

VisualStudio で新しいプロジェクトを作成すると図 2 のように Form1 という名前で新しい画面のフォームが自動的に作成されます。

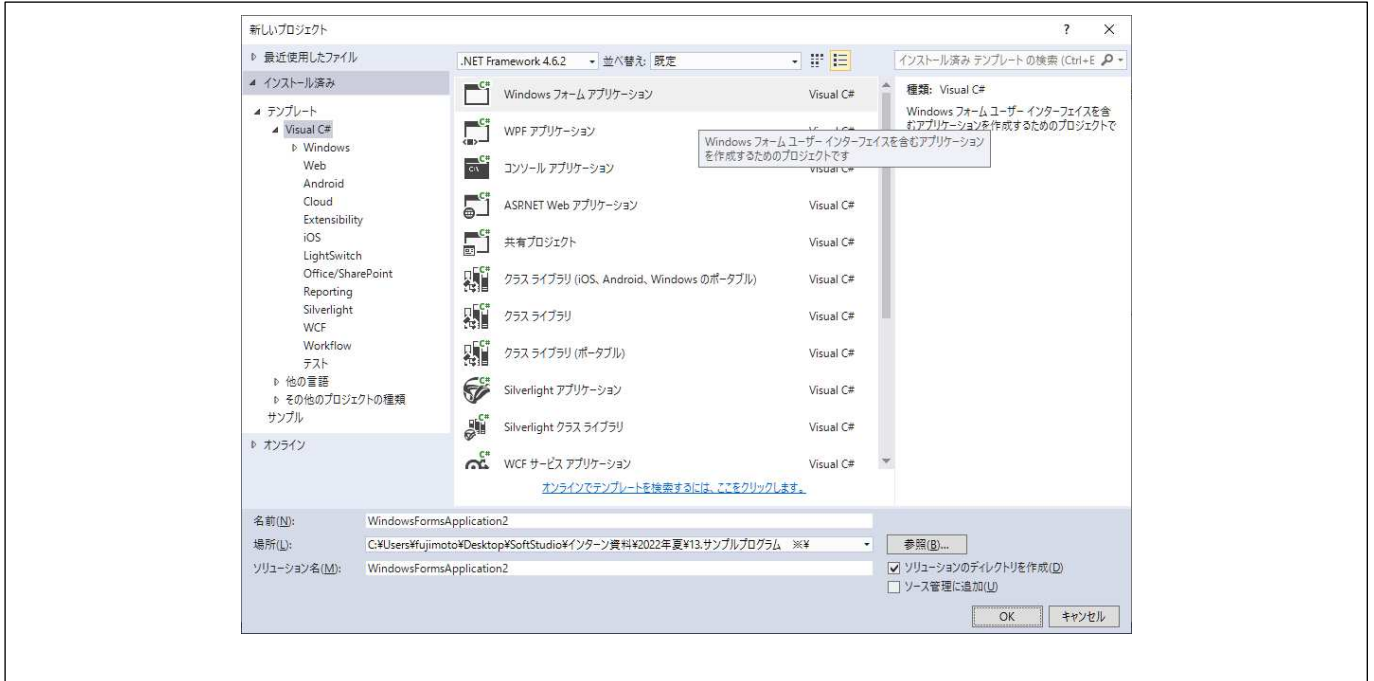


図 1 新しいプロジェクトの作成

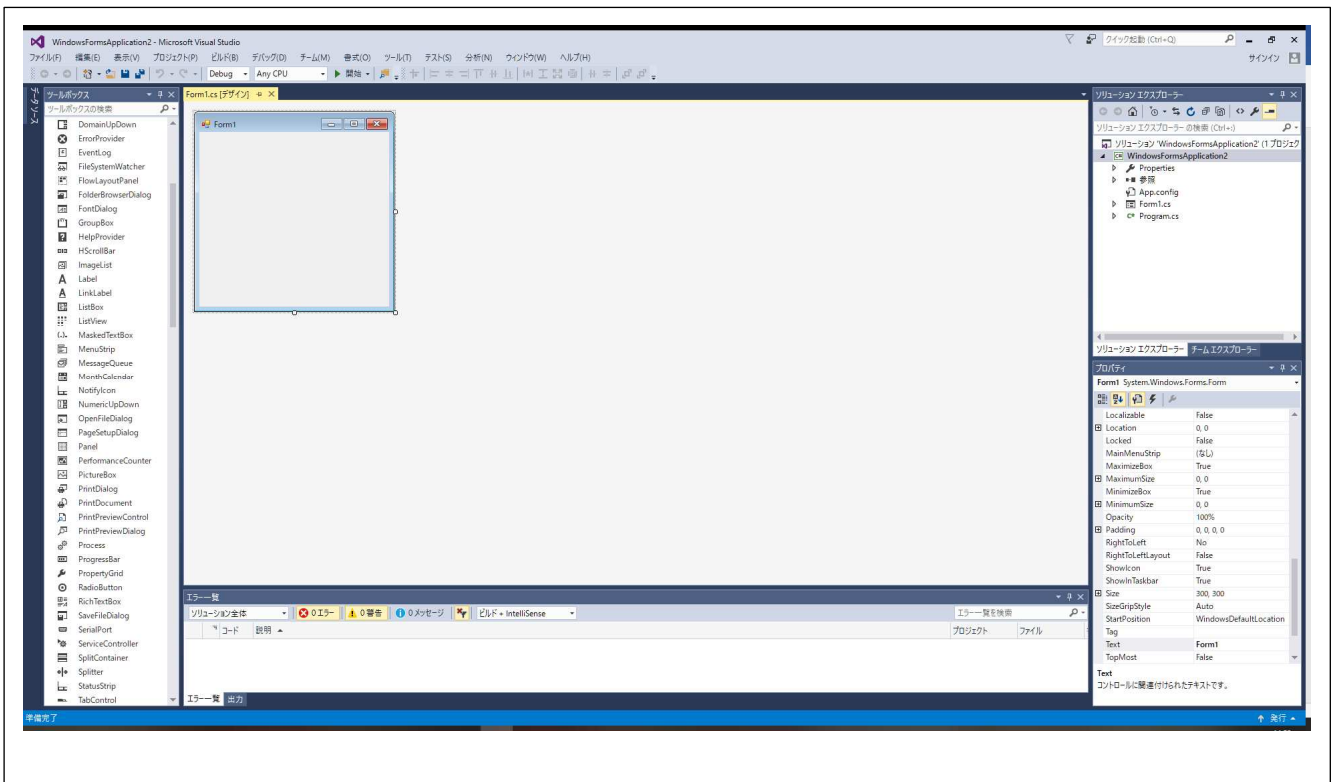


図 2 新しいフォーム画面

右側のソリューションエクスプローラーの「Form1. cs」を右クリックし、メニューの上から 3 番目「コードの表示」をクリックすると以下のような画面が表示されます。

