

2019 年 1 月 10 日 (木) 実施

構造体

レコードと構造体

複数の項目に渡るデータを一まとめにしたものを**レコード**という。例えば、次の様に学籍番号、氏名、履修科目コード、点数、評価といった項目による 1 人分のデータを一まとめにしたものは 1 件分のレコードである。

学籍番号	氏名	履修科目コード	点数	評価
18x0123	上田奈緒子	z1021234	100	S

C#言語では、このようなレコードのデータ構造を**構造体** (structure) によって表現する。構造体を用いれば、個々の項目を別々の変数で表す場合に比べてデータ間の関係が把握しやすく、プログラムの可読性が増す。また、C#言語のメソッドでは通常の変数を return 文に書いた場合、1 つのデータしか戻せないが、構造体を return 文に記述することが可能であるので、複数のデータを一まとめにして戻すことができる。なお、構造体は参照型ではなく、値型である。

構造体の利用

C#言語で構造体を用いる際の一連の流れは次のようになる。

- 1) レコード設計を行い、**構造体の定義**を行う。
- 2) **構造体変数の宣言**を行う。
- 3) 構造体変数の**メンバ**への代入、メンバの参照。 (レコードの各項目をメンバと呼ぶ)

上述の学生データを例にとって、この一連の流れを表すと、次のようになる。

- 1) 構造体の定義

```
struct Student
{
    public String id;
    public String name;
    public String class;
    public int point;
    public char eval;
}
```
- 2) 構造体変数の宣言

```
Student st;
```
- 3) メンバへの代入

```
st.id = "18x0123";
st.name = "上田奈緒子";
st.class = "z1021234";
st.point = 100;
st.eval = 'S';
```

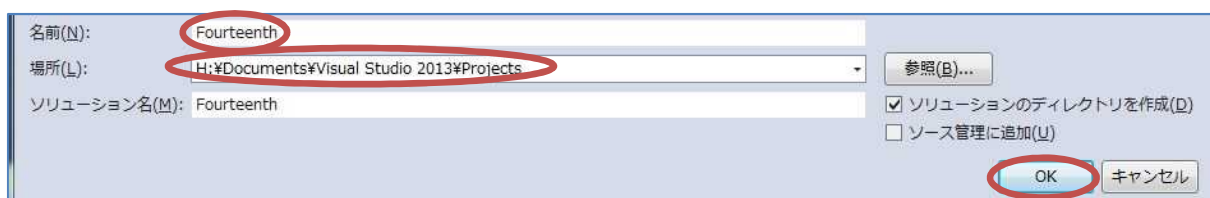
本日の課題

構造体の定義，構造体変数の扱い，及び引数や戻り値が構造体変数である様なメソッドについて，実例を通じて学ぶ。

手順

1) プロジェクトの作成

Visual Studio 2013 を起動したら，[ファイル] → [新規作成] → [プロジェクト] と辿って，プロジェクトを作成する。『新しいプロジェクト』ダイアログボックスでは，プログラミング言語を『Visual C#』，プロジェクトテンプレートとしては，『Windows フォームアプリケーション』を選択し，『名前』を「**Fourteenth**」に書き換え，『場所』が「H:\Documents\Visual Studio 2013\Projects」となっていることを確認してから『OK』を押す（詳細は第 1 回の教材を参照）。



