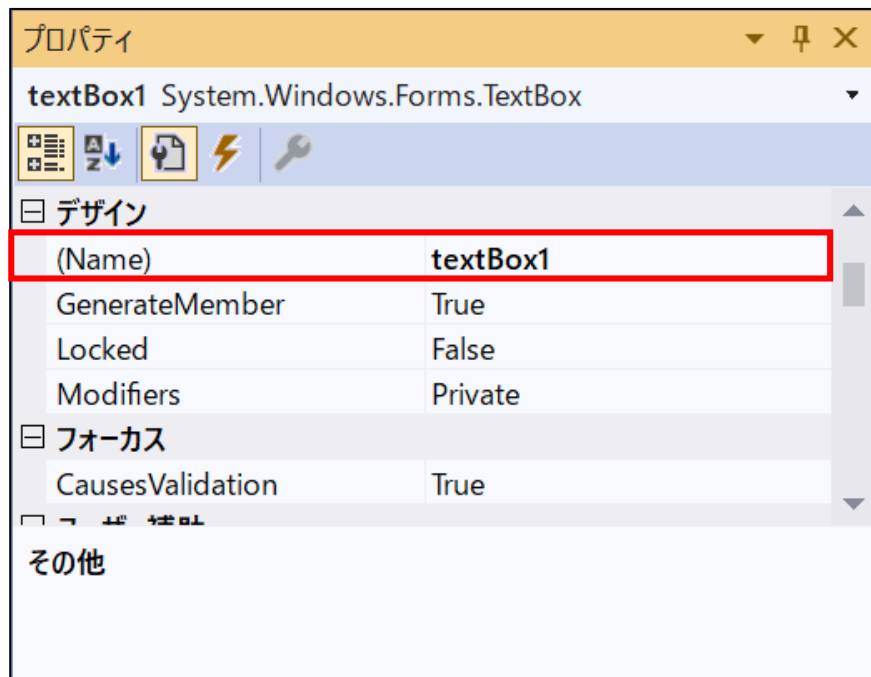
プロパティを利用して位置（Location）とサイズ（Size）の調整



テキストボックスの作成

テキストボックスの名前の変更

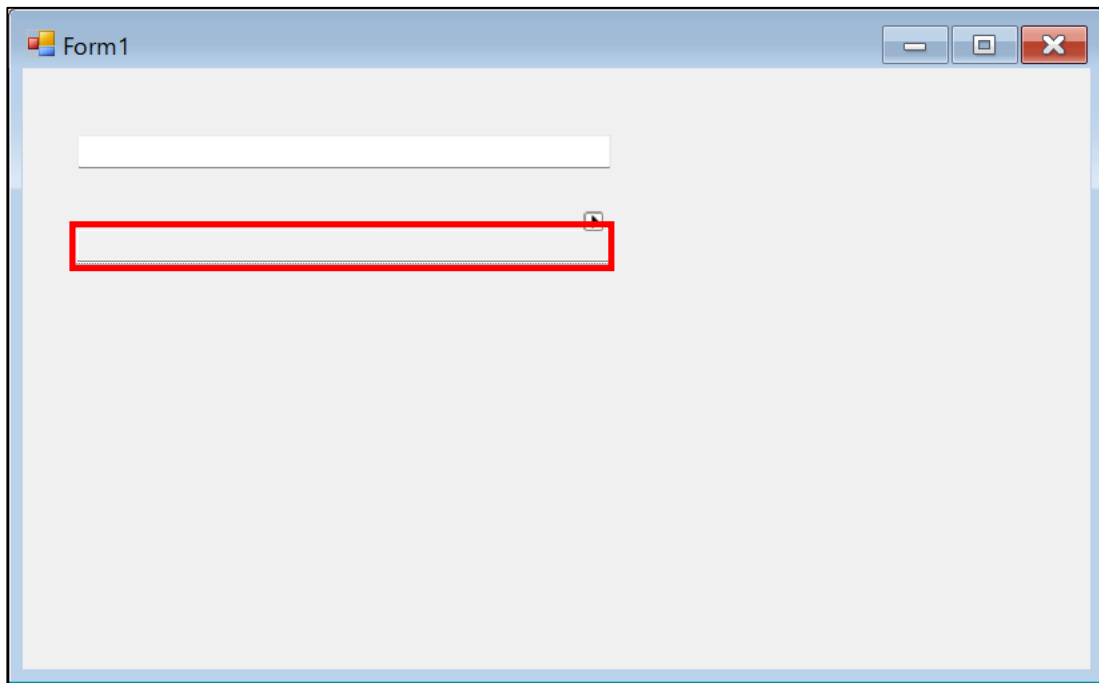
プロパティの「**name**」に名前 (**inputText**) を入力



テキストボックスの作成

2つ目のテキストボックスの配置

1つ目と同様に2つ目のテキストボックスを配置



Name: **outText**

Location: **40,120**

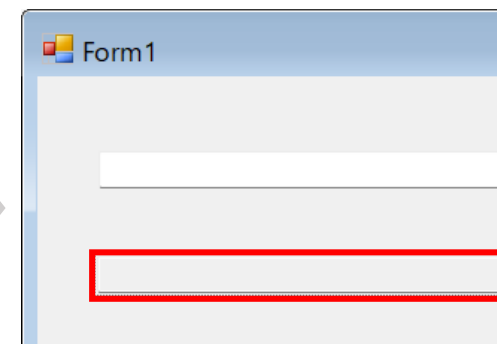
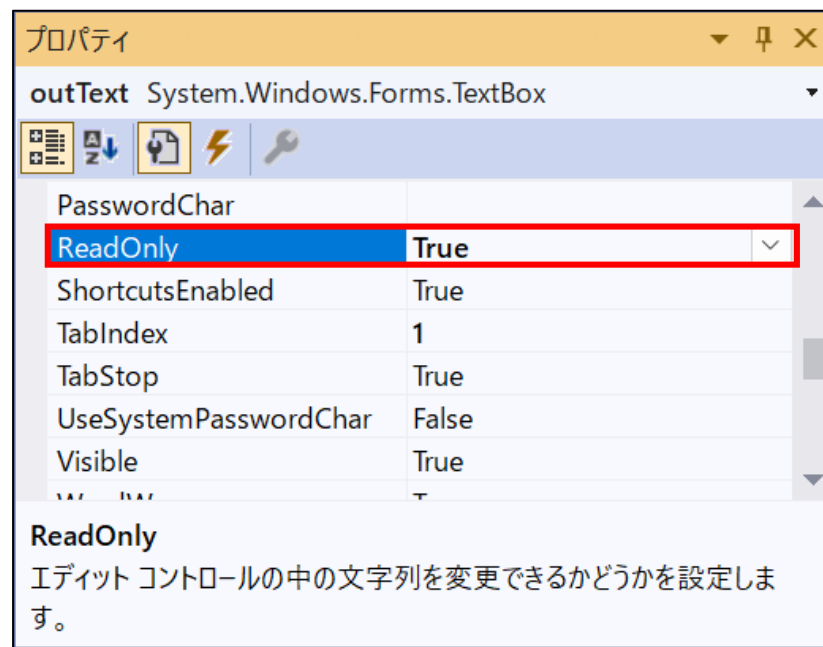
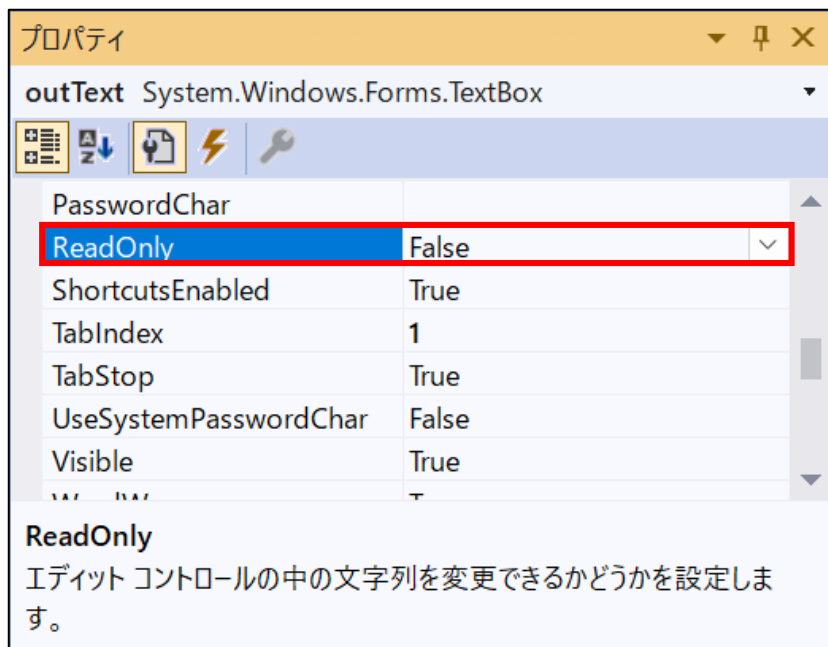
Size: **400,25**

プロパティで設定

テキストボックスの作成

2つ目のテキストボックスを読み取り専用に

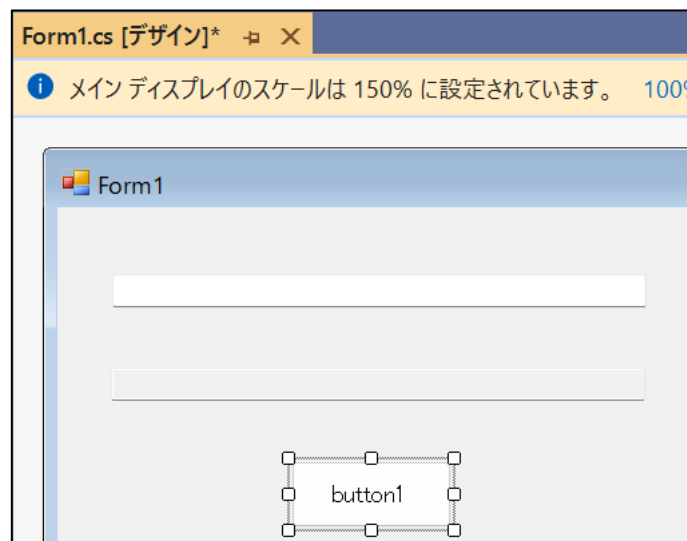
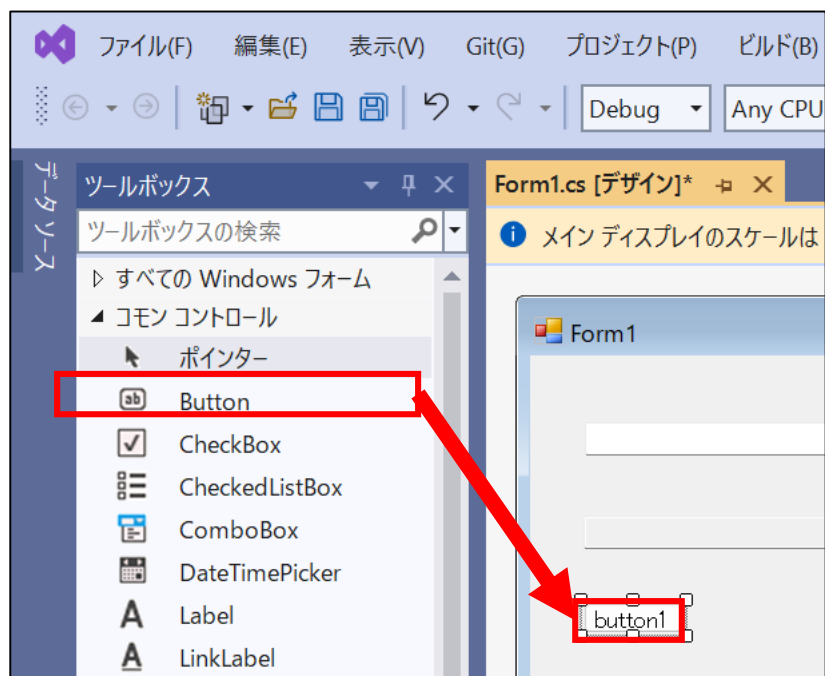
プロパティで「ReadOnly」の**False**を**True**に変更することで入力不可能にする



