2018 年 12 月 20 日（木）実施

例外処理

C#言語では，作成したプログラムを実行する際に，記述した処理が想定しない事態によって実行できなくなる場合を**例外**と呼び，その例外への対処，即ち**例外処理**が求められる。

例外処理を行うための try 文の一般形は次のようになる。

```
try
{
   例外を発生させる可能性のある処理
}
catch(例外のクラス名 1 変数 1 )
{
        例外に対処する処理 1
}
catch(例外のクラス名 2 変数 2 )
{
        例外に対処する処理 2
}
            ・・・
finally
{
        try 文の最後に必ず実行される処理
}
```

ここで，catch ブロックと finally ブロックとは何れかを必ず記述する。catch ブロックは必要に応じて，例外を場合分けして複数記述することが出来る。finally ブロックには例外発生時に中断した処理の後処理等を記述する。なお，例外クラスの最も基本的なクラスは System.Exception なので，catch ブロックの引数に Exception を記述した場合には，全ての例外クラスに対応することが出来る。

ファイル操作

ファイル(File)とは，データの集合体のことで，JIS(日本工業規格)では，ファイルはレコードの集合体，レコードはデータの集合体と定義されている。ファイル操作は，次の順序で行う。なお，**ストリーム**とは，入力元または出力先を持つ，順序付けられたデータ列である。

1) ストリームを開く
2) ストリームからの入力，ストリームへの出力
3) ストリームを閉じる

C#言語では，システムライブラリの System.IO 名前空間にファイル操作に関係するクラスが複数用意されている。今回は，テキストファイルから文字を読み込むためのクラス StreamReader を用いる。

ファイル操作を行う場合には，ファイルが開けない例外，開いたファイルからデータが読み込めない例外といった，様々な例外があり得るので，例外処理は不可欠である。そこで，テキストファイルから文字を読み込む場合には，次の様なコードが必要となる（using System.IO; が前提）。

```
StreamReader sr = null;

try
{
    sr = new StreamReader("パス名¥¥ファイル名")
    文字データの読み込み処理
    読み込んだデータを用いた処理
}
catch (Exception ex)
{
    例外を知らせるメッセージの表示
}
finally
{
    if (sr != null)
    {
        sr.Close;
    }
}
```

これに対して，using ステートメント（using ディレクティブとは別物）を用いると，オブジェクトの終了処理を行う Dispose メソッドを呼び出す。using ブロックで例外が発生した場合でも必ず Dispose メソッドが呼び出されることから，上のコードの様な finally ブロックは不要となり，次の様に書き換えることが出来る。

```
try
{
    using (StreamReader sr = new StreamReader("パス名¥¥ファイル名"))
    {
        文字データの読み込み処理
        読み込んだデータを用いた処理
    }
}
catch (Exception ex)
{
    例外を知らせるメッセージの表示
}
```

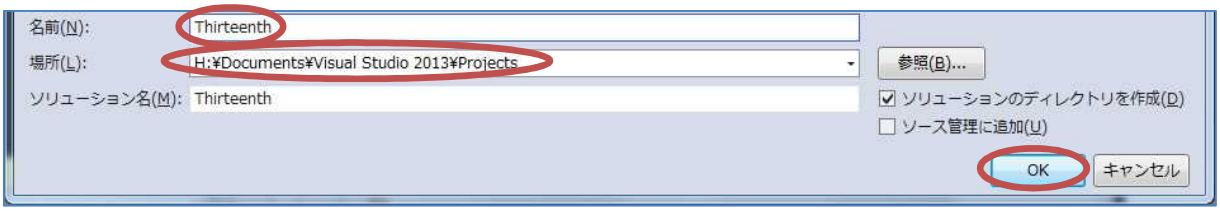
本日の課題

例外処理について，整数の入力及びファイル操作を通じて学ぶ。

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「Thirteenth」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。