2) コントロールの配置及びフォームの作成

今後，フォーム上に配置するコントロールのプロパティのフォントサイズは全て 14 ポイントに変更するものとする。

Form1 上にラベルを 3 つ，テキストボックスを 3 つ，ボタンを 3 つ貼り付ける。それぞれのプロパティは次の様に設定する。

【Form1】 Text 「構造体」

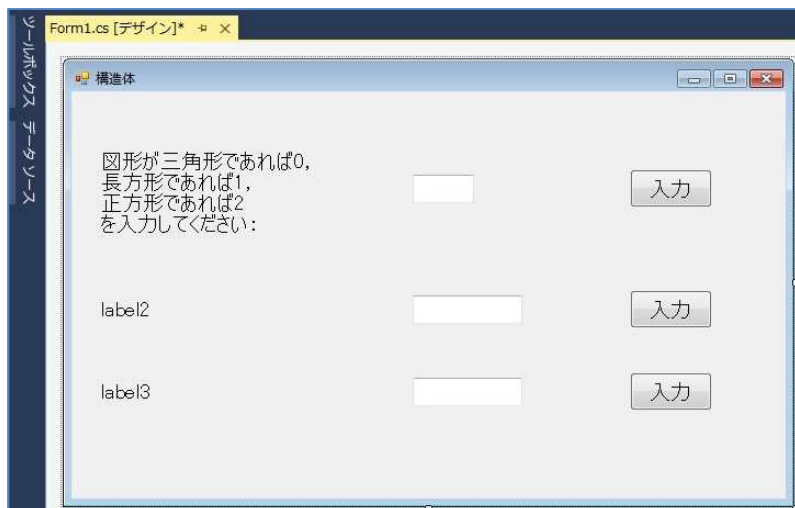
【label1】 Text

「図形が三角形であれば 0，
長方形であれば 1，
正方形であれば 2
を入力してください：」

【button1】 Text 「入力」

【button2】 Text 「入力」

【button3】 Text 「入力」



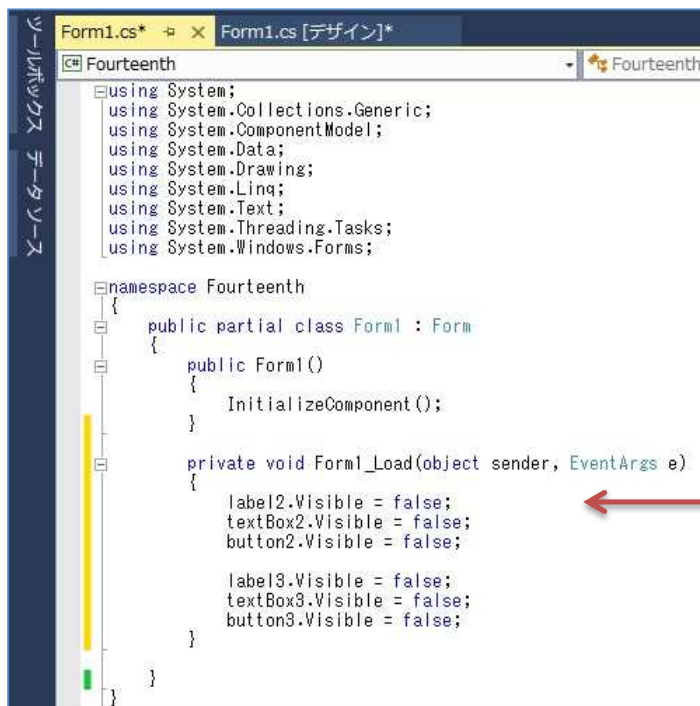
3) コーディング

Form1 のフォームデザイナー上でコントロールが貼られていない箇所をダブルクリックして Form1.cs のプログラムのソースコードを表示する。Form1_Load メソッドのブロック内に Form1 が読み込まれた際の処理として，『label2』，『textBox2』，『button2』，『label3』，『textBox3』及び『button3』の『Visible』プロパティに「false」を設定して非表示にする処理（赤枠の部分）

を記述する。

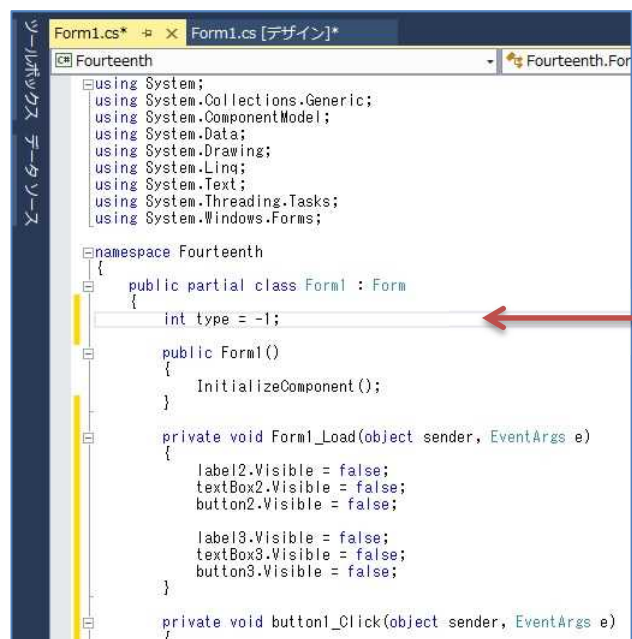
```
private void Form1_Load(object sender, EventArgs e)
{
    label2.Visible = false;
    textBox2.Visible = false;
    button2.Visible = false;

    label3.Visible = false;
    textBox3.Visible = false;
    button3.Visible = false;
}
```