ボタンの作成

ボタンの配置

テキストボックスと同様にツールボックスから配置し位置・サイズを調節



Location: **175,190**

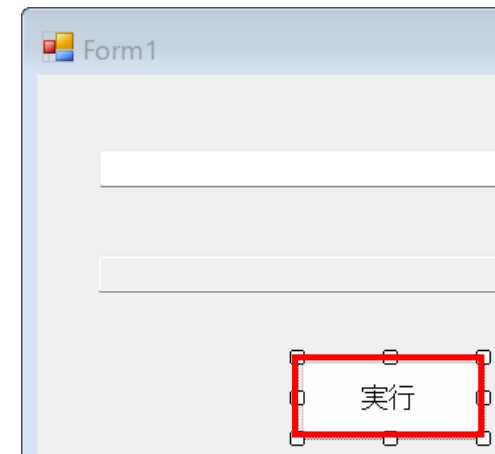
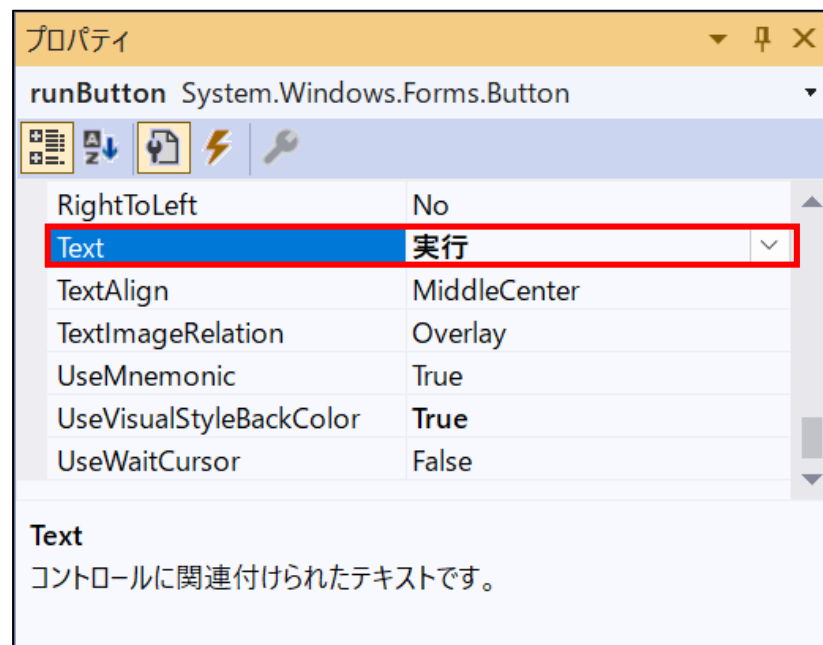
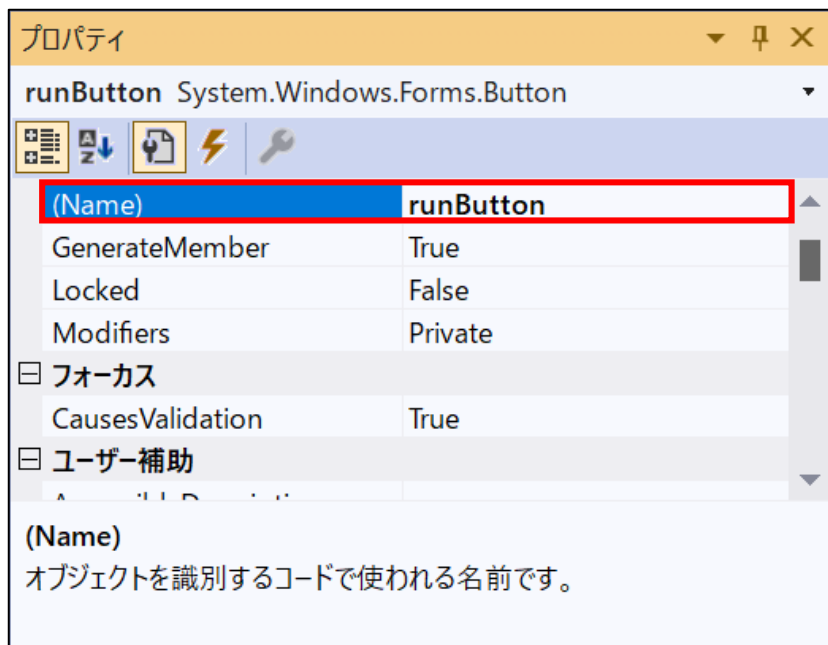
Size: **120,50**

プロパティで設定

ボタンの作成

ボタン名とボタンのテキストの変更

ボタン名を「runButton」、テキストを「実行」に変更

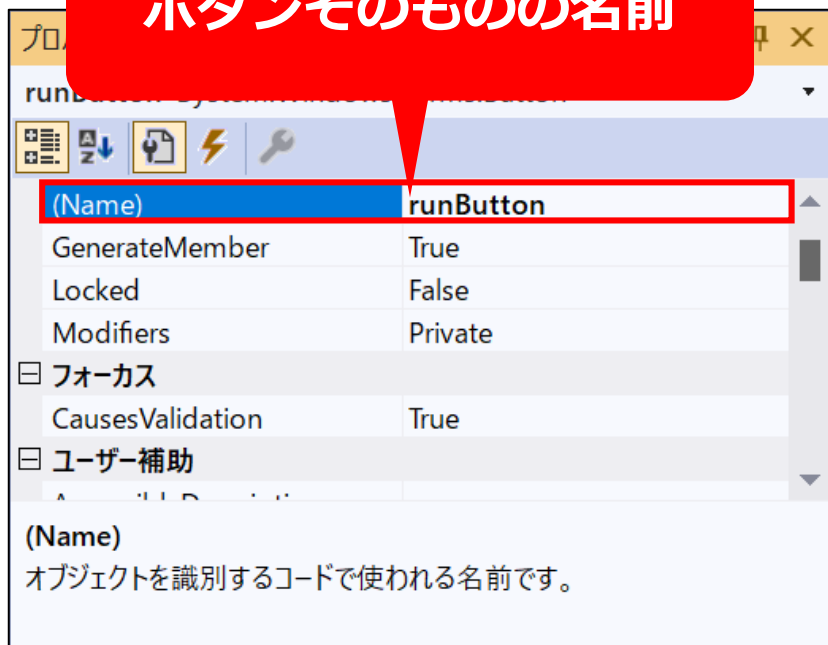


ボタンの作成

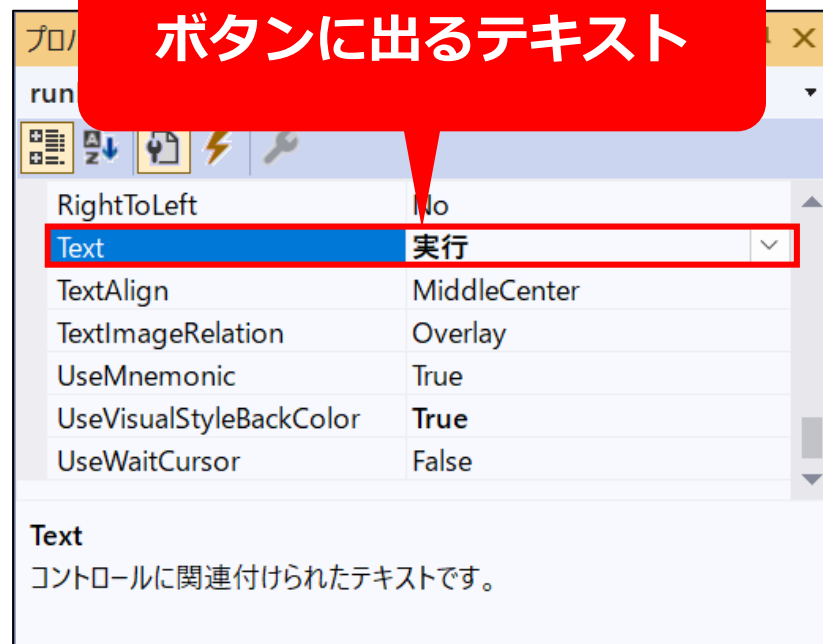
ボタン名とボタンのテキストの変更

ボタン名を「runButton」、テキストを「実行」に変更

ボタンそのものの名前



ボタンに出るテキスト



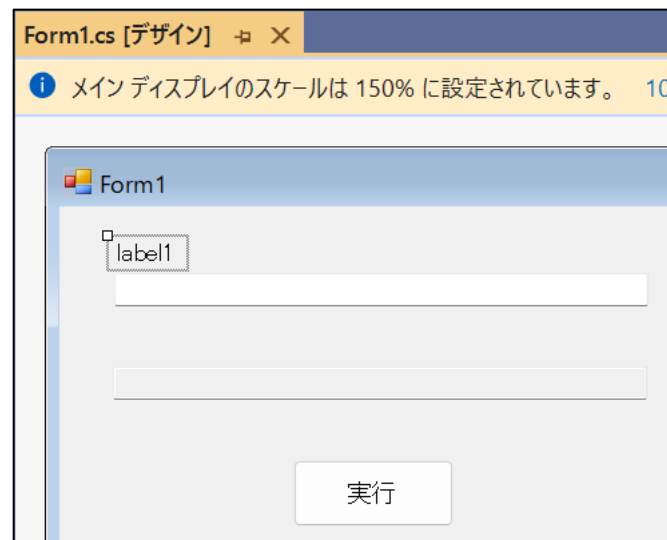
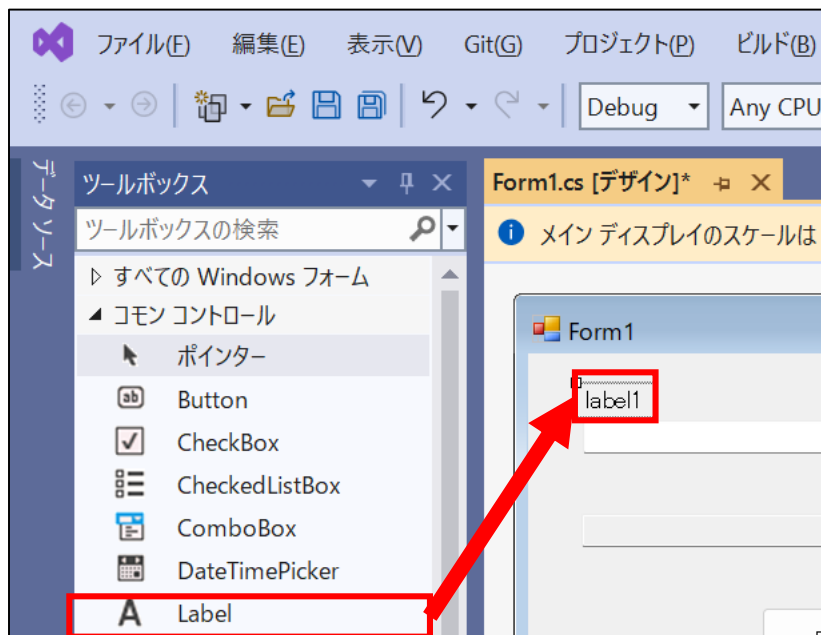
Textの内容



ラベルの作成

ラベルの配置

テキストボックス・ボタンと同様にツールボックスから配置し位置・サイズを調節



Location: **40,25**

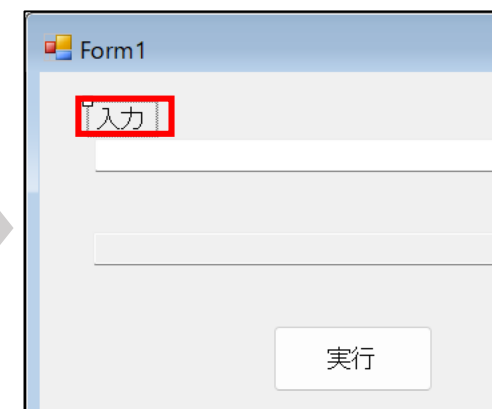
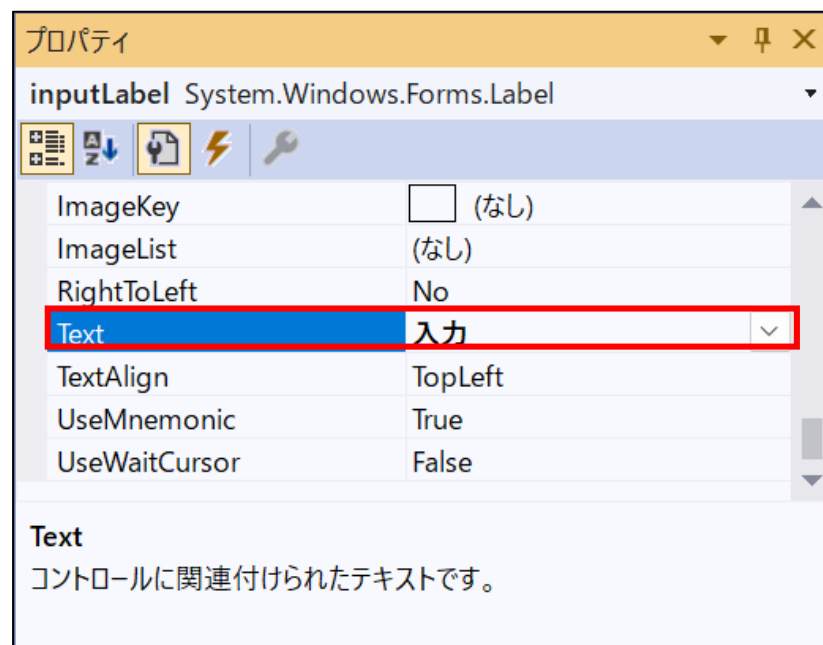
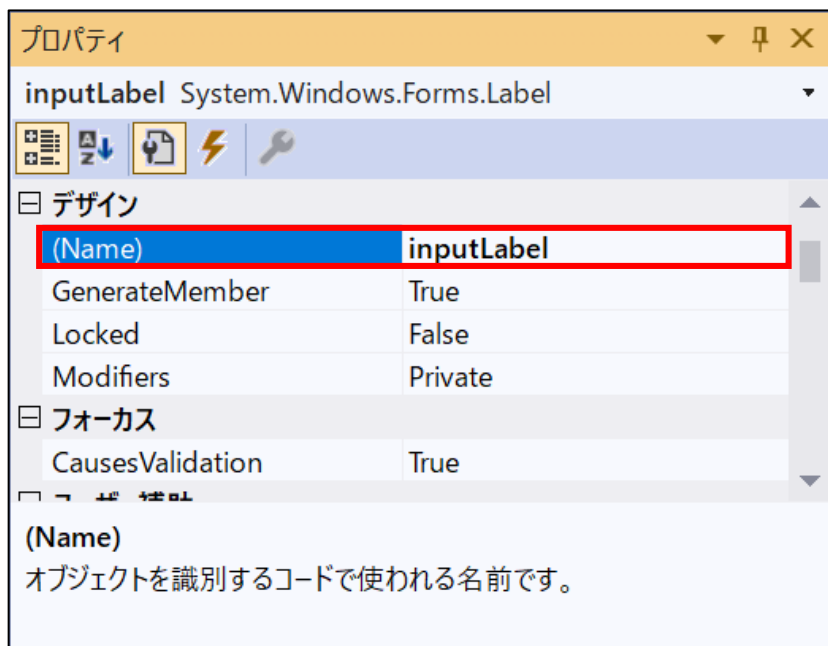
Size: **44,18**