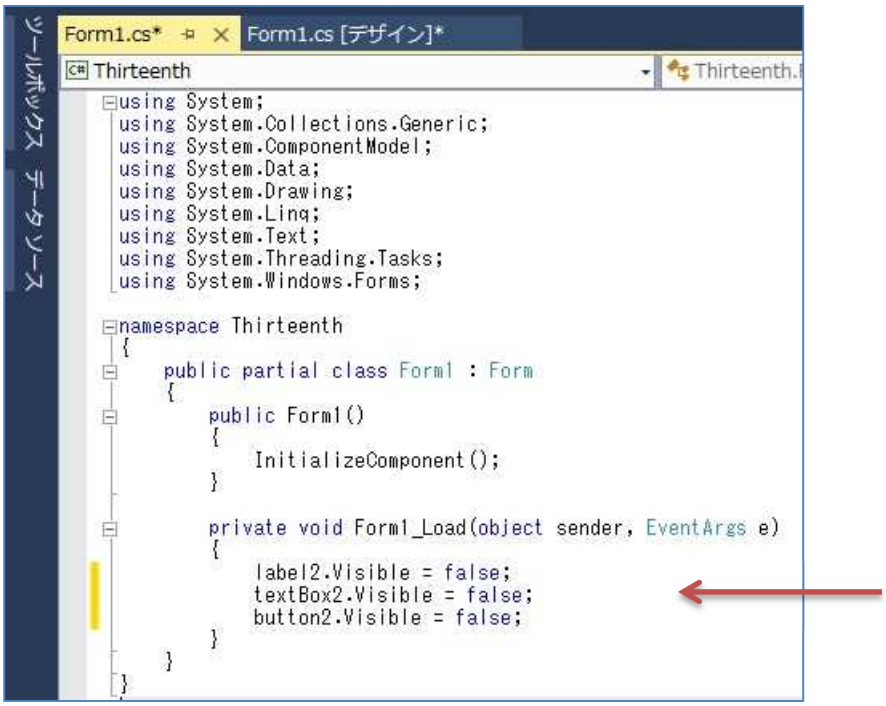
Form1 上にラベルを 2 つ，テキストボックスを 2 つ，ボタンを 2 つ貼り付ける。それぞれのプロパティは次の様に設定する。

【Form1】Text　「例外処理」
　　Icon　自作のアイコン
【label1】Text　「整数：」
【label2】Text　「ファイル名：」
【button1】Text　「入力」
【button2】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』及び『button2』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。

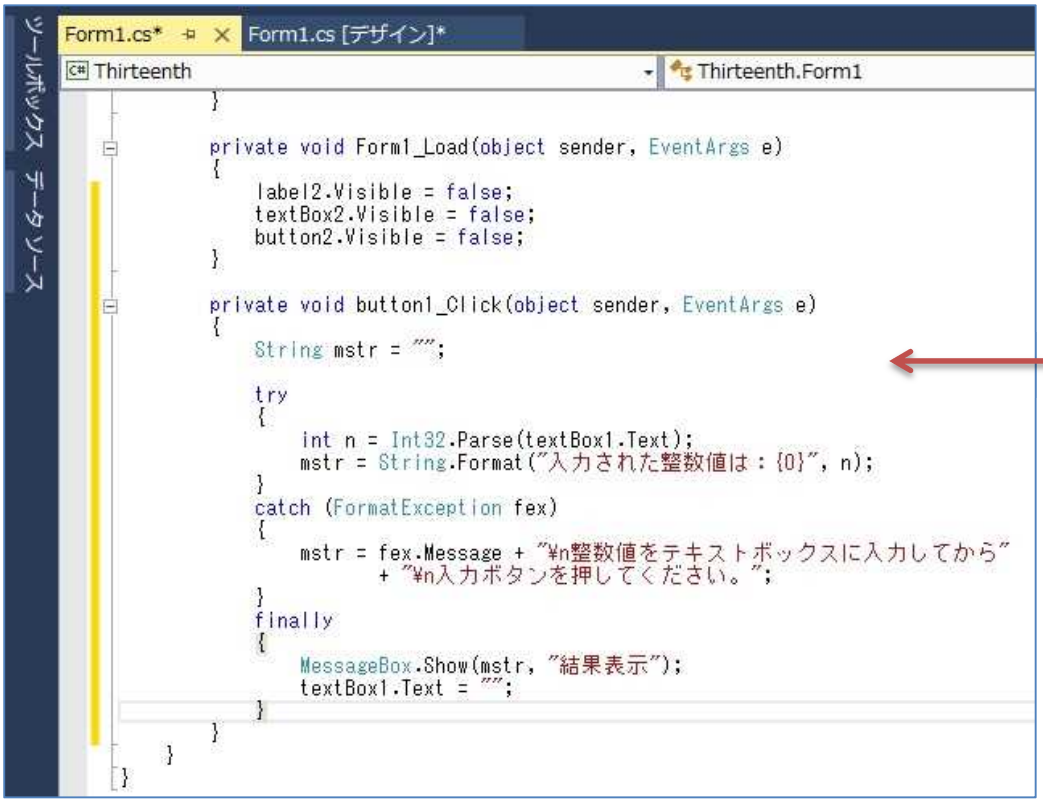
```
private void Form1_Load(object sender, EventArgs e)
{
    label2.Visible = false;
    textBox2.Visible = false;
    button2.Visible = false;
}
```