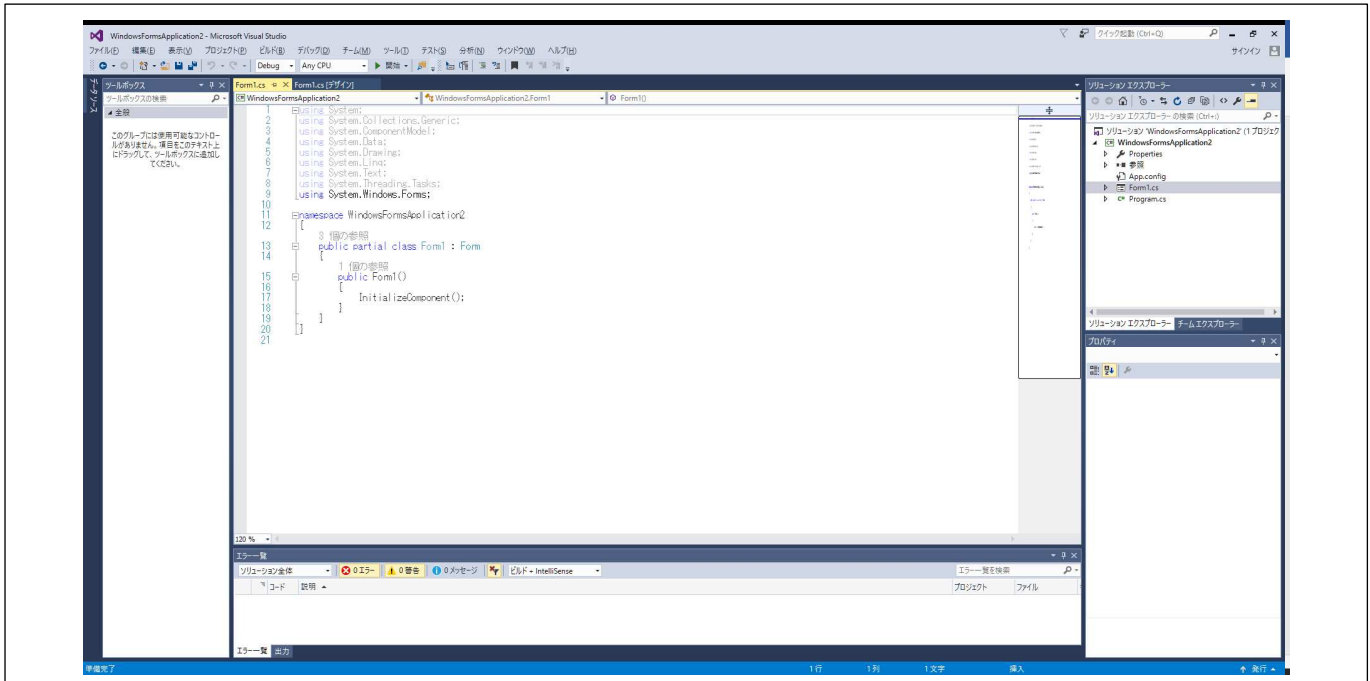


図 3 コードの表示

C#はオブジェクト指向言語といわれるもので自動的に「Form1」に対応したクラスとその初期化プログラムが自動生成されます。図 2 の画面で「Form1」画面の上部のバーをダブルクリックすると図 3 の画面に新たに以下のコーディングが追加されます。

```
private void Form1_Load(object sender, EventArgs e)
{
}

```

この Form1_Load というプログラムはこのフォームが表示されるとき一番最初に実行されるプログラムで初期化を行うときはここに記述します。

図 2 に右下の方にはプロパティが表示されておりそれぞれ名前を付けられるようになっています。このフォームの名前は「Form1」となっています。

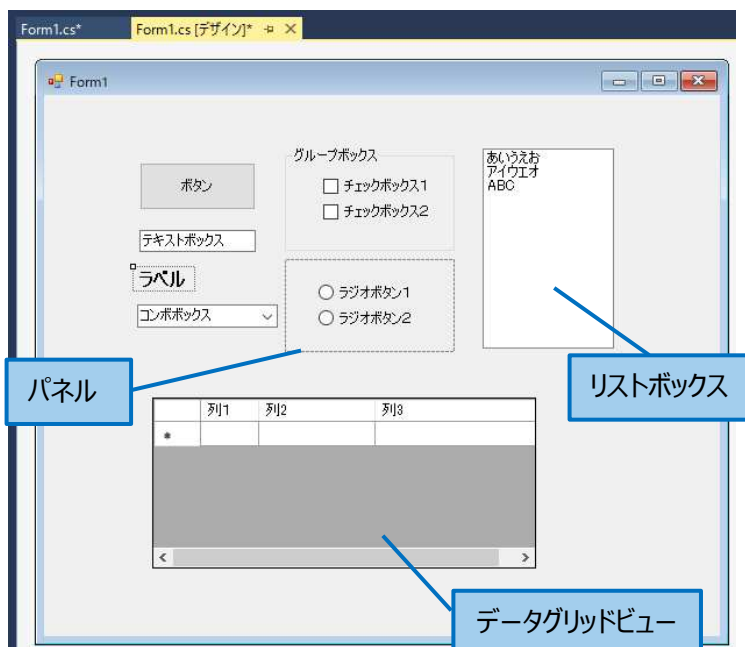