プロパティで設定

ラベルの作成

ボタン名とボタンのテキストの変更

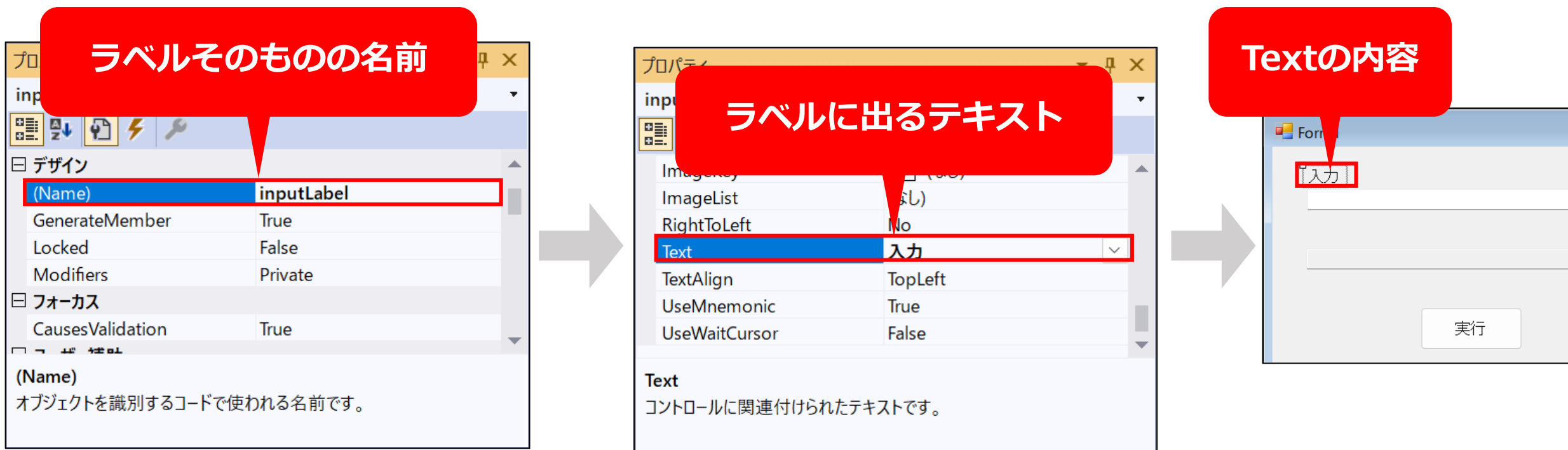
プロパティで「Name」を「inputLabel」、「Text」を「入力」に変更



ラベルの作成

ボタン名とボタンのテキストの変更

プロパティで「Name」を「inputLabel」、「Text」を「入力」に変更



ラベルの作成

2つ目のラベルの配置と設定

1つ目と同様に2つ目のラベルを配置しプロパティで各種設定を行う



Name: **outputLabel**

Text: **出力**

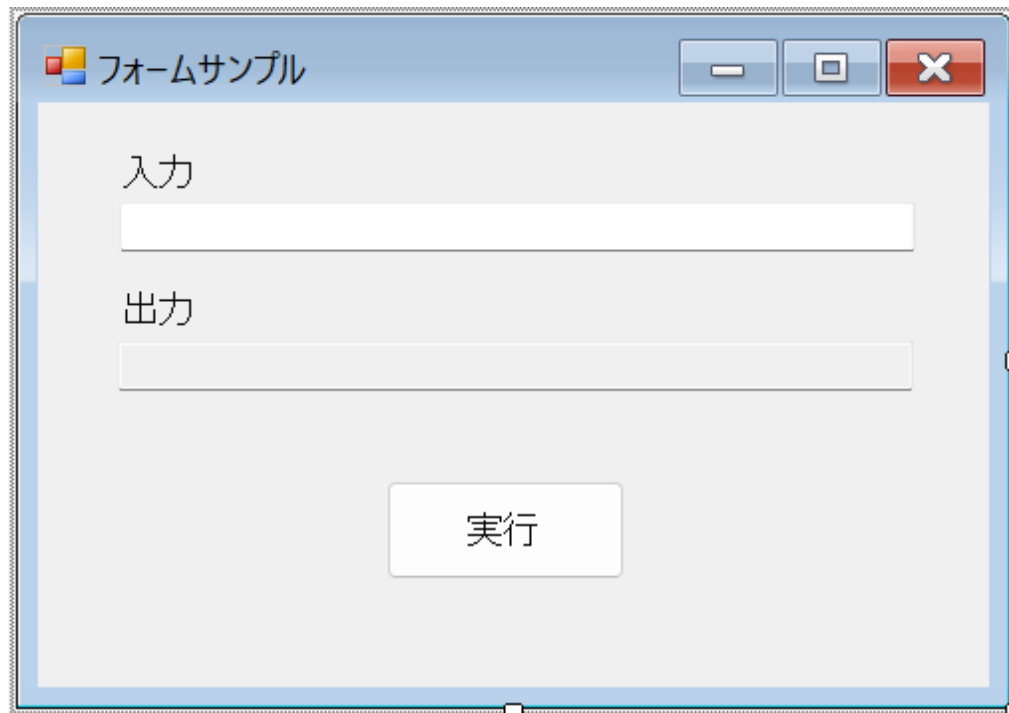
Location: **40,95**

Size: **44,18**

フォームの調整

フォームのサイズを変更

最後にフォームを選択しフォームのプロパティを設定して外観を完成させる

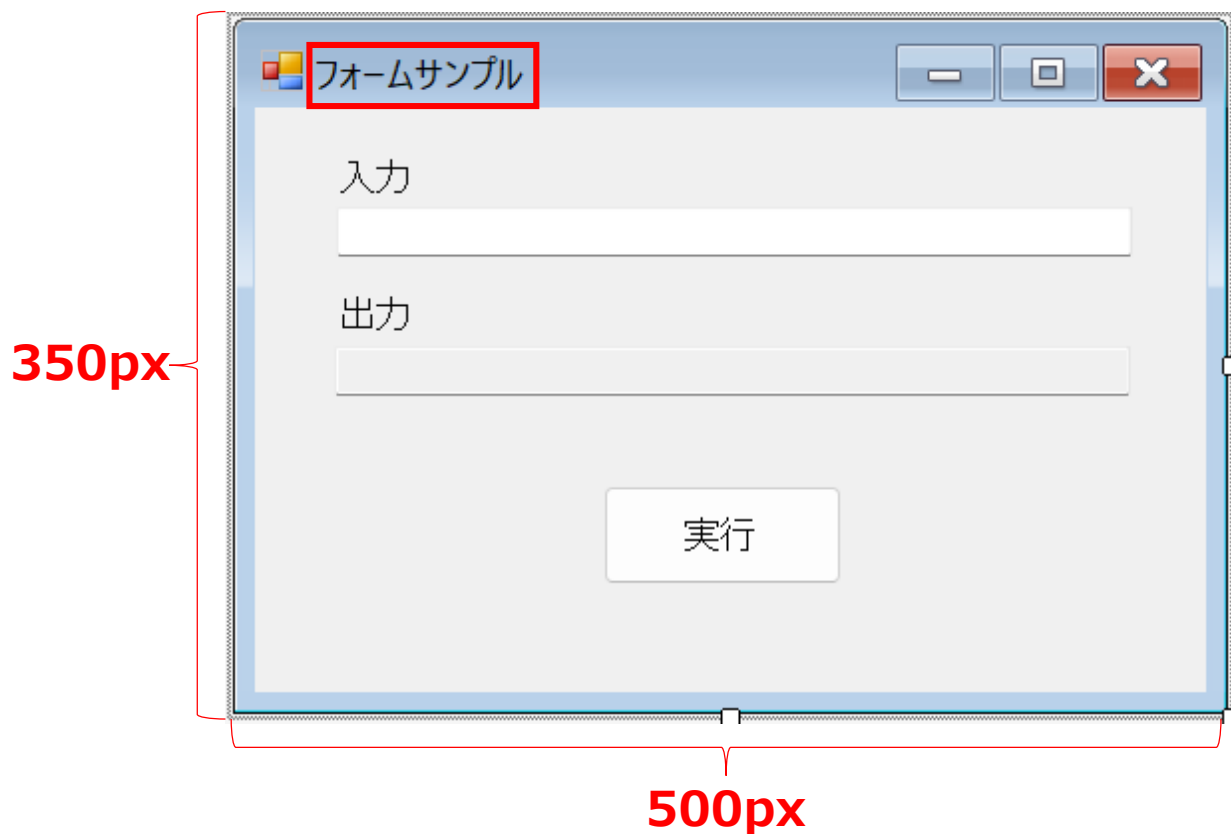


Name: **FormSample**
Text: **フォームサンプル**
Size: **500,350**

フォームの調整

フォームのサイズを変更

最後にフォームを選択しフォームのプロパティを設定して外観を完成させる



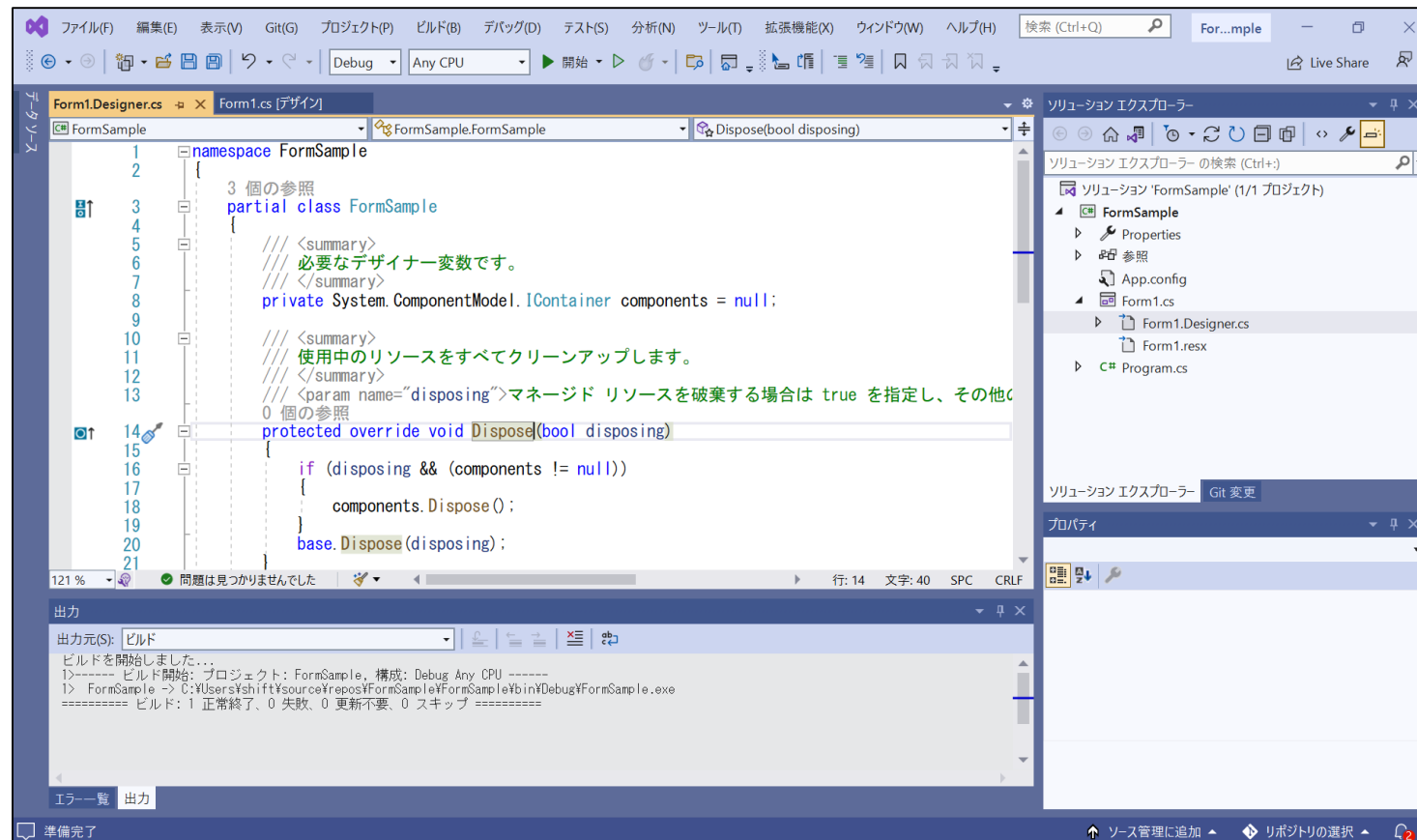
Name: **FormSample**
Text: **フォームサンプル**
Size: **500,350**