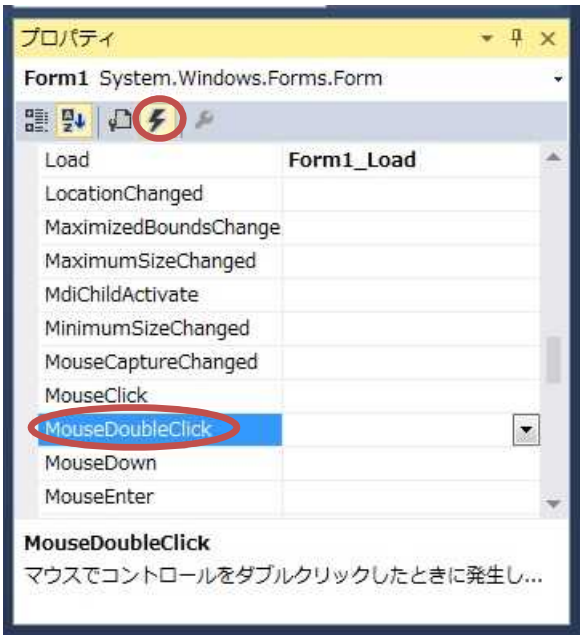
Form1 のフォームデザイナー上で『button1』をダブルクリックして，Form1.cs のプログラムのソースコードを表示し，button1_Click メソッドのブロック内にボタンがクリックされた際の処理（赤枠の部分）を記述していく。

```
private void button1_Click(object sender, EventArgs e)
{
    String mstr = "";

    try
    {
        int n = Int32.Parse(textBox1.Text);
        mstr = String.Format("入力された整数値は：{0}", n);
    }
    catch (FormatException fex)
    {
        mstr = fex.Message + "¥n 整数値をテキストボックスに入力してから"
               + "¥n 入力ボタンを押してください。";
    }
    finally
    {
        MessageBox.Show(mstr, "結果表示");
        textBox1.Text = "";
    }
}
```

＊ FormatException クラスは引数の形式が無効である場合，または複合書式指定文字列の形式が整えられていない例外を扱う。

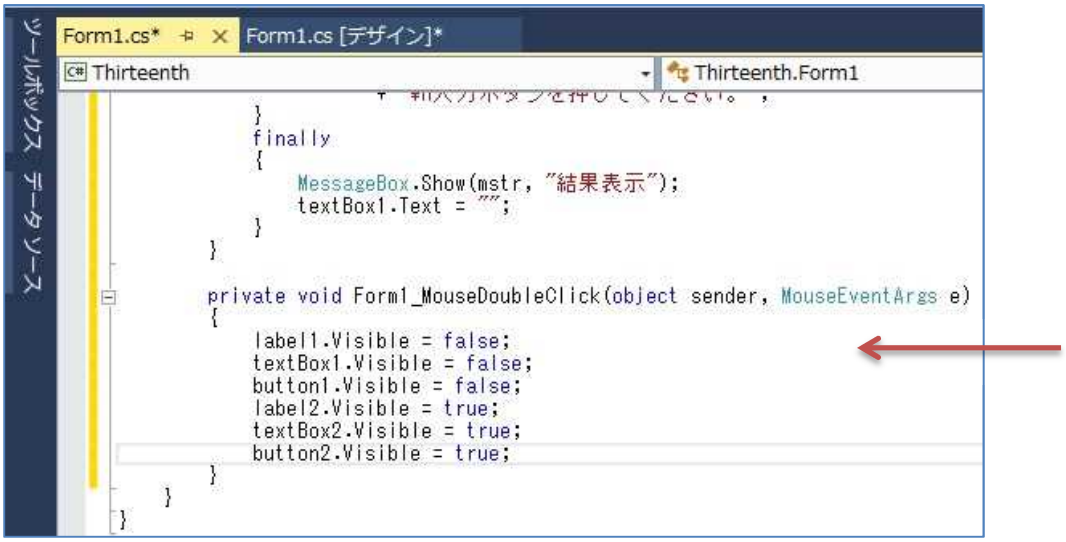


Form1 のフォームデザイナー上で，フォームを選択し（ダブルクリックはしない），プロパティの『イベントボタン』（稲妻のマーク）をクリックした後に，『MouseDoubleClick』の箇所をダブルクリックする。