このように C#では画面や次に示すコントロールなどがオブジェクトとして存在し、それごとにプロパティ、イベントをもちプログラムを書いていくことができます。

コントロール

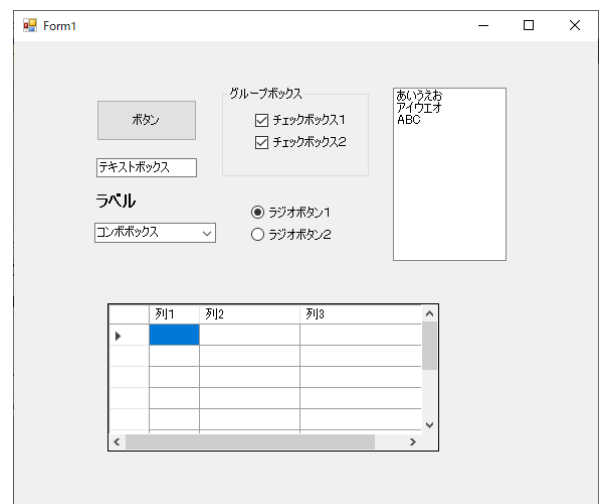
コマンドボタンやテキストボックスなど作成する画面の部品となるものの総称。

No	名称	
1	コマンドボタン	クリック時にイベントを発生する
2	テキストボックス	文字列を入力したり、表示したりする
3	ラベル	文字列を表示する
4	チェックボックス	関連オプションの選択、クリア（複数選択可）
5	ラジオボタン	複数の中から 1 項目のみ選択
6	コンボボックス	使用できる値のドロップダウンリストを表示させることのできるテキストボックス
7	データグリッドビュー	表形式でデータを表示、編集する
8	グループボックス	複数コントロールをグループ化して周囲にフレームを表示する
9	リストボックス	ユーザーが使用できる項目を一覧で表示する
10	パネル	コントロールをグループ化することができる