フォームを選択し入力

各種csファイルの構成確認

Form1.Designers.cs

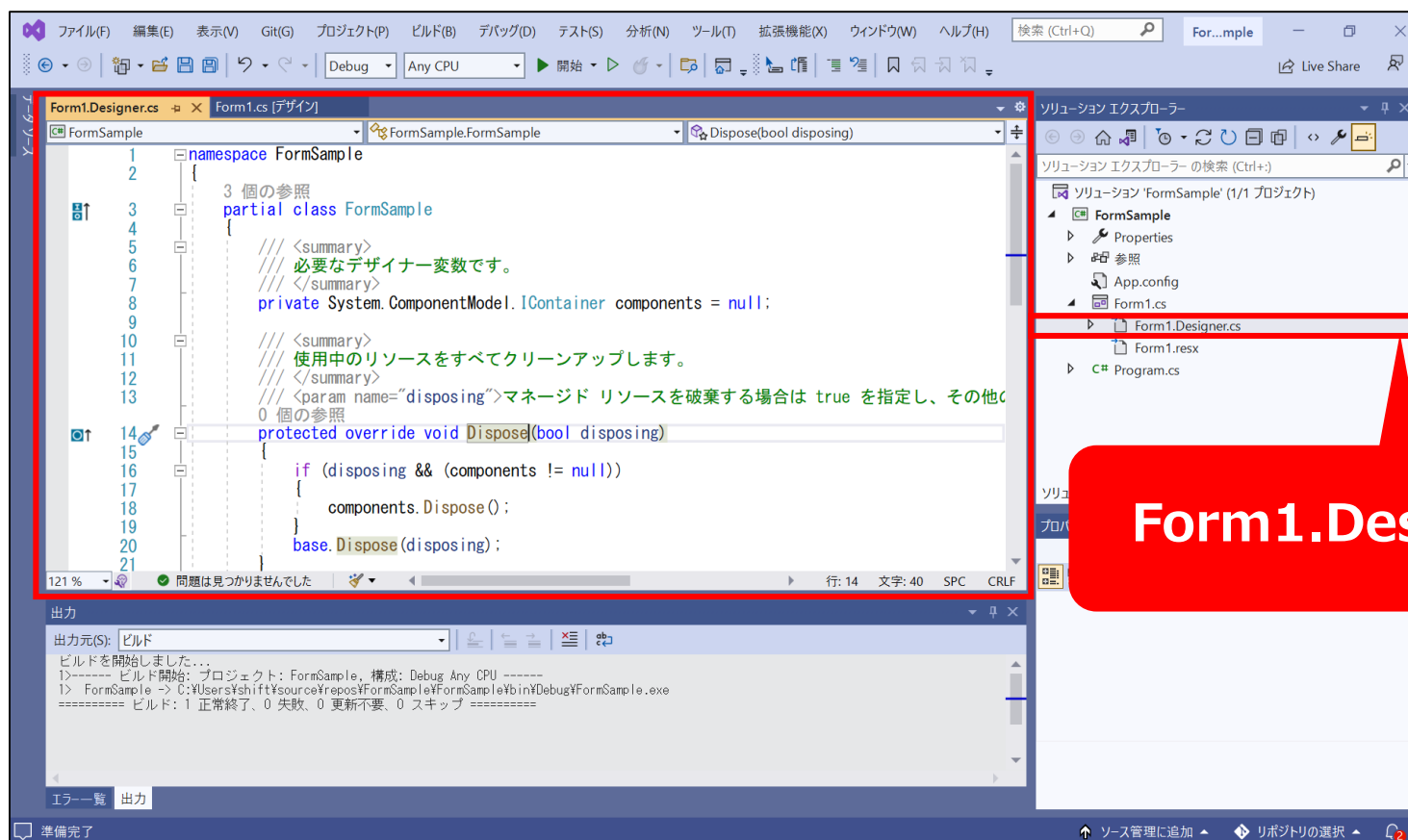
作成したフォームに対応したソースファイル



各種csファイルの構成確認

Form1.Designers.cs

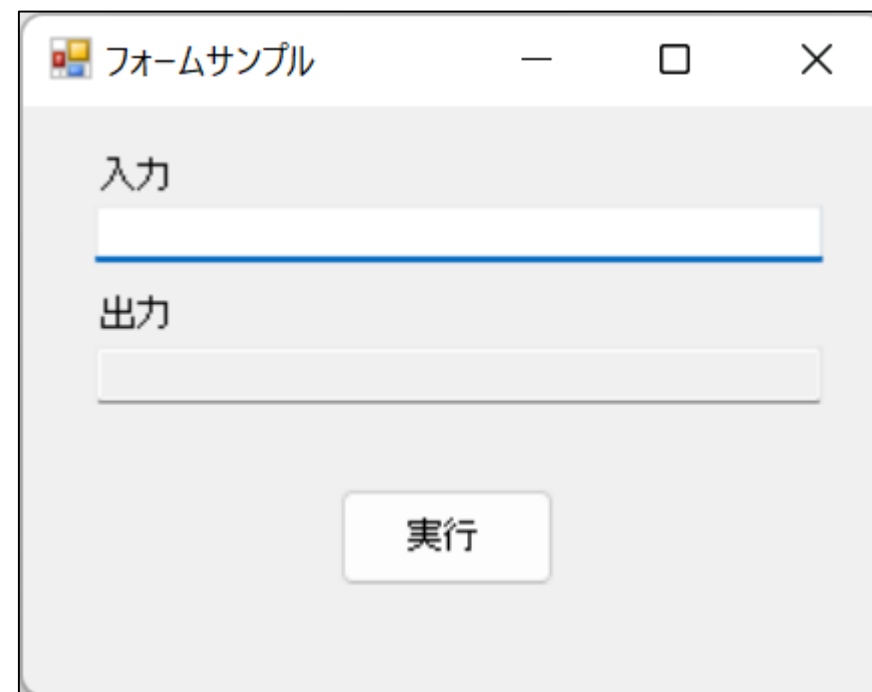
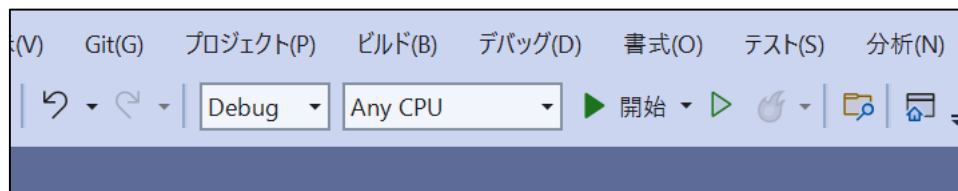
作成したフォームに対応したソースファイル



動作の確認

フォームのサイズを変更

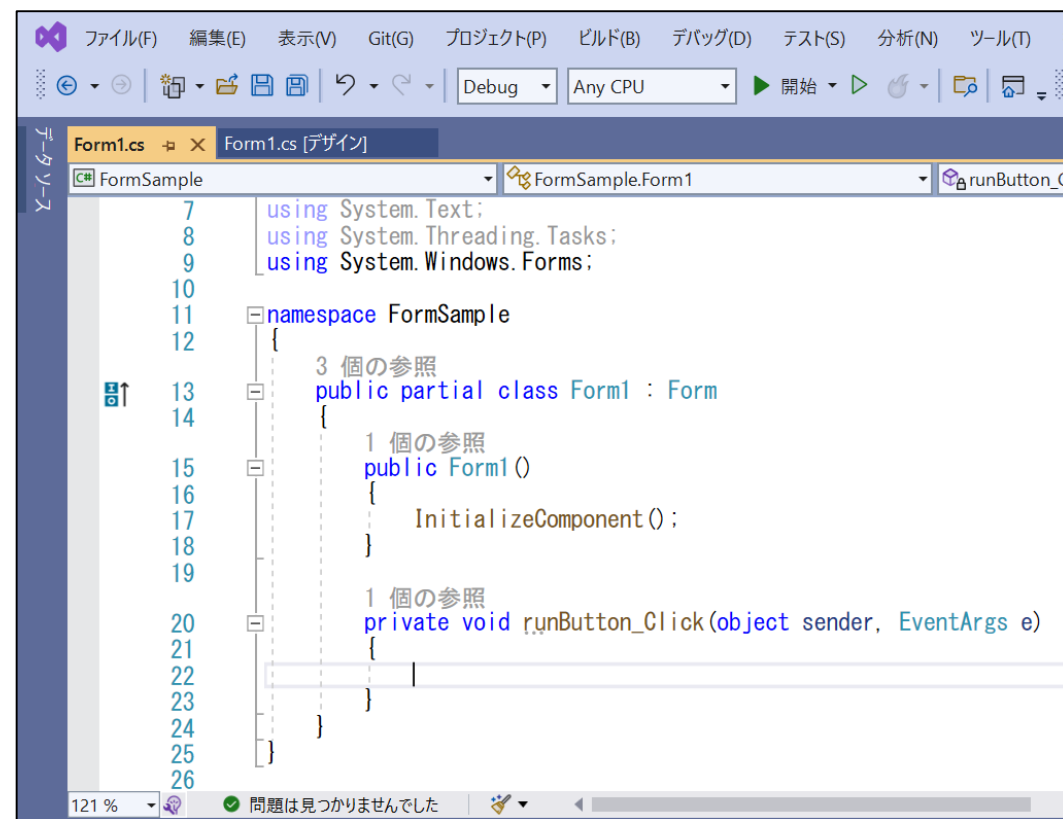
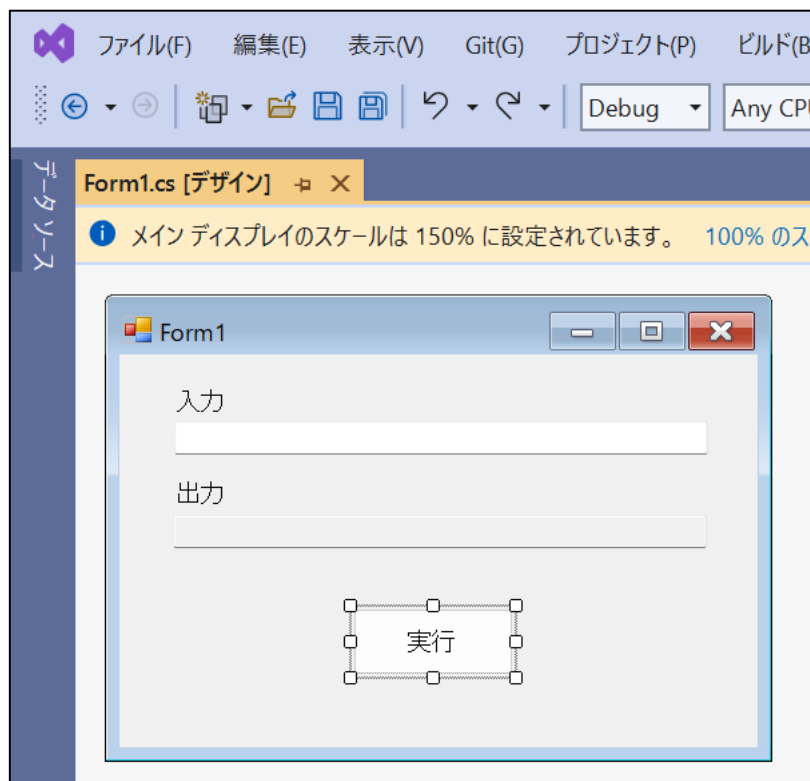
「デバッグなしで開始」ボタンもしくは[CTRL]+[F5]キーでプログラムが実行される



イベントハンドラの作成

実行ボタンをダブルクリック

「実行」 ボタンをダブルクリックしてボタンがダブルクリックした時の処理を追加



イベントハンドラの作成

フォームに配置したボタン操作などの処理には**イベントハンドラ**が必要

イベントハンドラとは何か

- ・ **イベントとは？** ... GUIのボタンを押す・マウスが乗るなどの様々な操作。
- ・ **イベントハンドラとは？** ... イベントが発生したときに呼び出されるメソッド。
- ・ **フォームアプリとイベントアプリ** ... イベント発生時にのみイベントハンドラを実行。

イベントハンドラの作成

フォームに配置したボタン操作などの処理には**イベントハンドラ**が必要

イベントハンドラとは何か

- ・ **イベントとは？** ... GUIのボタンを押す・マウスが乗るなどの様々な操作。
- ・ **イベントハンドラとは？** ... イベントが発生したときに呼び出されるメソッド。
- ・ **フォームアプリとイベントアプリ** ... イベント発生時にのみイベントハンドラを実行。