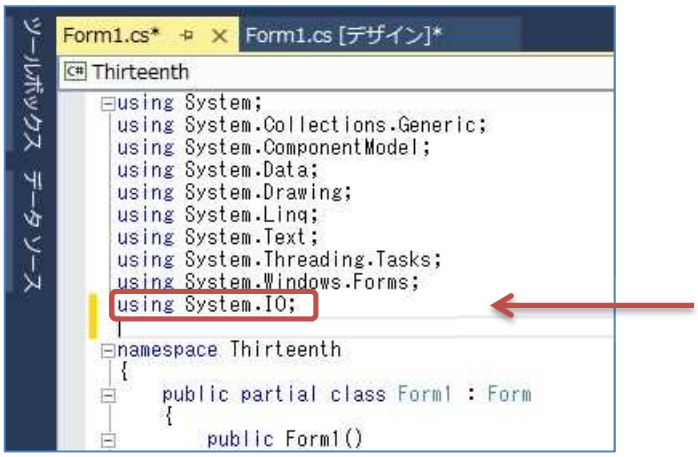


Form1_MouseDoubleClick メソッドのブロック内にフォームがダブルクリックされた際の処理（次のページの赤枠の部分）を記述していく。

```
private void Form1_MouseDoubleClick(object sender, MouseEventArgs e)
{
    label1.Visible = false;
    textBox1.Visible = false;
    button1.Visible = false;
    label2.Visible = true;
    textBox2.Visible = true;
    button2.Visible = true;
}
```



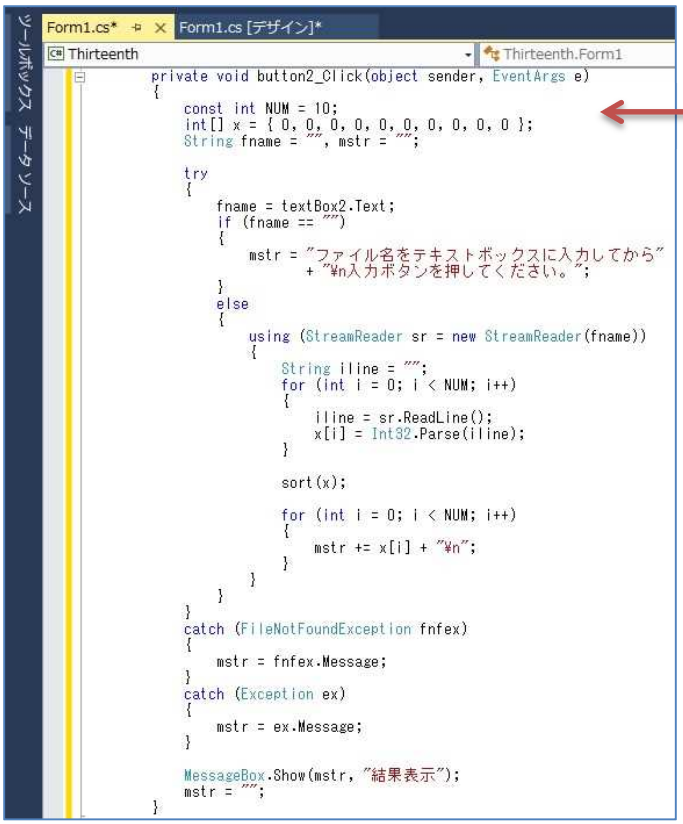
namespace Thirteenth の行より上に，using System.IO; を書き加える。