作成画面



実行画面



作成画面でコントロールのテキストや背景色など変更した内容を確認することができますが、コントロールによっては作成画面と実行画面で表示が変わるものもあります。また、プログラムでコントロールの制御をした場合は作成画面には反映されず、実行画面でのみ確認することができます。実行画面で確認することができます。

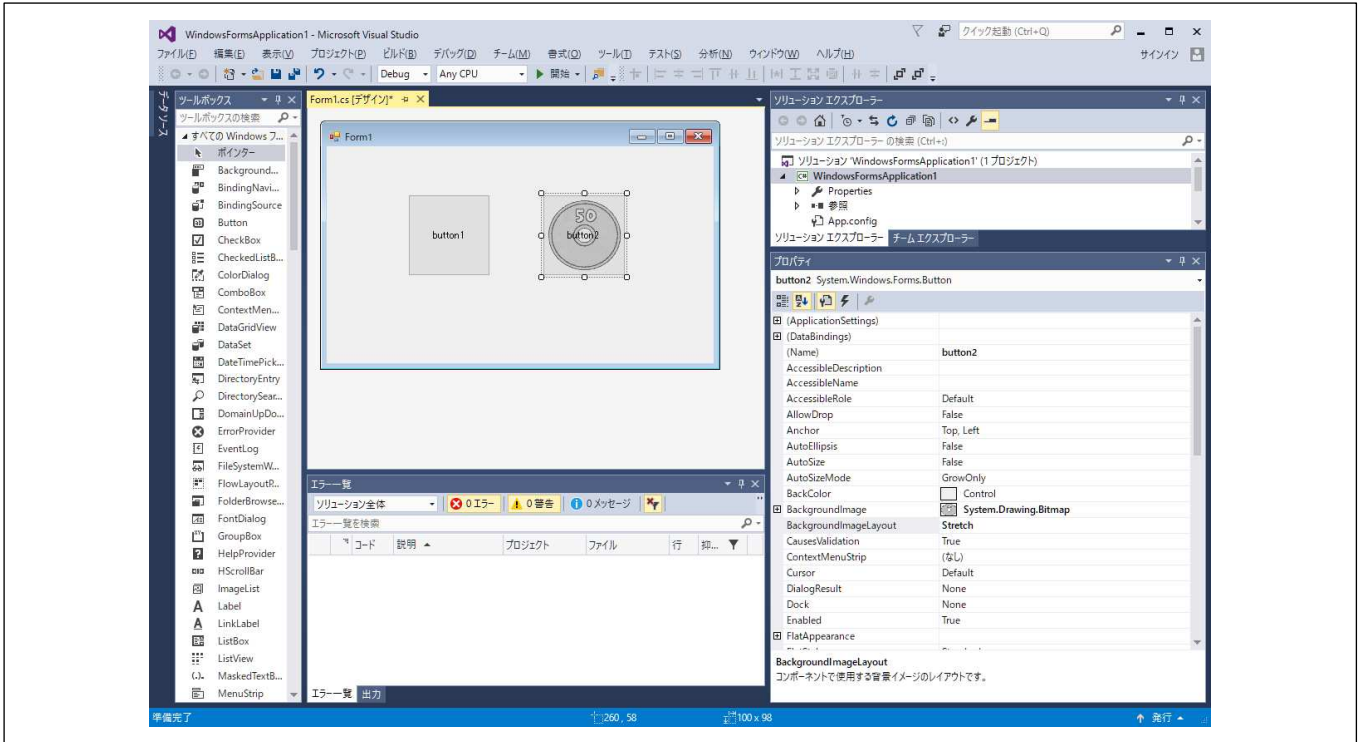
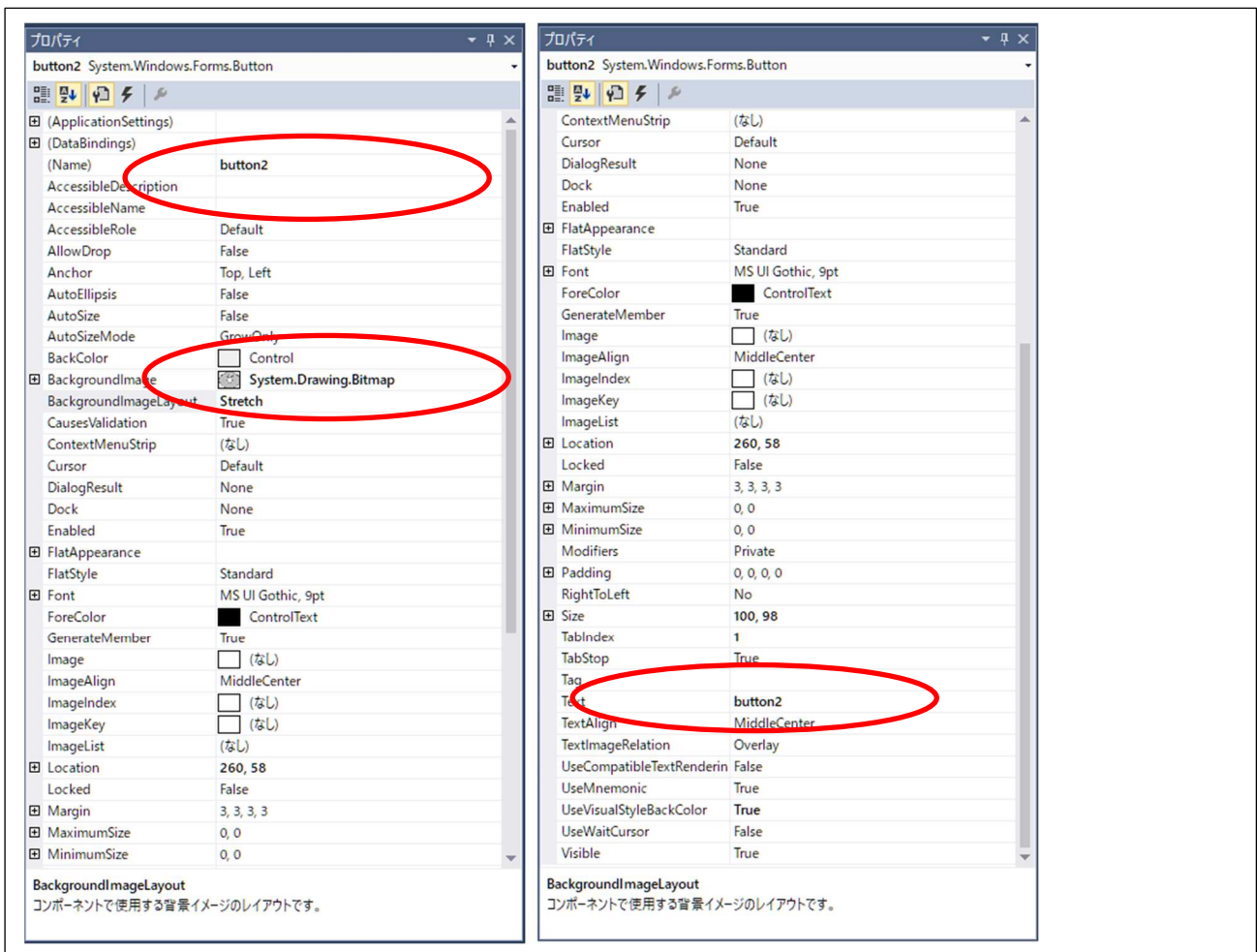