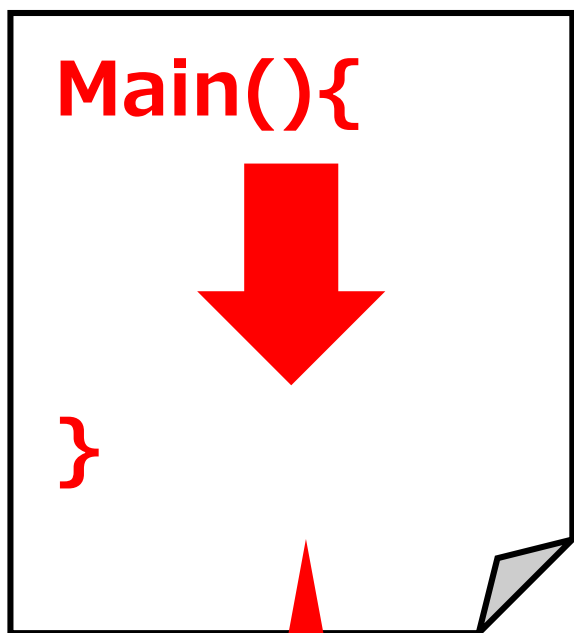
！ポイント

イベントハンドラは処理が必要なイベントの種類・数に応じて追加できる

イベントハンドラの作成

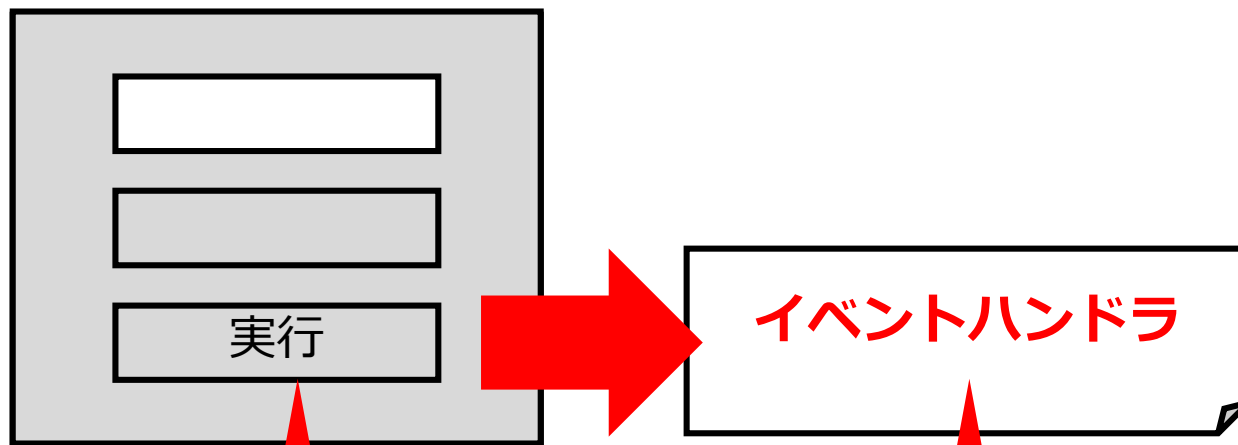
コンソールアプリとフォームアプリの根本的な違い

▼コンソールアプリ



原則的に上から下に実行

▼フォームアプリ



①イベントの発生を監視

②イベント発生時に実行

イベントハンドラの作成

イベント処理の記述

出現したイベントハンドラの部分にイベント処理を記述

```
13 public partial class Form1 : Form
14 {
15     1 個の参照
16     public Form1()
17     {
18         InitializeComponent();
19     }
20     1 個の参照
21     private void runButton_Click(object sender, EventArgs e)
22     {
23         // 入力されたテキストの取得
24         string txt = this.inputText.Text;
25         // 出力の所に入力した値を代入
26         this.outText.Text = txt;
27     }
28 }
```

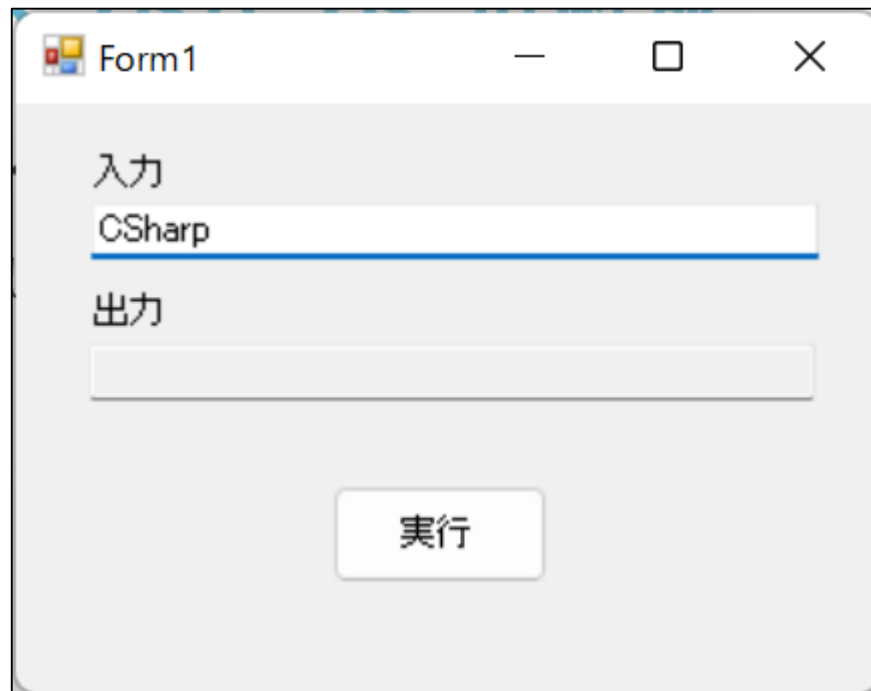
▼記述内容

```
// 入力されたテキストの取得
string txt = this.inputText.Text;
// 出力の所に入力した値を代入
this.outText.Text = txt;
```

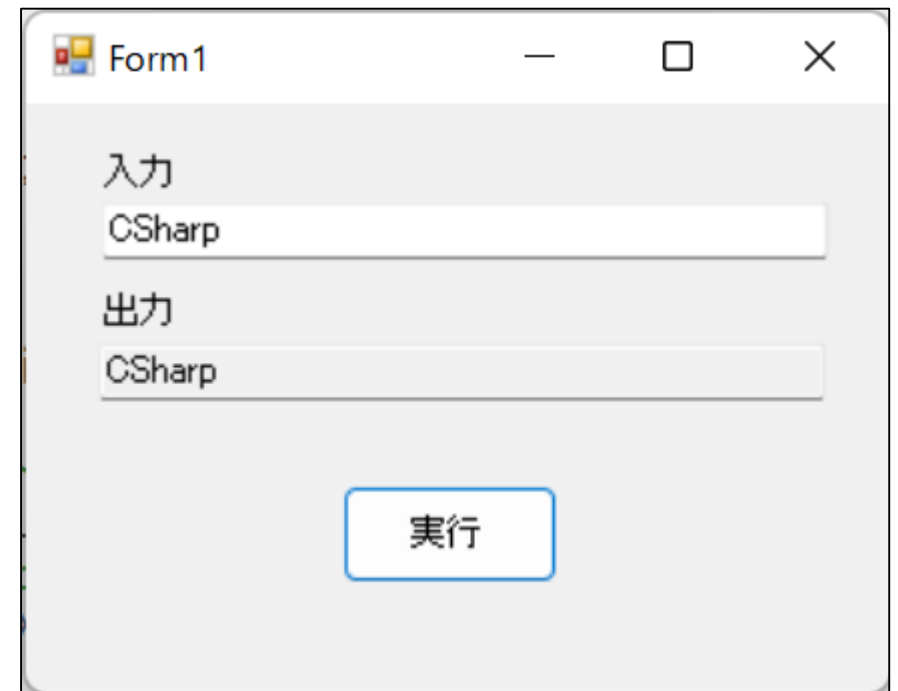
イベントハンドラの作成

プログラムの実行と確認

実行して入力欄にテキストを入力して「実行」ボタンをクリックすると出力部分にあらわれる



A screenshot of a Windows form titled 'Form1'. It contains two text input fields. The first field, labeled '入力' (Input), contains the text 'CSharp'. The second field, labeled '出力' (Output), is empty. Below the fields is a button labeled '実行' (Execute).

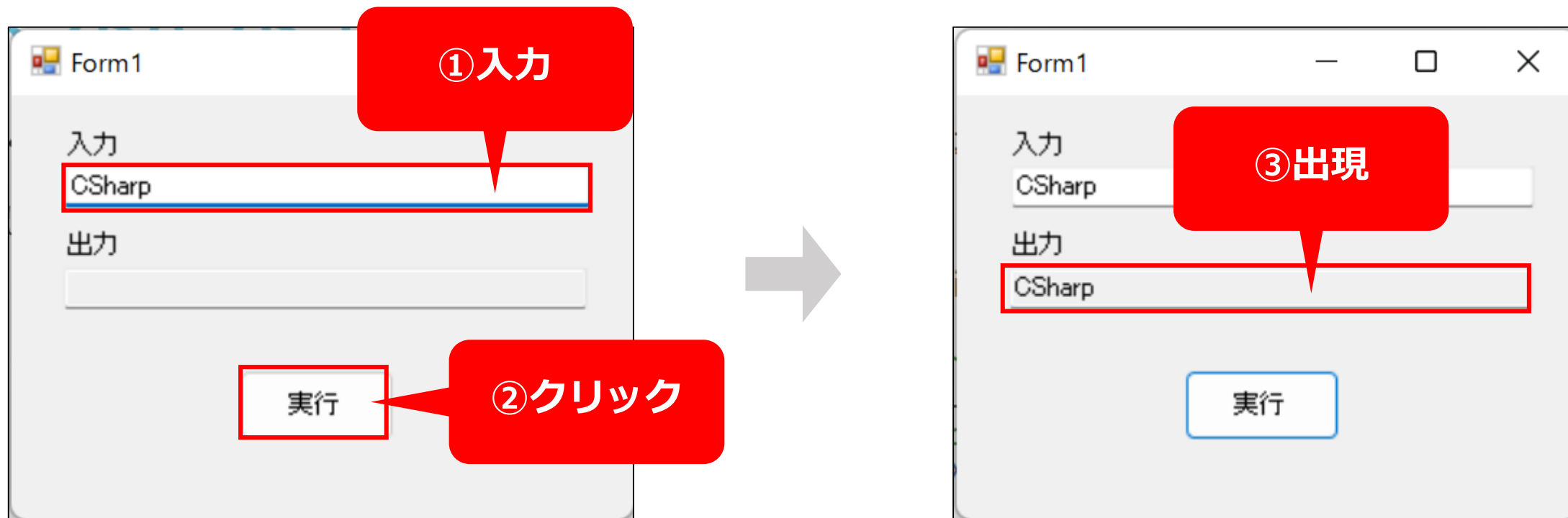


A screenshot of the same Windows form 'Form1' after execution. The '入力' (Input) field still contains 'CSharp'. The '出力' (Output) field now also contains the text 'CSharp'. The '実行' (Execute) button is highlighted with a blue border.

イベントハンドラの作成

プログラムの実行と確認

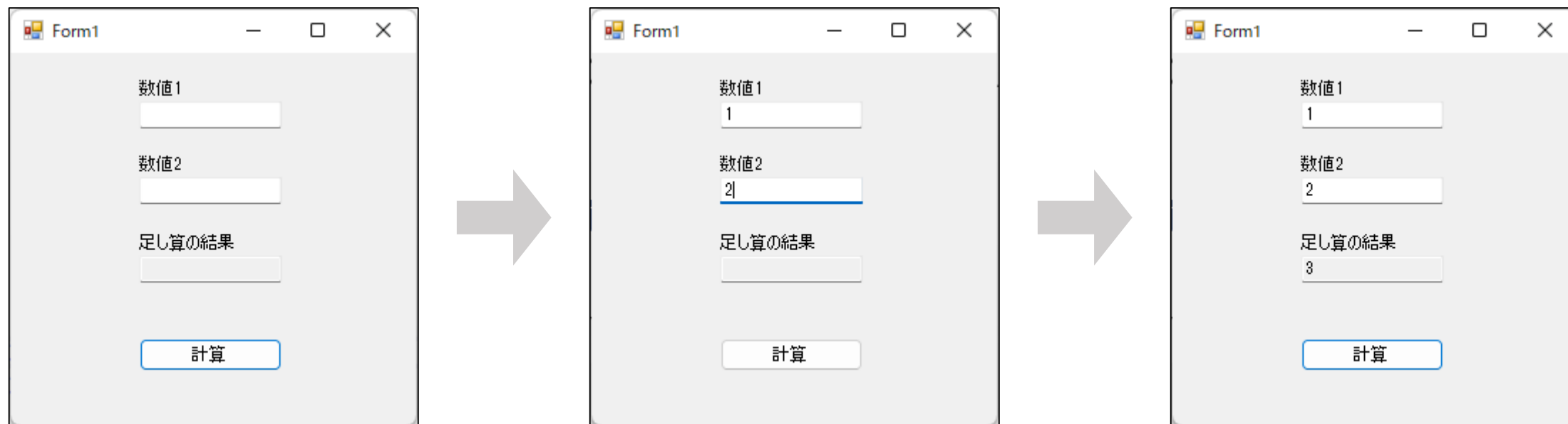
実行して入力欄にテキストを入力して「実行」ボタンをクリックすると出力部分にあらわれる