Form1 のフォームデザイナー上で『button2』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し，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)
{
    const int NUM = 10;
    int[] x = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    String fname = "", mstr = "";

    try
    {
        fname = textBox2.Text;
        if (fname == "")
        {
            mstr = "ファイル名をテキストボックスに入力してから"
                   + "¥n 入力ボタンを押してください。";
        }
        else
        {
```

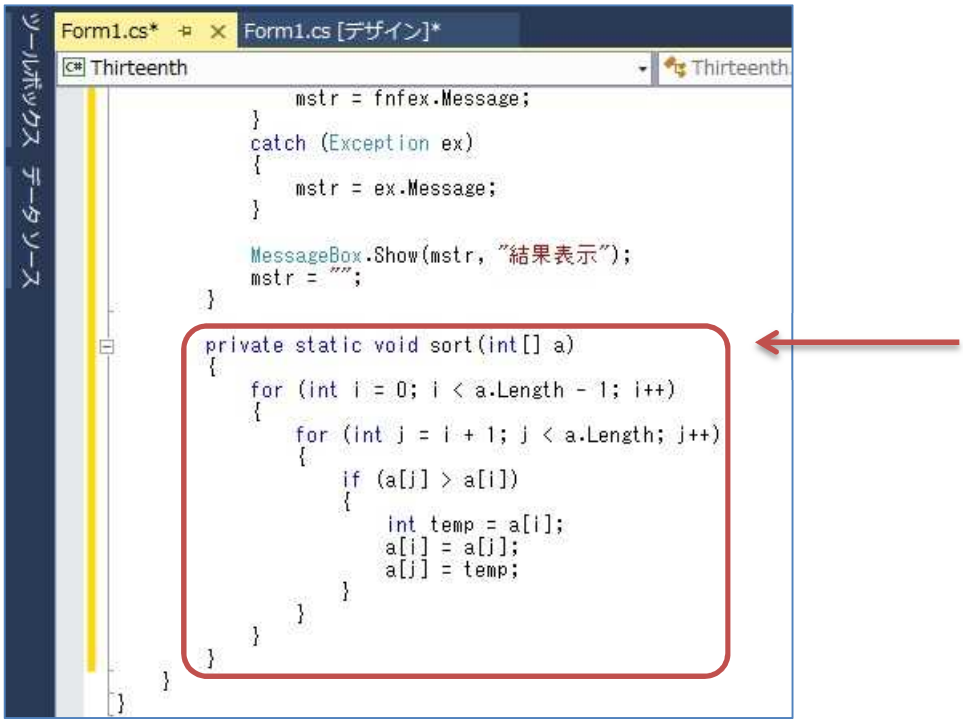
```
            using (StreamReader sr = new StreamReader(fname))
            {
                String iline = "";
                for (int i = 0; i < NUM; i++)
                {
                    iline = sr.ReadLine();
                    x[i] = Int32.Parse(iline);
                }

                sort(x);

                for (int i = 0; i < NUM; i++)
                {
                    mstr += x[i] + "¥n";
                }
            }
        }
    }
    catch (FileNotFoundException fnfex)
    {
        mstr = fnfex.Message;
    }
    catch (Exception ex)
    {
        mstr = ex.Message;
    }