Form1 のフォームデザイナー上で『button1』をダブルクリックして、Form1.cs のプログラムのソースコードを表示する。先ず、クラス全体に適用可能な変数の宣言をクラスの冒頭に記述する。

```
int type = -1;
```



次に, `button1_Click` メソッドのブロック内にボタンがクリックされた際の処理 (赤枠の部分) を記述していく。

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        type = Int32.Parse(textBox1.Text);
        String s1 = "", s2 = "";

        if (type < 0 || type > 2)
        {
            MessageBox.Show("入力する数値は¥n 0, 1, 2 のいずれかです。", "範囲外");
        }
        else if (type == 0)
        {
            s1 = "三角形の底辺の長さ(実数値)¥n を入力してください: ";
            s2 = "三角形の高さ(実数値)¥n を入力してください: ";
        }
        else if (type == 1)
        {
            s1 = "長方形の一边 X の長さ(実数値)¥n を入力してください: ";
            s2 = "長方形の一边 Y の長さ(実数値)¥n を入力してください: ";
        }
        else
        {
            s1 = "正方形の一边の長さ(実数値)¥n を入力してください: ";
        }

        label2.Text = s1;
        label3.Text = s2;

        if (type >= 0 && type <= 2)
        {
            label2.Visible = true;
            textBox2.Visible = true;
            button2.Visible = true;
        }
    }
    catch (FormatException fex)
    {
        MessageBox.Show(fex.Message, "エラー");
    }
    finally
    {
        textBox1.Text = "";
    }
}
```

```

private void button1_Click(object sender, EventArgs e)
{
    try
    {
        type = Int32.Parse(textBox1.Text);
        String s1 = "", s2 = "";

        if (type < 0 || type > 2)
        {
            MessageBox.Show("入力する数値は0, 1, 2のいずれかです。", "範囲外");
        }
        else if (type == 0)
        {
            s1 = "三角形の底辺の長さ(実数値)%nを入力してください。";
            s2 = "三角形の高さ(実数値)%nを入力してください。";
        }
        else if (type == 1)
        {
            s1 = "長方形の辺Xの長さ(実数値)%nを入力してください。";
            s2 = "長方形の辺Yの長さ(実数値)%nを入力してください。";
        }
        else
        {
            s1 = "正方形の辺の長さ(実数値)%nを入力してください。";
        }

        label2.Text = s1;
        label3.Text = s2;

        if (type >= 0 && type <= 2)
        {
            label2.Visible = true;
            textBox2.Visible = true;
            button2.Visible = true;
        }
    }
    catch (FormatException fex)
    {
        MessageBox.Show(fex.Message, "エラー");
    }
    finally
    {
        textBox1.Text = "";
    }
}
    
```

Form1 のフォームデザイナー上で『button2』及び『button3』をダブルクリックして、Form1.cs のプログラムのソースコードを表示する。先ず、クラスの冒頭に構造体の定義及び構造体変数の宣言を行う。

```

struct Trirect // triangle/rectangle (三角形/四角形)
{
    public String name; // name (名称)
    public double s; // side (辺)
    public double h; // height (高さ)
    public double a; // area (面積)
}

private Trirect figure;
    