演習問題1

C#3で作成したCalcクラスを用いて足し算を行うプログラムを作ってみましょう

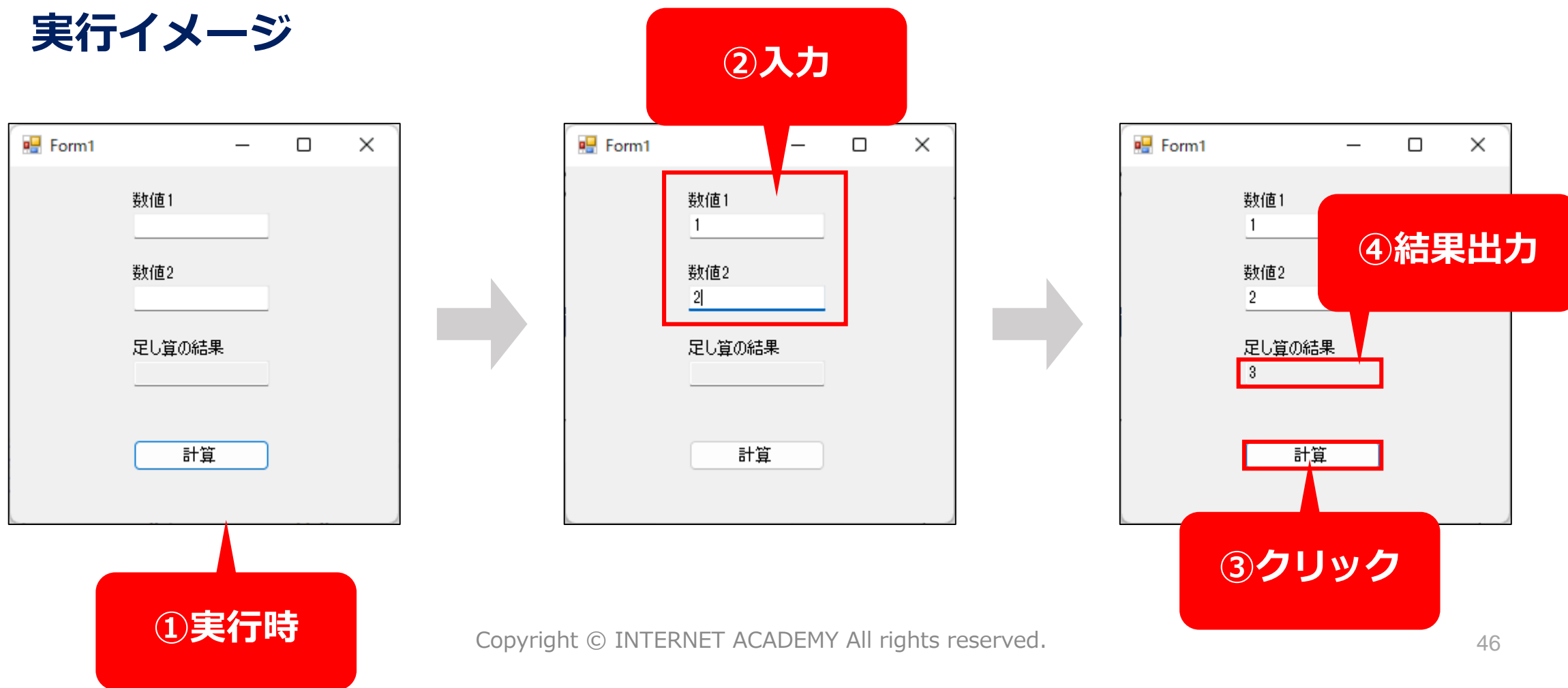
実行イメージ



演習問題1

C#3で作成したCalcクラスを用いて足し算を行うプログラムを作る

実行イメージ



演習問題1

手順①プロジェクトの作成

C# 用の [Windows フォーム アプリケーション (.NET Framework)] テンプレートを選択



新しいプロジェクトを構成します

Windows フォーム アプリケーション (.NET Framework) C# Windows デスクトップ

プロジェクト名(P)

Calculator

場所(L)

C:\Users\shift\source\repos

ソリューション(S)

新しいソリューションを作成する

ソリューション名(M) ⓘ

Calculator

☐ ソリューションとプロジェクトを同じディレクトリに配置する(O)

フレームワーク(F)

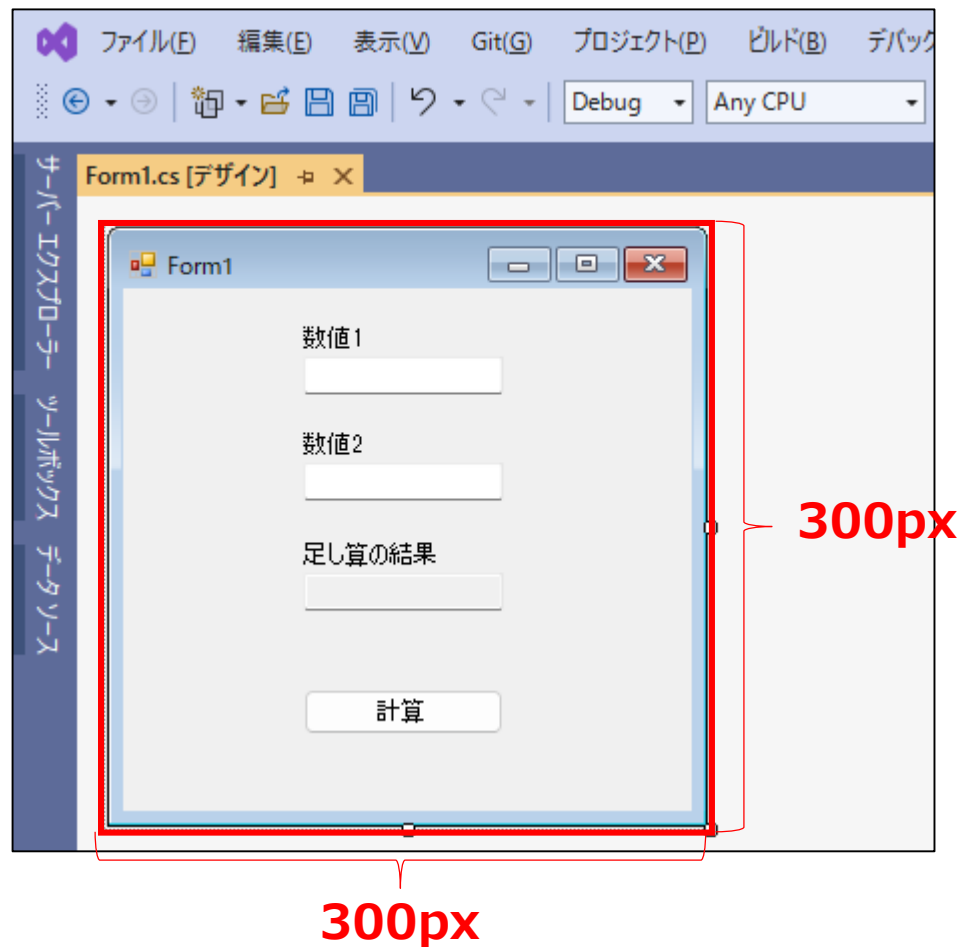
.NET Framework 4.7.2

戻る(B) 作成(C)

プロジェクト名を入力し
「作成」ボタンをクリック

演習問題1

手順②フォームの作成



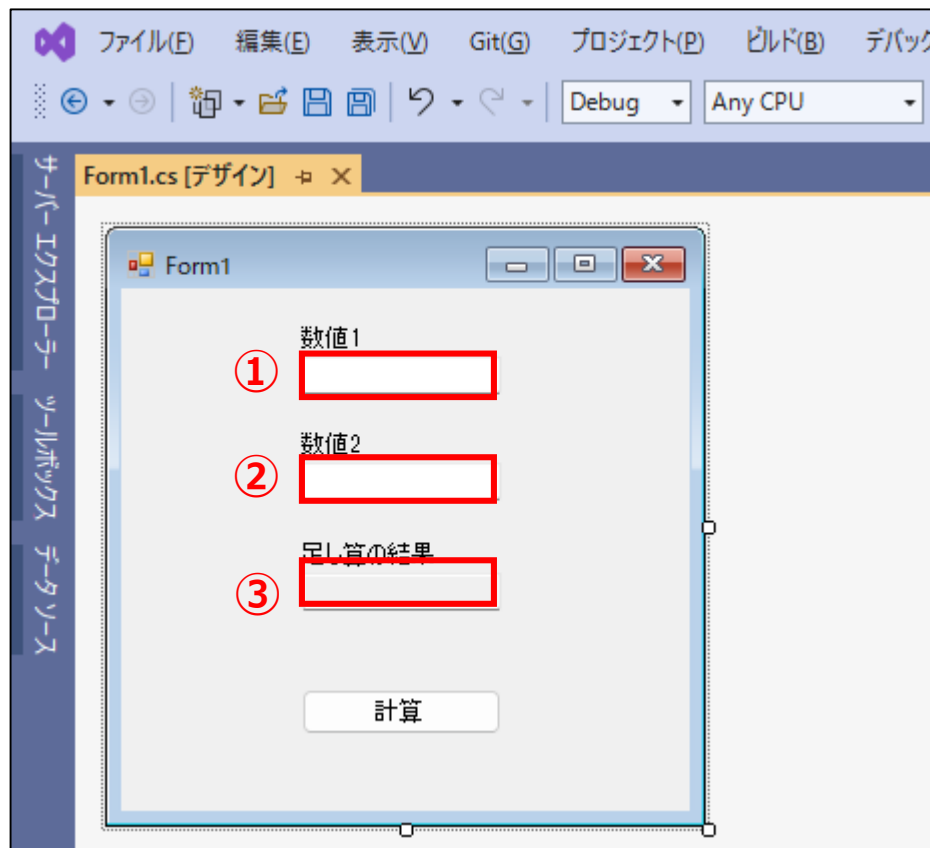
Name: **CalcForm**

Size: **300,300**

フォームを選択し入力

演習問題1

手順③テキストボックスの配置



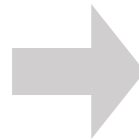
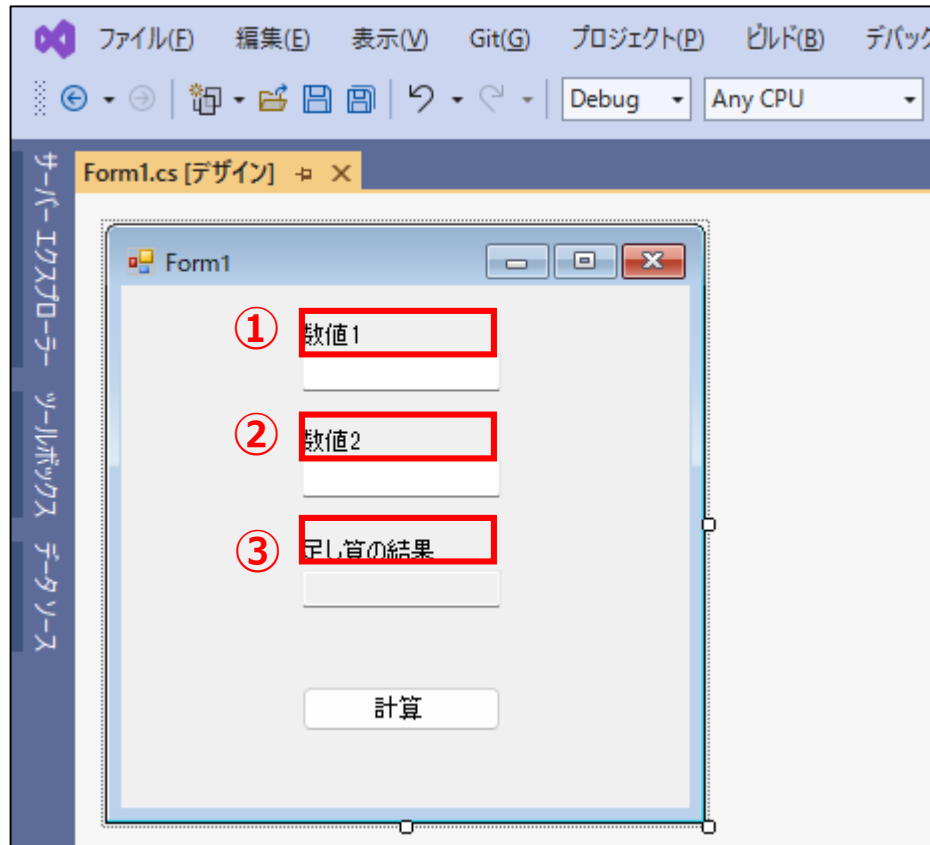
① Name: **number1**
Location: **90,34**
Size: **100,19**