    MessageBox.Show(mstr, "結果表示");
    mstr = "";
}
```



ReadLine() は 1 行分のデータを読み込む。

button2_Click メソッドのブロックより下に sort メソッドを記述する(i と j との違いに注意)。

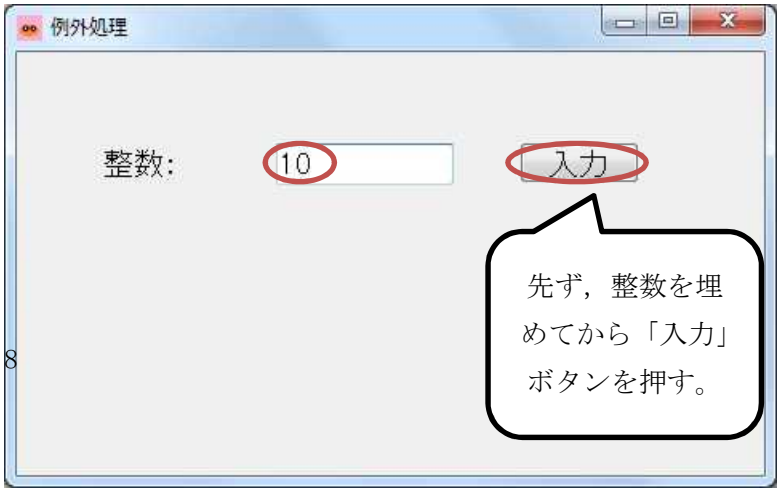
```
        private static void sort(int[] a)
        {
            for (int i = 0; i < a.Length - 1; i++)
            {
                for (int j = i + 1; j < a.Length; j++)
                {
                    if (a[j] > a[i])
                    {
                        int temp = a[i];
                        a[i] = a[j];
                        a[j] = temp;
                    }
                }
            }
        }
```



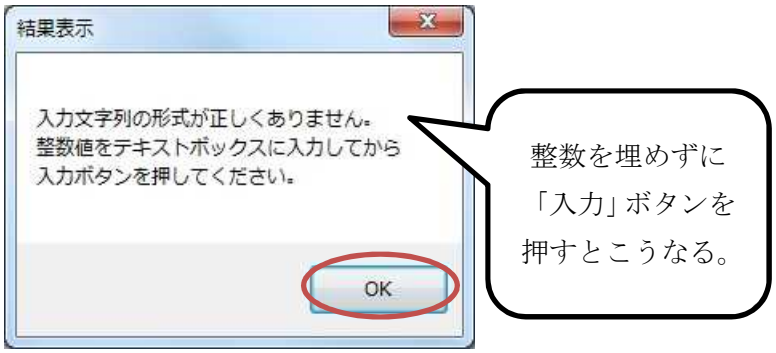
4) プログラムの実行・最終確認

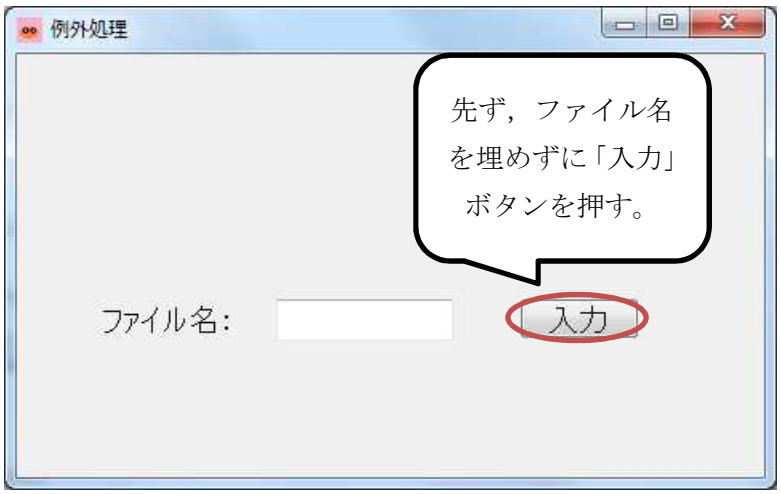
『すべてを保存』ボタンを押してから,『開始』ボタンを押して，プログラムを実行する。
エラーが出ている場合には，修正してから保存，開始と進む。

今回のプログラムでは，整数をテキストボックスに入力して入力ボタンを押す操作と，ファイル名をテキストボックスに入力して入力ボタンを押す操作とが用意されているが，フォームをダブルクリックするとファイル名の入力を促すラベル，テキストボックス及びボタンが現れる。