図4 プロパティ



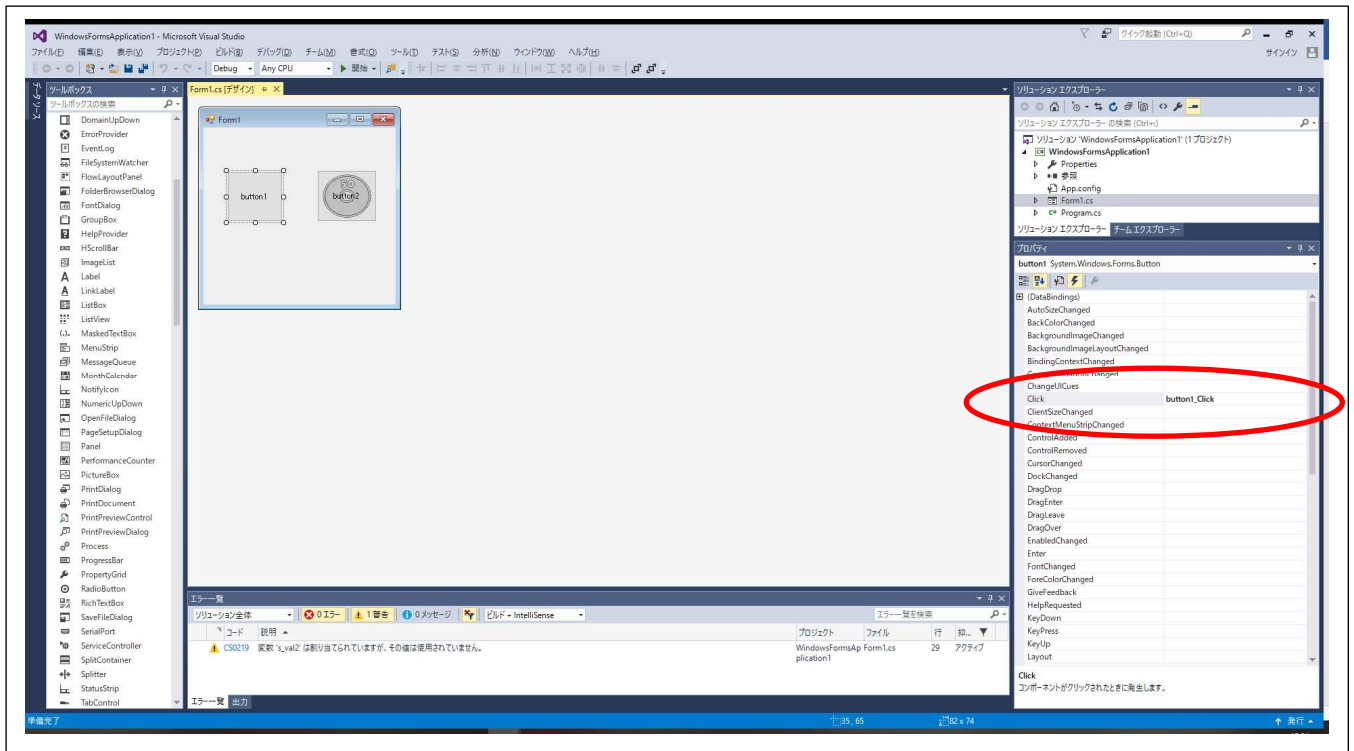



図 5 イベント

プロパティ画面で  このボタンをクリックするとこのボタンで使用できるイベントが表示されます。VisualStudio ではイベントドリブン型といってイベントといものを起点にしてプログラミングを行います。

条件に一致する場合と、一致しない場合で異なる処理を行うときは、if ステートメントを使います。

条件式は&&演算子や||演算子を使って複数記述できます。

else や else if ブロックは省略可能です。

```
if (条件式 1)
{
    条件式 1 が true のときの処理;
}
else if (条件式 2)
{
    条件式 2 が true のときの処理;
}
else
{
    どちらも false のときの処理;
}
```

例 1

```
private void buttonIF_Click(object sender, EventArgs e)
{
    int a = 0;
    label1.ForeColor = Color.Green;

    //テキストボックスに値が入力されていない場合
    if (textBox1.Text == "")
    {
        label1.Text = "入力してください";
        return;
    }
    //入力された値が数値であるかチェック
    else if (int.TryParse(textBox1.Text, out a) == false)
    {
        label1.Text = "数値で入力してください";
        return;
    }
    //入力された値が数値だった場合
    else
    {
        label1.Text = "入力した値は" + a.ToString() + "です。";
    }
}
```

条件式①が「true」の場合に処理が行われる

条件式①が「false」で、条件式②が「true」の場合に処理が行われる

条件式①、条件式②どちらも「false」の場合に処理が行われる

Point

if ブロックや else ブロックなど各ブロック内にも、if ステートメントを記述できます。ブロック内にブロックを記述した状態を**ネスト**（入れ子）といいます。

例 2

```
private void buttonNest_Click(object sender, EventArgs e)
{
    int a = 0;
    label1.ForeColor = Color.Red;

    //テキストボックスに値が入力されている場合（空白でないとき）
    if (textBox1.Text != "")
    {
        //入力された値が数値であるかチェック
        if (int.TryParse(textBox1.Text, out a))
        {
            label1.Text = "入力した値は" + a.ToString() + "です。";
        }
        else //入力された値が数値でなかった場合
        {
            label1.Text = "数値で入力してください";
            return;
        }
    }
    //テキストボックスに値が入力されていない場合
    else
    {
        label1.Text = "入力してください";
        return;
    }
}
```

演算子

足し算や引き算などの計算は、演算子を使って行います。演算子には、計算を行うもののほかに、文字列を連結したり、変数に値を代入したり、比較を行ったりするものなどがあります。