② Name: **number2**
Location: **90,87**
Size: **100,19**

③ Name: **add_result**
Location: **90,142**
Size: **100,19**
ReadOnly: **True**

演習問題1

手順④ラベルの配置



①

Name: **label1**

Location: **88,19**

Text: **数値1**

②

Name: **label2**

Location: **88,72**

Text: **数値1**

③

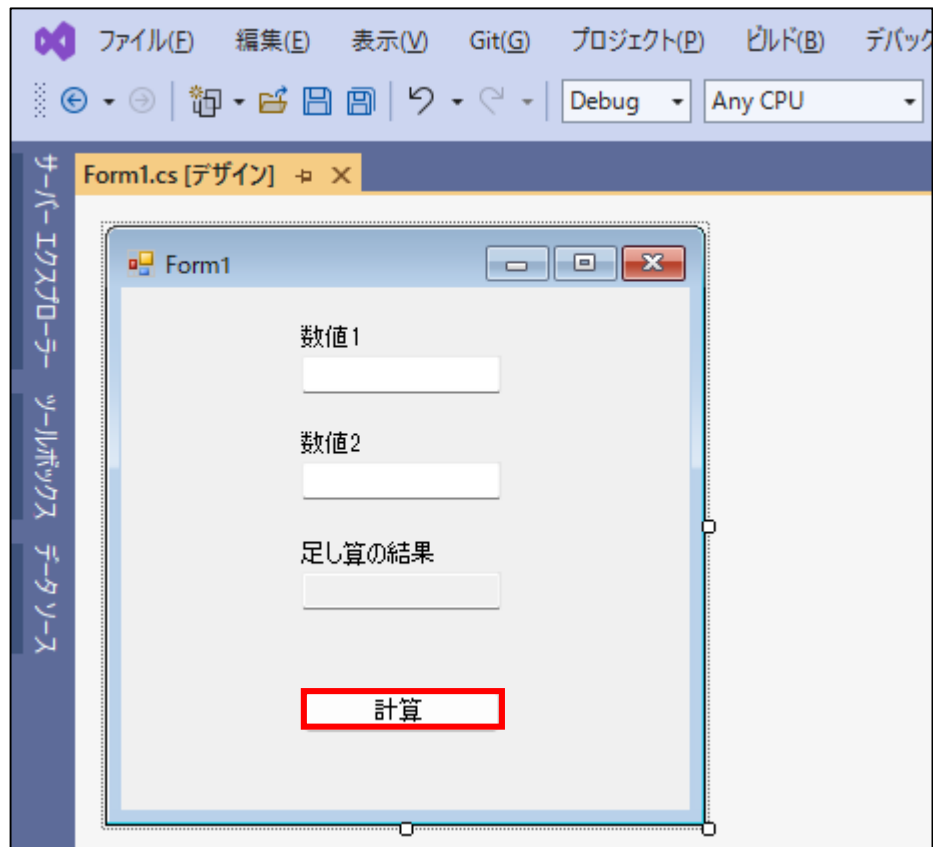
Name: **label3**

Location: **88,127**

Text: **足し算の結果**

演習問題1

手順⑤ ボタンの配置



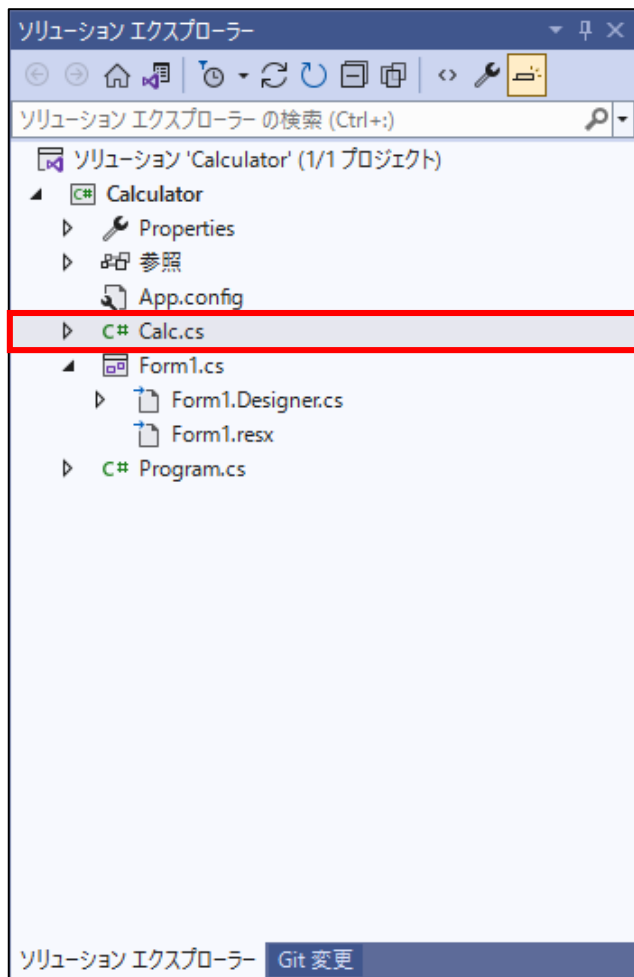
Name: **calc**

Location: **90,200**

Size: **300,300**

演習問題1

手順⑥Calcクラスの追加

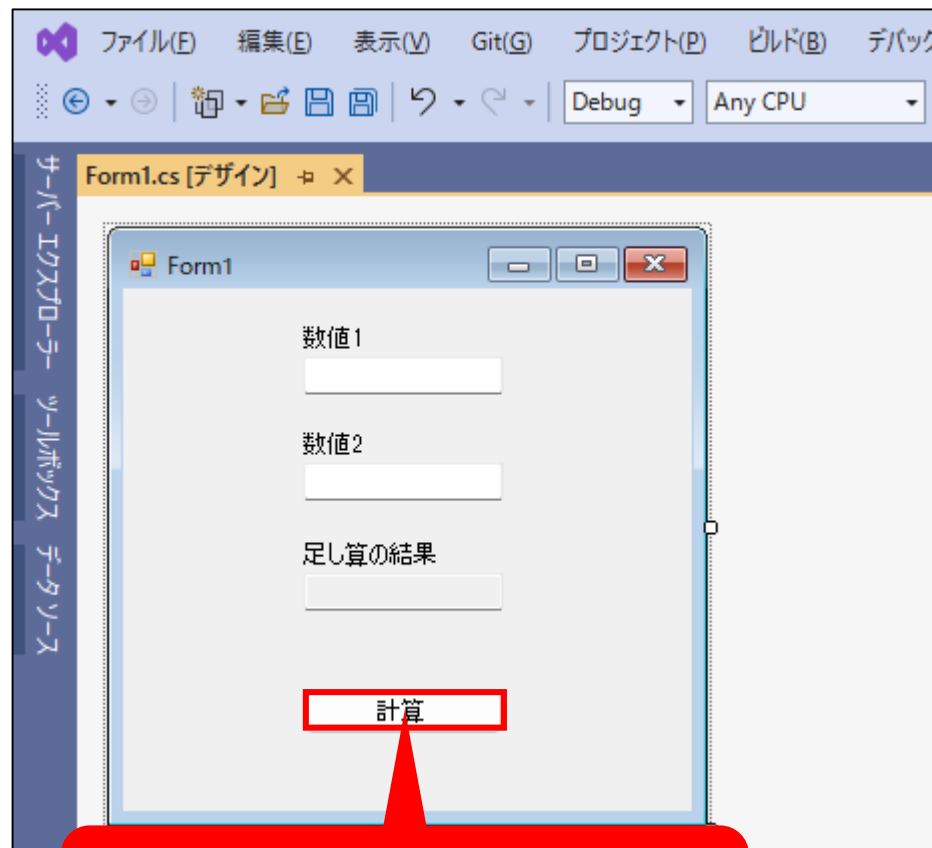


▼作成するクラス（Calcクラス）

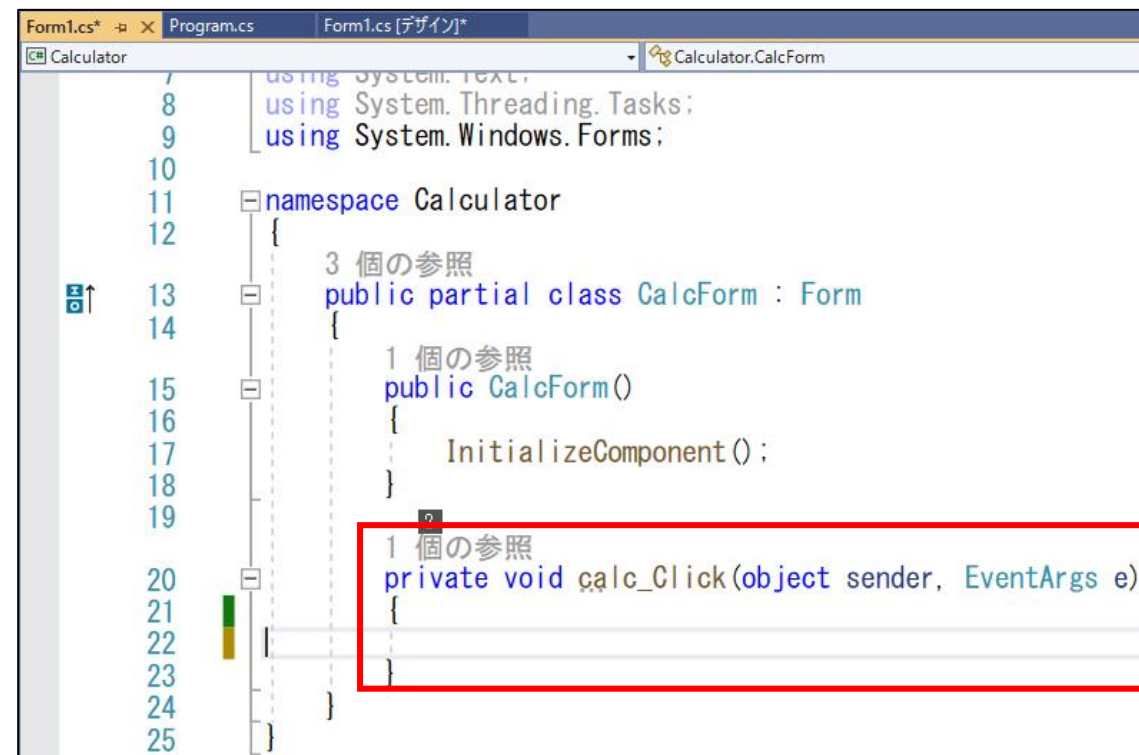
```
public class Calc
{
    public int num1 = 0;    // 1つ目の数
    public int num2 = 0;    // 2つ目の数
    // 足し算の結果を得るメソッド
    public int Add()
    {
        return this.num1 + this.num2;
    }
}
```

演習問題1

手順⑦ イベントハンドラの追加



ダブルクリック



演習問題1

手順⑧イベント処理の追加

▼calc_Clickメソッドに追加する処理

```
Calc calc = new Calc();  
calc.num1 = int.Parse(number1.Text);  
calc.num2 = int.Parse(number2.Text);  
int result = calc.Add();  
add_result.Text = result.ToString();
```

演習問題1

手順⑧ イベント処理の追加

▼ calc_Clickメソッドに追加する処理

```
Calc calc = new Calc();  
calc.num1 = int.Parse(number1.Text);  
calc.num2 = int.Parse(number2.Text);  
int result = calc.Add();  
add_result.Text = result.ToString();
```

Calcクラスのインスタンス生成

演習問題1

手順⑧ イベント処理の追加

▼ calc_Clickメソッドに追加する処理

```
Calc calc = new Calc();  
calc.num1 = int.Parse(number1.Text);  
calc.num2 = int.Parse(number2.Text);  
int result = calc.Add();  
add_result.Text = result.ToString();
```

数値の取得

！ ポイント

入力された値はテキスト（string）なのでint.Parseで整数に変換する

演習問題1

手順⑧ イベント処理の追加

▼ calc_Clickメソッドに追加する処理

```
Calc calc = new Calc();  
calc.num1 = int.Parse(number1.Text);  
calc.num2 = int.Parse(number2.Text);  
int result = calc.Add();  
add_result.Text = result.ToString();
```

計算結果の取得

演習問題1

手順⑧ イベント処理の追加

▼ calc_Clickメソッドに追加する処理

```
Calc calc = new Calc();  
calc.num1 = int.Parse(number1.Text);  
calc.num2 = int.Parse(number2.Text);  
int result = calc.Add();  
add_result.Text = result.ToString();
```

結果の出力

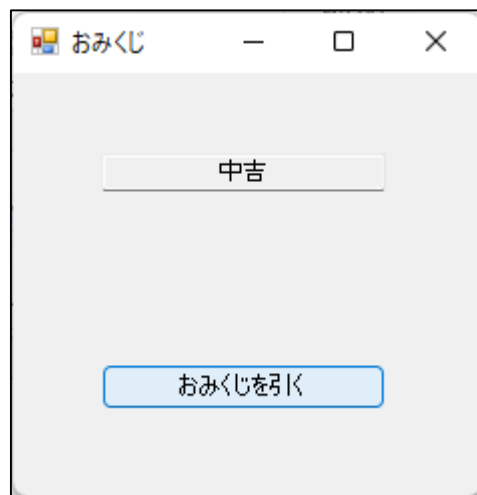
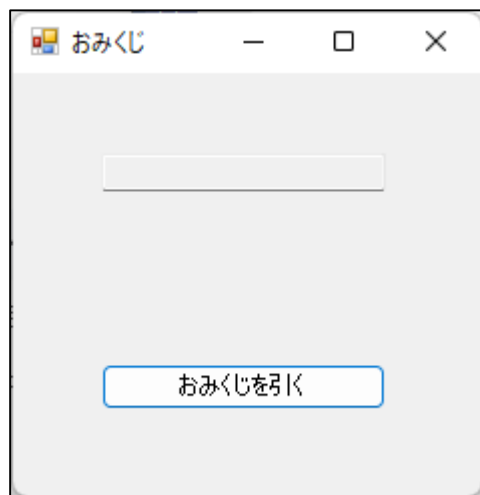
！ ポイント