```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Fourteenth
{
    public partial class Form1 : Form
    {
        int type = -1;

        struct Trirect // triangle/rectangle (三角形/四角形)
        {
            public String name; // name (名称)
            public double s; // side (辺)
            public double h; // height (高さ)
            public double a; // area (面積)
        }

        private Trirect figure;

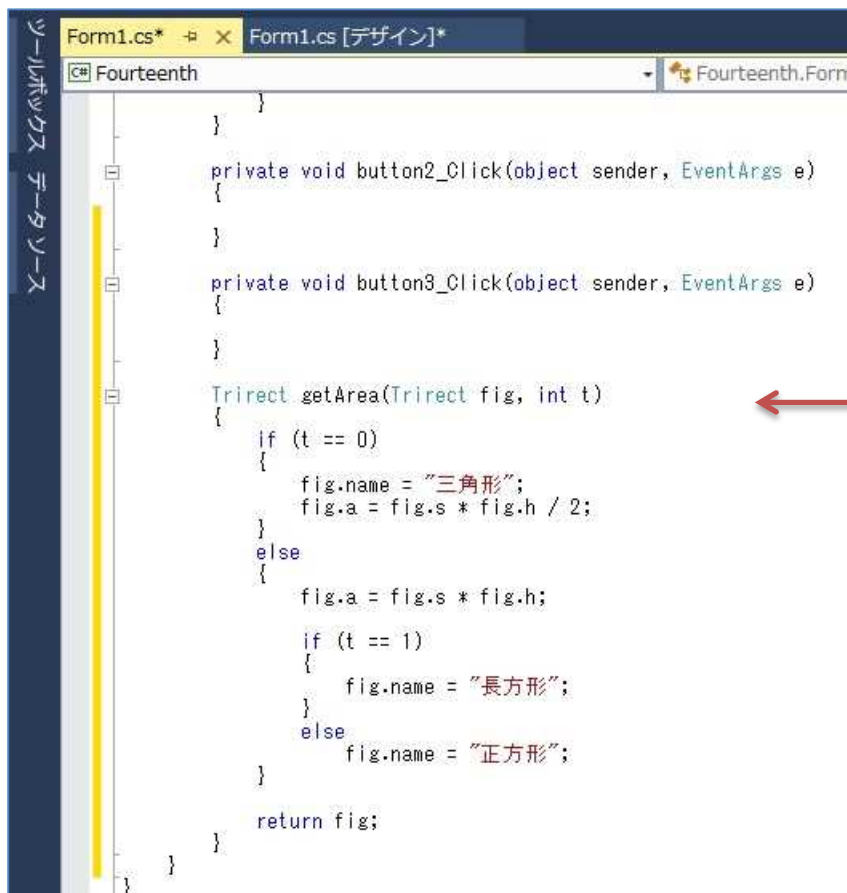
        public Form1()
    }
}
    
```

続けて、`getArea` メソッドをクラスの終わりの方に記述する。

```
Trirect getArea(Trirect fig, int t)
{
    if (t == 0)
    {
        fig.name = "三角形";
        fig.a = fig.s * fig.h / 2;
    }
    else
    {
        fig.a = fig.s * fig.h;

        if (t == 1)
        {
            fig.name = "長方形";
        }
        else
            fig.name = "正方形";
    }

    return fig;
}
```



The screenshot shows a Visual Studio window with the following tabs: `Form1.cs*`, `Form1.cs [デザイン]*`, and `C# Fourteenth`. The `Fourteenth` window is active and displays the following code:

```
    }
}

private void button2_Click(object sender, EventArgs e)
{
}

private void button3_Click(object sender, EventArgs e)
{
}

Trirect getArea(Trirect fig, int t)
{
    if (t == 0)
    {
        fig.name = "三角形";
        fig.a = fig.s * fig.h / 2;
    }
    else
    {
        fig.a = fig.s * fig.h;

        if (t == 1)
        {
            fig.name = "長方形";
        }
        else
            fig.name = "正方形";
    }

    return fig;
}
```

A red arrow points to the `Trirect getArea(Trirect fig, int t)` line.