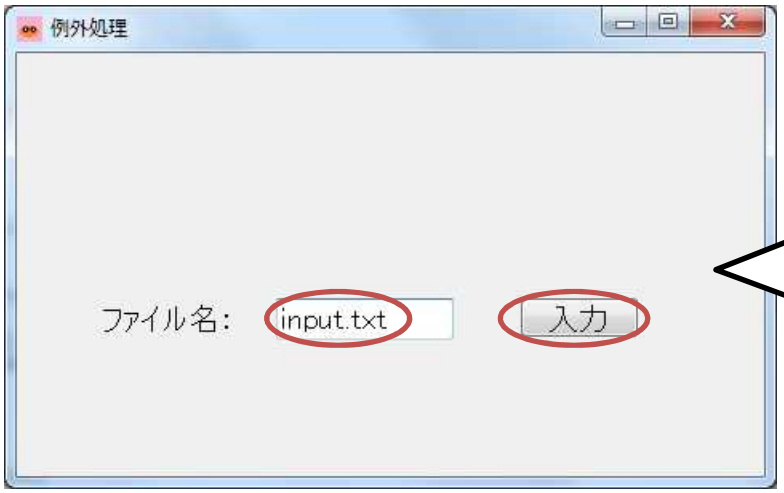

例外処理
整数:
10
入力
先ず，整数を埋めてから「入力」ボタンを押す。


結果表示
入力された整数値は：10
OK

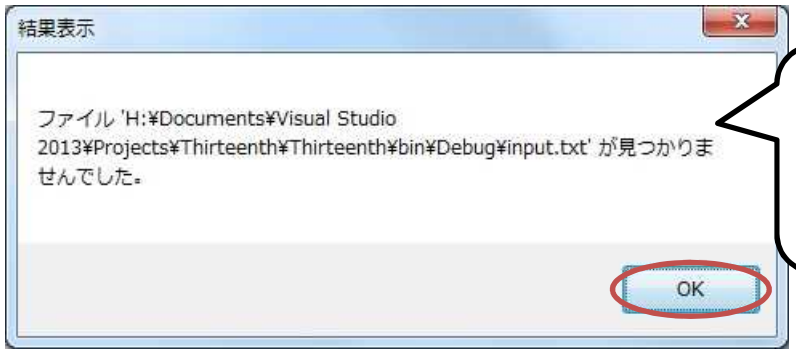

結果表示
入力文字列の形式が正しくありません。
整数値をテキストボックスに入力してから
入力ボタンを押してください。
OK
整数を埋めずに「入力」ボタンを押すとこうなる。


例外処理
先ず，ファイル名を埋めずに「入力」ボタンを押す。
ファイル名:
入力


結果表示
ファイル名をテキストボックスに入力してから
入力ボタンを押してください。
OK
これは if 文による処理

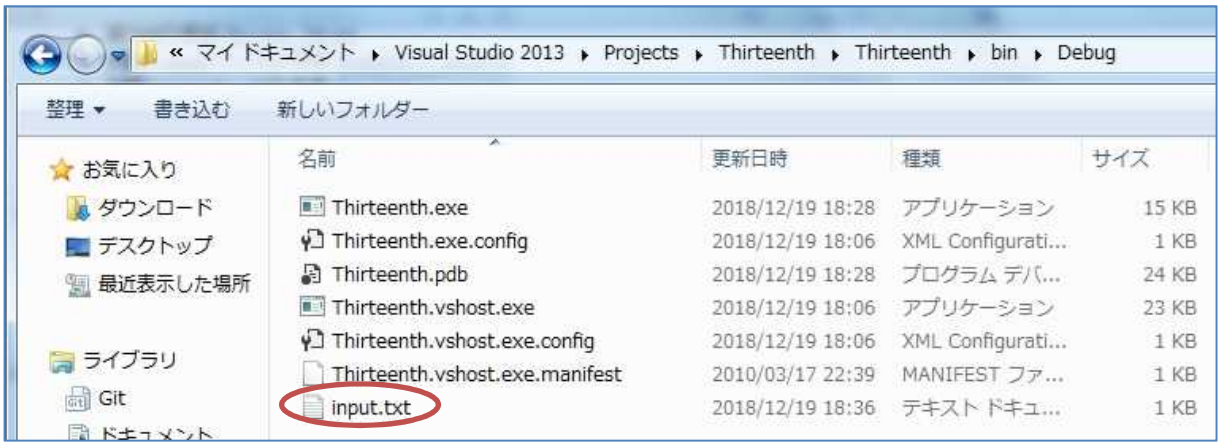


次に，ファイル名として input.txt を埋めて「入力」ボタンを押す。



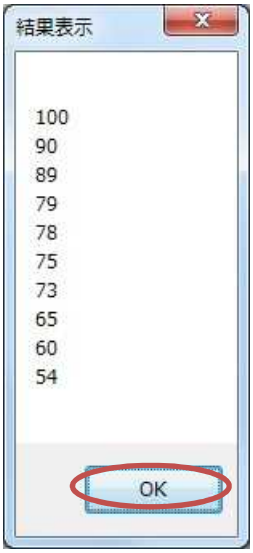
これは FileNotFoundException をキャッチしたことによる例外処理

ここで，ダウンロードしたテキストファイル『input.txt』をコピーして，H:¥Documents¥Visual Studio 2013¥Projects¥Thirteenth¥Thirteenth¥bin¥Debug の中に貼り付ける。



＊ Thirteenth.exe は実行形式のファイルで，直接実行が可能である。今回の StreamReader にはパス名を与えていないので，操作するファイルはこの実行形式のファイルと同じ位置にある必要がある。

このページの先頭の図の状態で，再度，「入力」ボタンを押す。



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¥Thirteenth¥Thirteenth

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_13th.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。