テキストボックスに計算結果の値を入れる際にはToStringで整数を文字列に変換する

演習問題2

ボタンを押すとおみくじが引けるアプリを作ってみましょう

実行イメージ

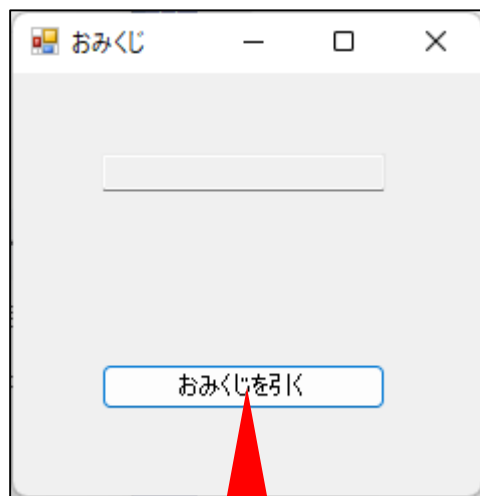


おみくじの結果（ランダム）
大吉・中吉・小吉・吉・大凶

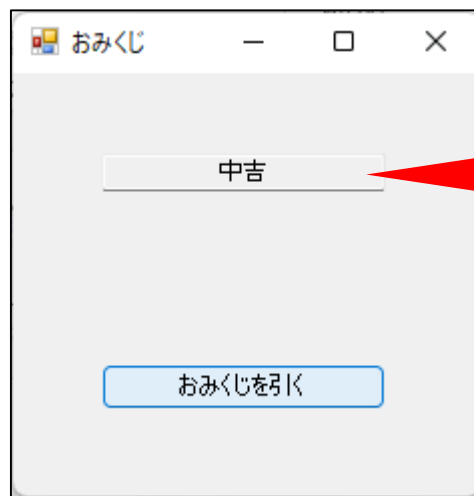
演習問題2

ボタンを押すとおみくじが引けるアプリを作ってみましょう

実行イメージ



①ボタンを押す



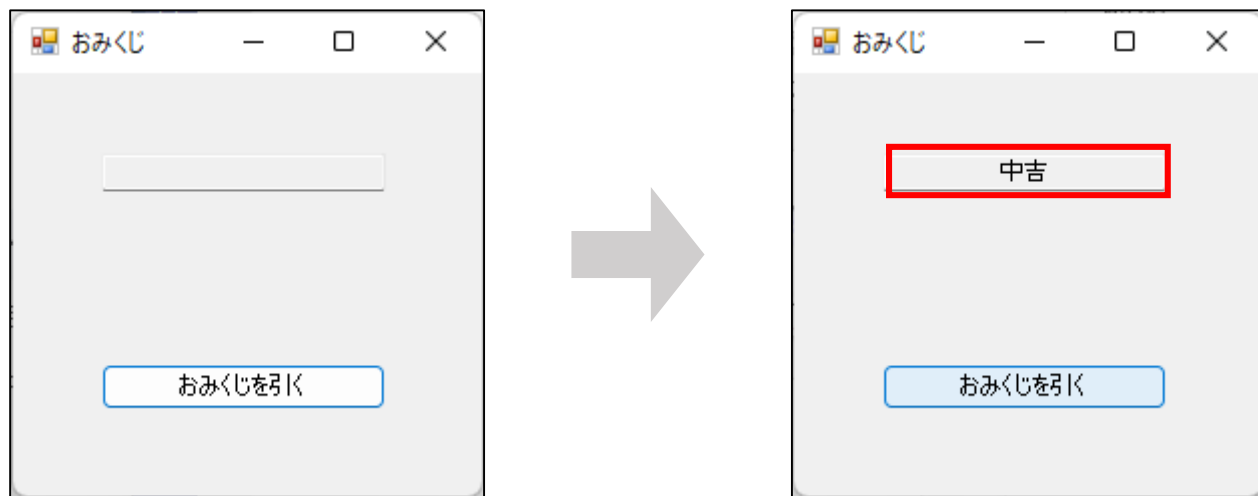
②結果の出力

おみくじの結果（ランダム）
大吉・中吉・小吉・吉・大凶

演習問題2

ボタンを押すとおみくじが引けるアプリを作ってみましょう

実行イメージ



おみくじの結果（ランダム）
大吉・中吉・小吉・吉・大凶

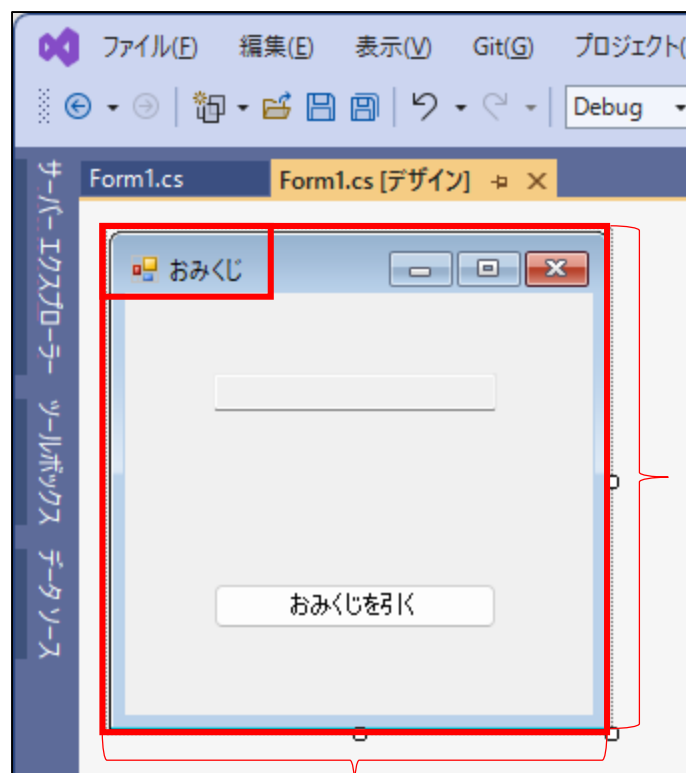
！ポイント

テキストボックスのテキストを中心に表示するには「TextAlign」を「Center」にする

演習問題2

ボタンを押すとおみくじが引けるアプリを作ってみましょう

手順① フォームの配置



250px

250px

Text: おみくじ

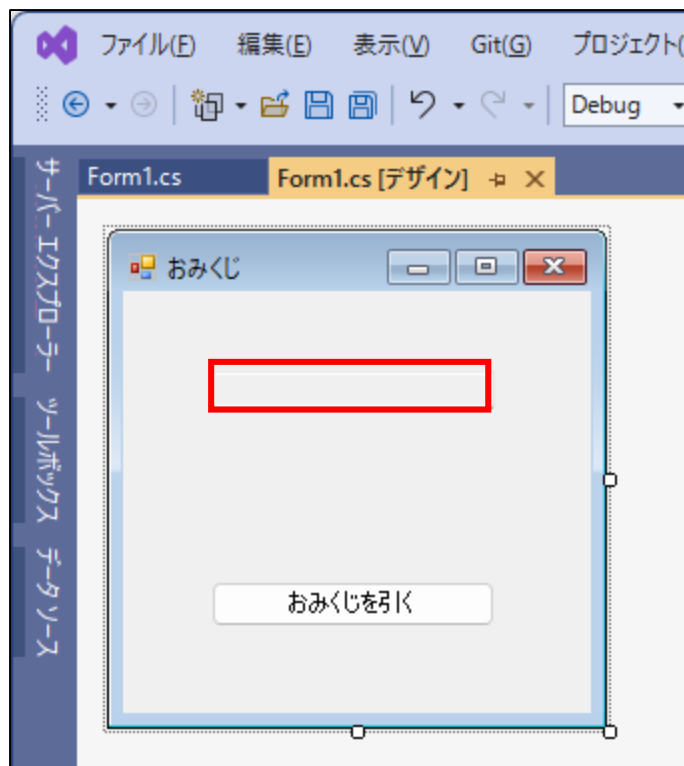
Size: 250,250

フォームを選択し入力

演習問題2

ボタンを押すとおみくじが引けるアプリを作ってみましょう

手順②テキストボックスの配置



Name: **kuji**

TextAlign: **Center**

ReadOnly: **True**

Location: **44,40**

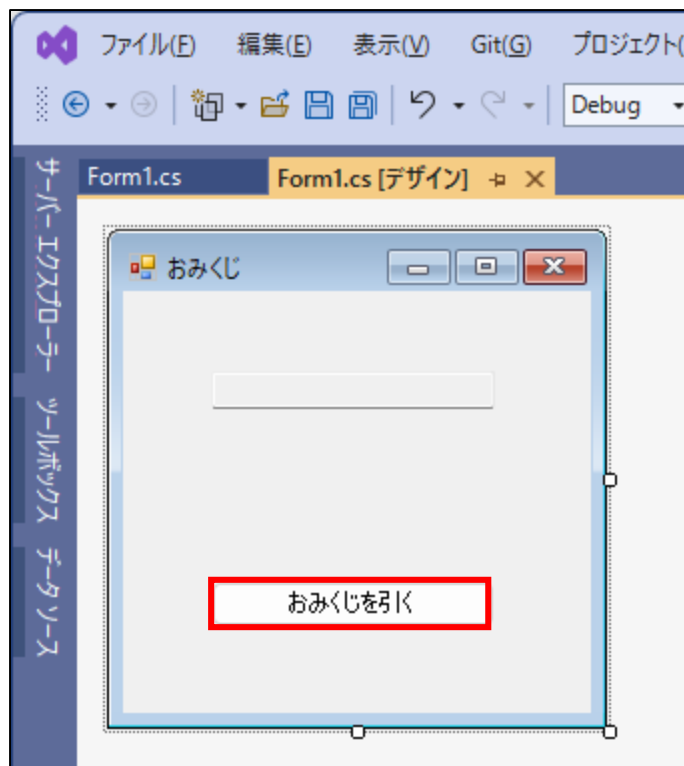
Size: **142,19**

ボタン選択し入力

演習問題2

ボタンを押すとおみくじが引けるアプリを作ってみましょう

手順③ボタンの配置



Text: **おみくじを引く**

Location: **44,145**

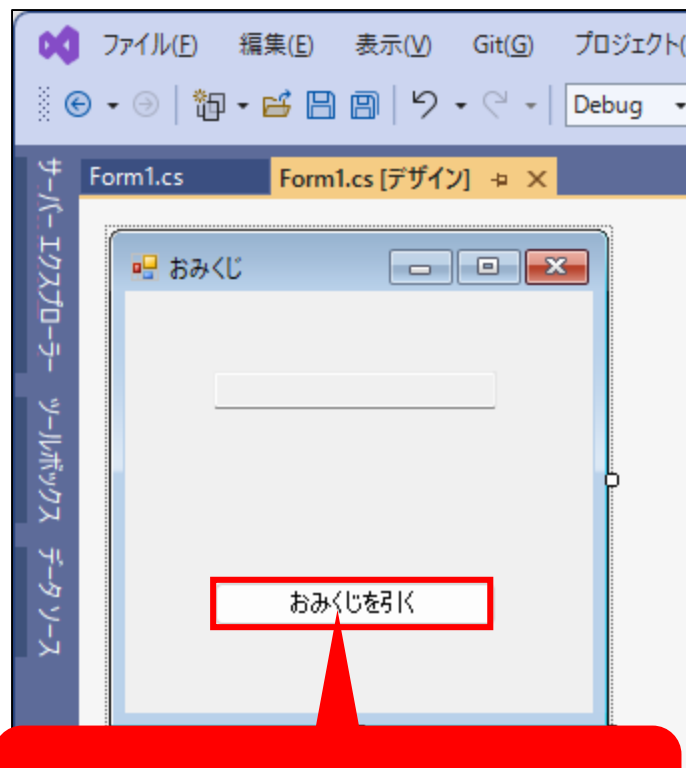
Size: **142,23**

ボタン選択し入力

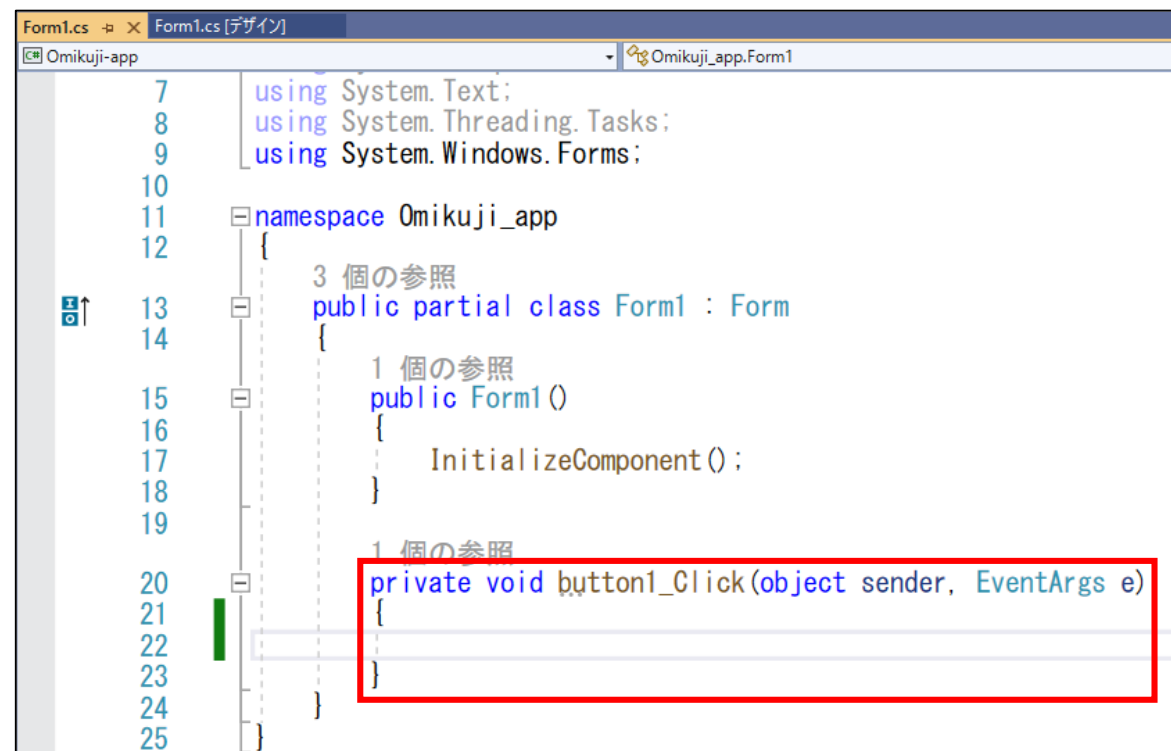
演習問題2

ボタンを押すとおみくじが引けるアプリを作ってみましょう

手順④ イベントハンドラの追加



ダブルクリック



演習問題1

手順⑤ イベント処理の追加

▼ button1_Clickメソッドに追加する処理

```
Random random = new Random();  
int num = random.Next(0, 6);  
// おみくじの結果  
string[] kekka = { "大吉", "中吉", "小吉", "吉", "凶", "大凶" };  
kuji.Text = kekka[num];
```

演習問題1

手順⑤ イベント処理の追加

▼ button1_Clickメソッドに追加する処理

```
Random random = new Random();  
int num = random.Next(0, 6);
```

乱数の発生

```
// おみくじの結果
```

```
string[] kekka = { "大吉", "中吉", "小吉", "吉", "凶", "大凶" };
```

```
kuji.Text = kekka[num];
```

演習問題1

手順⑤ イベント処理の追加

▼ button1_Clickメソッドに追加する処理

```
Random random = new Random();  
int num = random.Next(0, 6);  
// おみくじの結果
```

```
string[] kekka = { "大吉", "中吉", "小吉", "吉", "凶", "大凶" };  
kuji.Text = kekka[num];
```

くじの結果の出力