次に、`button2_Click` メソッドのブロック内にボタンがクリックされた際の処理 (赤枠の部分) を記述していく。

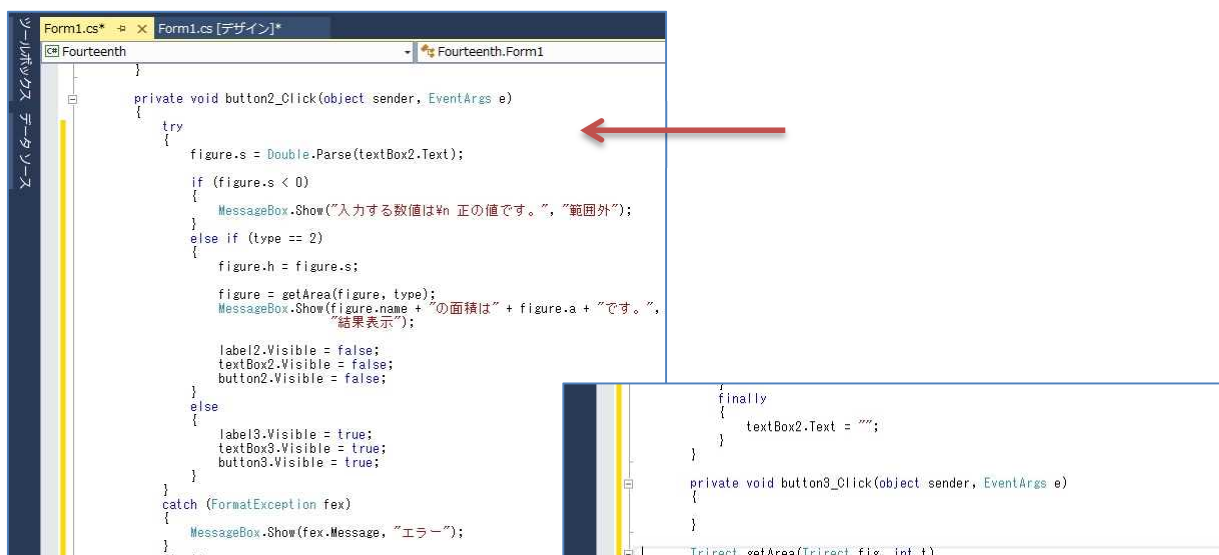
```

private void button2_Click(object sender, EventArgs e)
{
    try
    {
        figure.s = Double.Parse(textBox2.Text);

        if (figure.s < 0)
        {
            MessageBox.Show("入力する数値は#n 正の値です。", "範囲外");
        }
        else if (type == 2)
        {
            figure.h = figure.s;

            figure = getArea(figure, type);
            MessageBox.Show(figure.name + "の面積は" + figure.a + "です。",
                "結果表示");

            label2.Visible = false;
            textBox2.Visible = false;
            button2.Visible = false;
        }
        else
        {
            label3.Visible = true;
            textBox3.Visible = true;
            button3.Visible = true;
        }
    }
    catch (FormatException fex)
    {
        MessageBox.Show(fex.Message, "エラー");
    }
    finally
    {
        textBox2.Text = "";
    }
}
    
```



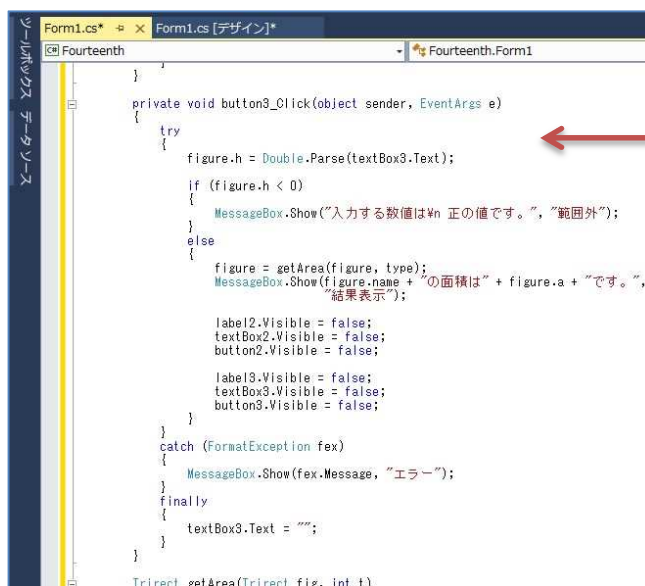
最後に、button3_Click メソッドのブロック内にボタンがクリックされた際の処理（赤枠の部分）を記述していく。

```
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        figure.h = Double.Parse(textBox3.Text);

        if (figure.h < 0)
        {
            MessageBox.Show("入力する数値は¥n 正の値です。", "範囲外");
        }
        else
        {
            figure = getArea(figure, type);
            MessageBox.Show(figure.name + "の面積は" + figure.a + "です。",
                "結果表示");

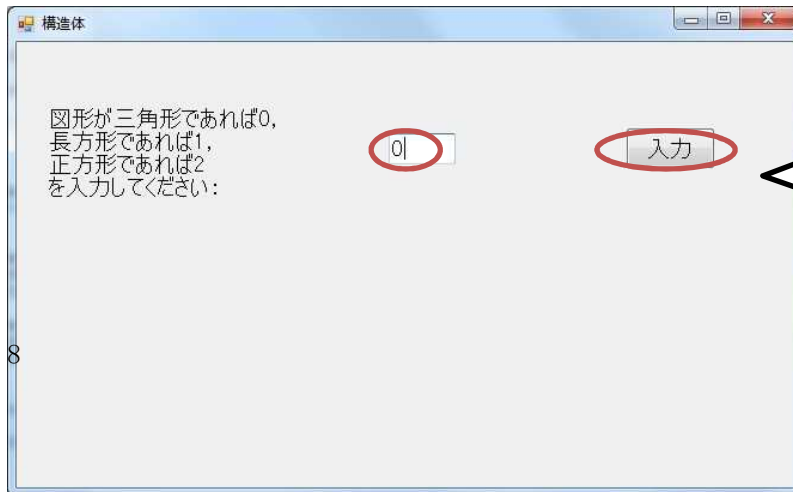
            label2.Visible = false;
            textBox2.Visible = false;
            button2.Visible = false;

            label3.Visible = false;
            textBox3.Visible = false;
            button3.Visible = false;
        }
    }
    catch (FormatException fex)
    {
        MessageBox.Show(fex.Message, "エラー");
    }
    finally
    {
        textBox3.Text = "";
    }
}
```