また、演算と代入を一つにした演算子を複合代入演算子といいます。

演算子	説明	例	結果
+	加算	10+20	30
-	減算	10-5	5
*	乗算	2*3	6
/	除算	10.0/4	2.5
/	除算（商が整数）	10/4	2
%	剰余	10%4	2
++	インクリメント	a++	a に 1 を加算した結果を a に代入
--	デクリメント	a--	a から 1 を減算した結果を a に代入

▼主な複合代入演算子

演算子	説明	例	結果
+=	右の値を左のオブジェクトに加算	a+=10	a に 10 を加算した結果を a に代入
-=	右の値を左のオブジェクトから減算	a-=10	a から 10 を減算した結果を a に代入
=	右の値を左のオブジェクトに乗算	a=10	a に 10 を乗算した結果を a に代入
/=	右の値で左のオブジェクトを除算	a/=10	a を 10 で除算した結果を a に代入
%=	右の値で左のオブジェクトの剰余	a%=10	a を 10 で除算した剰余を a に代入

値の大小を比較したり、論理積や論理和を求めたりする際には演算子を使用します。また、文字列を連結したり、変数などに値を代入したりするときにも演算子を使います。

▼主な連結演算子

演算子	説明	例	結果
+	文字列を連結	"C#"+"2019"	"C#2019"

▼主な代入演算子

演算子	説明	例	結果
=	右の値を左のオブジェクトに格納	a=10	aの値が10になる

▼主な比較演算子

演算子	説明	例	結果
==	等しい	15 == 30	false
!=	等しくない	15 != 30	true
>	より大きい	15 > 30	false
<	より小さい	15 < 30	true
>=	以上	15 >= 30	false
<=	以下	15 <= 30	true

▼主な論理演算子

演算子	説明	例	結果
!	論理否定	!(15 < 30)	false
&	論理積	15 < 30 & 3 > 2	true
		15 < 30 & 3 < 2	false
	論理和	15 < 30 3 < 2	true
		15 > 30 3 < 2	false
^	排他的論理和	15 < 30 ^ 3 < 2	true
		15 < 30 ^ 3 > 2	false
		15 > 30 ^ 3 < 2	false

複数の条件を判断する

&演算子、または|演算子を使って複数の式を評価する場合、代わりに&&演算子または||演算子を使うこともできます。

▼ &&演算子

&演算子(論理積)のように、演算子の両側の式がともに「true」の場合のみ、「true」を返します。

ただし、&演算子と違って、演算子の左側の式が「false」の場合は、右側の式を評価せずに「false」を返します。したがって、パフォーマンスは向上しますが、右側の式の実行が必要な場合は使えません。