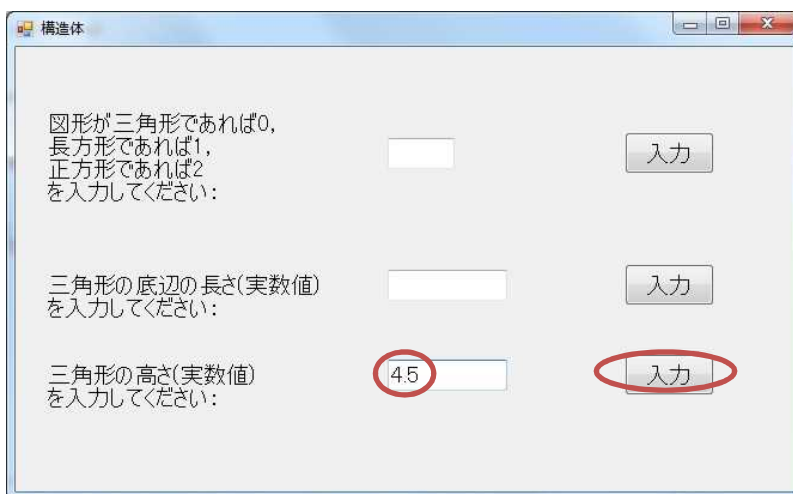
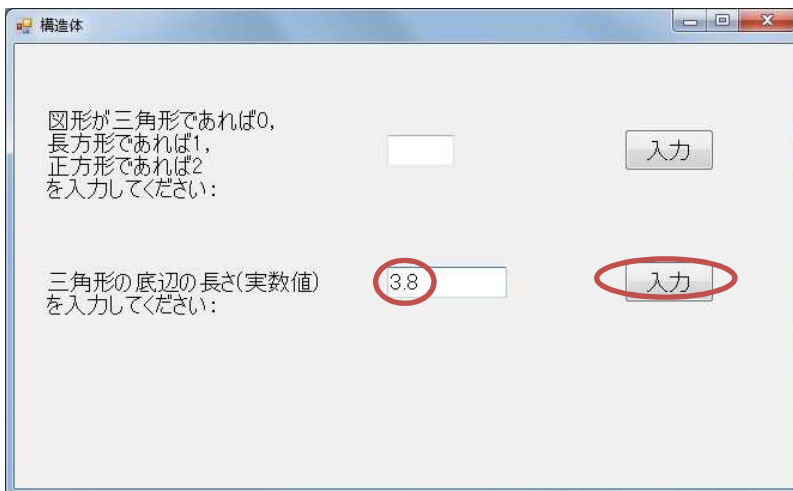


4) プログラムの実行・最終確認

『すべてを保存』ボタンを押してから、『開始』ボタンを押して、プログラムを実行する。
エラーが出ている場合には、修正してから保存、開始と進む。



三角形の面積を
求める場合の例



* 長方形や正方形の面積についても正しく求められるか確認する。

確認を終えたら、プログラムを終了する。

【ファイルが保存されている場所】 H:¥Documents¥Visual Studio 2013¥Projects¥Fourteenth¥Fourteenth

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) フォームを含むソースファイル **Form1.cs** をメールに添付して提出する。
- 3) 質問を記述したファイル **Questions_14th.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。