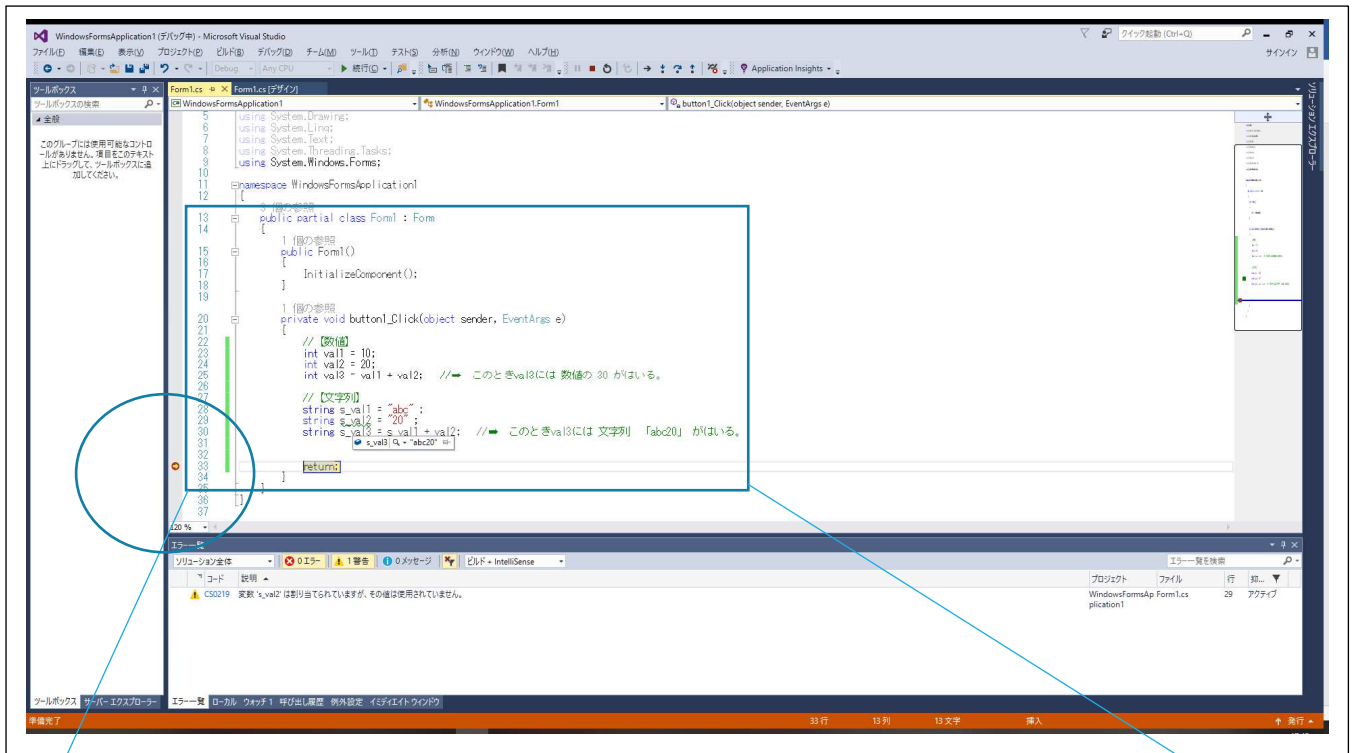
```
15<30 && 3>2
```

▼ ||演算子

|演算子(論理和)のように、演算子の両側の式のどちらかが「true」の場合に、「true」を返します。

ただし、|演算子と違い、演算子の左側の式が「true」の場合は、右側の式を評価せずに「true」を返します。したがって、パフォーマンスは向上しますが、右側の式の実行が必要な場合は使えません。

```
15<30 || 3<2
```



```

11 namespace WindowsFormsApplication1
12 {
13     3 個 の 参 照
14     public partial class Form1 : Form
15     {
16         1 個 の 参 照
17         public Form1()
18         {
19             InitializeComponent();
20         }
21
22         1 個 の 参 照
23         private void button1_Click(object sender, EventArgs e)
24         {
25             // 【数値】
26             int val1 = 10;
27             int val2 = 20;
28             int val3 = val1 + val2; //⇒ このときval3には数値の 30 が入る。
29
30             // 【文字列】
31             string s_val1 = "abc";
32             string s_val2 = "20";
33             string s_val3 = s_val1 + val2; //⇒ このときval3には文字列「abc20」が入る。
34             s_val3.Q = "abc20"
35         }
36     }
37 }
    
```

コメントの活用方法

1. コメントを追加することで他の人が見てもどのような処理が行われているのかがわかりやすくなります。
2. コード変更時などに修正箇所が分かりやすいようにコメントを残します。
3. コメント化された部分はコンパイルされないでコードをコメント化して一時的に実行しない場合にも使用します。

```
//下の行はコメント化されているので実行されない  
// i += 100;
```

コメントの入力方法

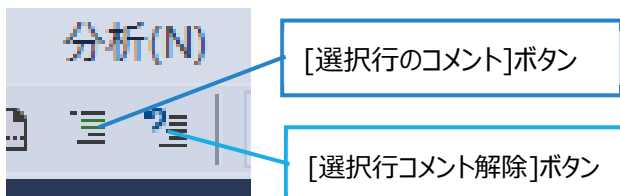
1. 「/」を2つ入力する

「/」を2つ続けて入力後にコメントを入力します。
「//」からその行の末尾までがコメント化されます。

```
int i;  
// コメント  
i = 100 * 2; //コメント
```

2. ツールバーの[選択行のコメント]ボタンを押す

コメント化を解除するときは[選択行コメント解除]
ボタンを押します。



コメントの種類

1. 1行のみのコメント
2. ブロックコメント

コメント開始位置に「/*」を入力し、コメント終了位置に「*/」を入力すると連続した複数の行をコメントにすることができます。また、全ての行頭に「//」を記述することでも可能です。

```
/*  
*  
* ブロックコメント  
*  
*/
```

```
//  
//  
// ブロックコメント  
//  
//
```

3. サマリー

各メソッドのひとつ前の行で「/」を3つ入力するとサマリーを作成することができます。

```
/// <summary>  
/// ボタンクリック時の処理  
/// </summary>  
/// <param name="sender"></param>  
/// <param name="e"></param>  
private void button1_Click(object sender, EventArgs e)  
{
```

ブレークポイント

処理の途中で一時的に停止したいときに使用します。

ブレークポイントを使うと、その後の処理を1命令ずつ進めることができ、エラーの発生元を調べたり、変数にどのような値が入っているのかを確認したりすることができます。

