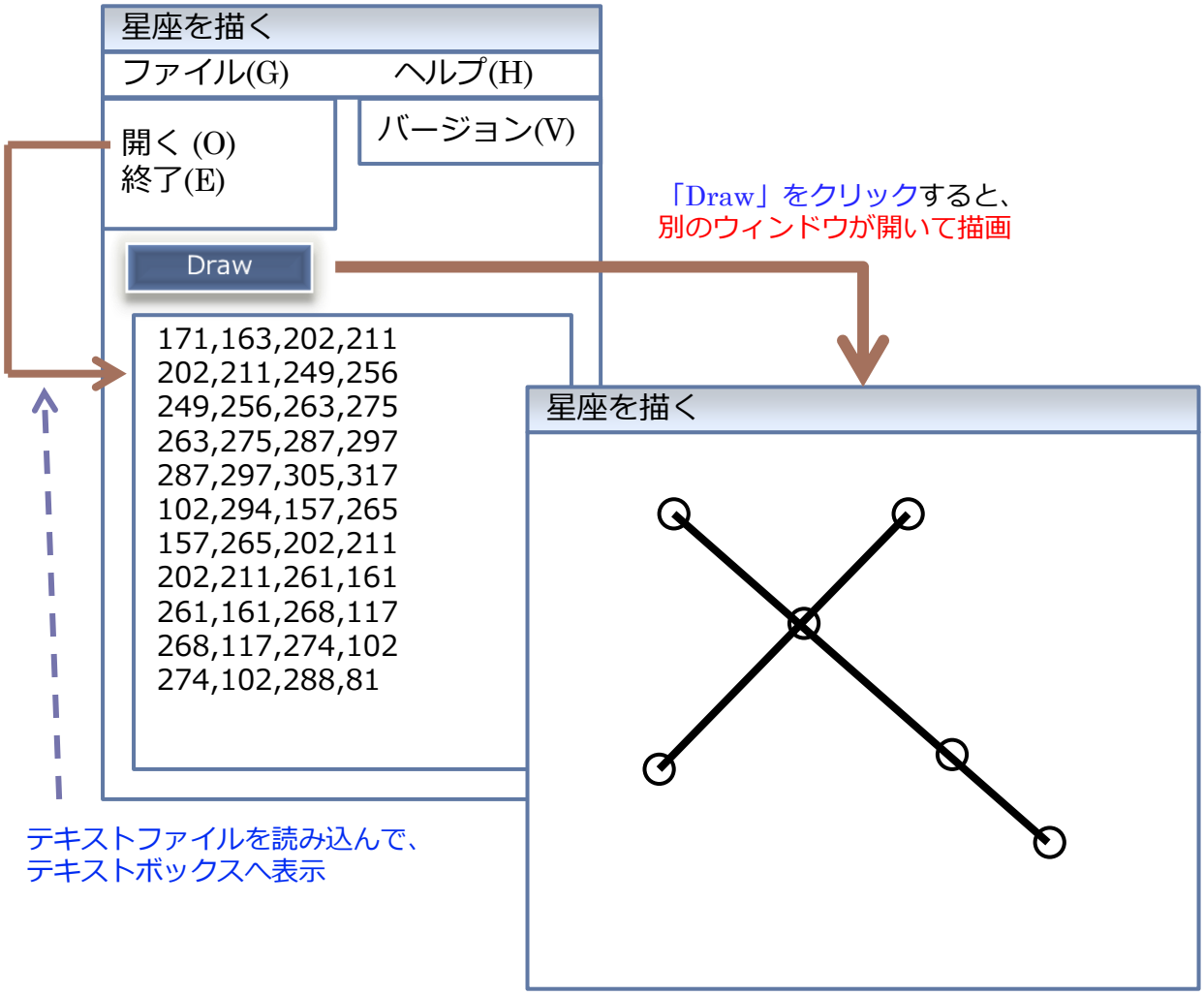
【05】 「テキストファイルからの入力」と「別のフォームを開く」をいっぺんにやる 星座を描く

1 今回作成するアプリケーションの概要

座標の記入されているテキストファイルを読み込んで、表示
ただし、表示するのは別のウィンドウ（フォーム）

◆ 行われる動作

[1] 座標の記入されているテキストファイルを指定する。
[2] テキストファイルで読み込んだ内容をテキストボックスにそのまま表示する
[3] 「Draw」ボタンをクリックすると別のウィンドウが開く
[4] テキストボックスの内容を読み込んで、その座標使ってウィンドウ上に図を描く



- 使用者とコンピュータの関係をまとめる

[使用者 → コンピュータ] メニューの「開く」をクリック
[使用者 ← コンピュータ] ファイルオープンダイアログが開く
[使用者 → コンピュータ] 読み込むファイルを選ぶ
[使用者 ← コンピュータ] ファイルの内容を読み込み、
テキストボックスに表示

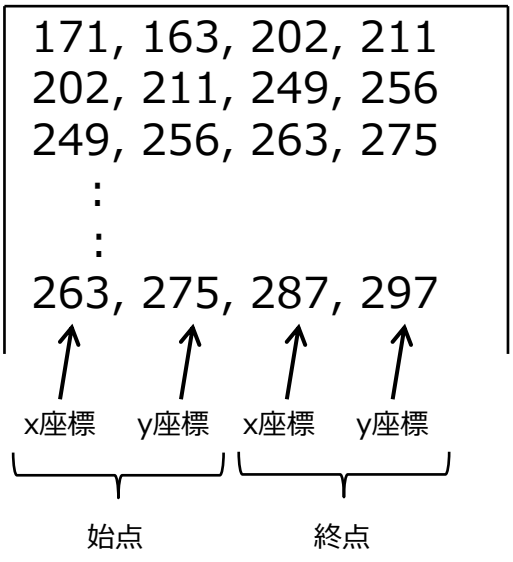
[使用者 → コンピュータ] 「Draw」ボタンをクリック
[使用者 ← コンピュータ] 新しいウィンドウを開く
テキストボックスの内容を読み取る
新しいウィンドウで描画

- 必要なコントロールは次の通り
  - TextBox コントロール（テキストの読み込み、記入、編集）
  - Button コントロール（「Draw」ボタン）
  - MenuStrip コントロール（メニューを表示、実行）
  - OpenFileDialog コントロール（読み込むファイルを示す）

- 利用するフォームは次の２つ
  - テキストファイルを読み込んで、内容を表示 ← こっちがメイン
  - 座標データを受け取り、描画する

2 テキストファイルを用意する

座標を記入したテキストファイルを用意する。
形式は次の通り、始点の座標と終点の座標を一行にしたものである。



例として、白鳥座を描くファイル **Cygnus.txt** とオリオン座を描く **Orion.txt** を次に示す。

Cygnus.txt

```
171,163,202,211
202,211,249,256
249,256,263,275
263,275,287,297
287,297,305,317
102,294,157,265
157,265,202,211
202,211,261,161
261,161,268,117
268,117,274,102
274,102,288,81
```

Orion.txt

```
87,57,56,115
56,115,42,118
42,118,66,165
66,165,86,186
86,186,123,282
123,282,103,359
103,359,188,345
188,345,142,263
142,263,161,200
161,200,135,162
135,162,86,186
161,200,229,124
229,124,248,171
248,171,249,191
249,191,244,204
244,204,238,238
123,282,133,273
133,273,142,263
```

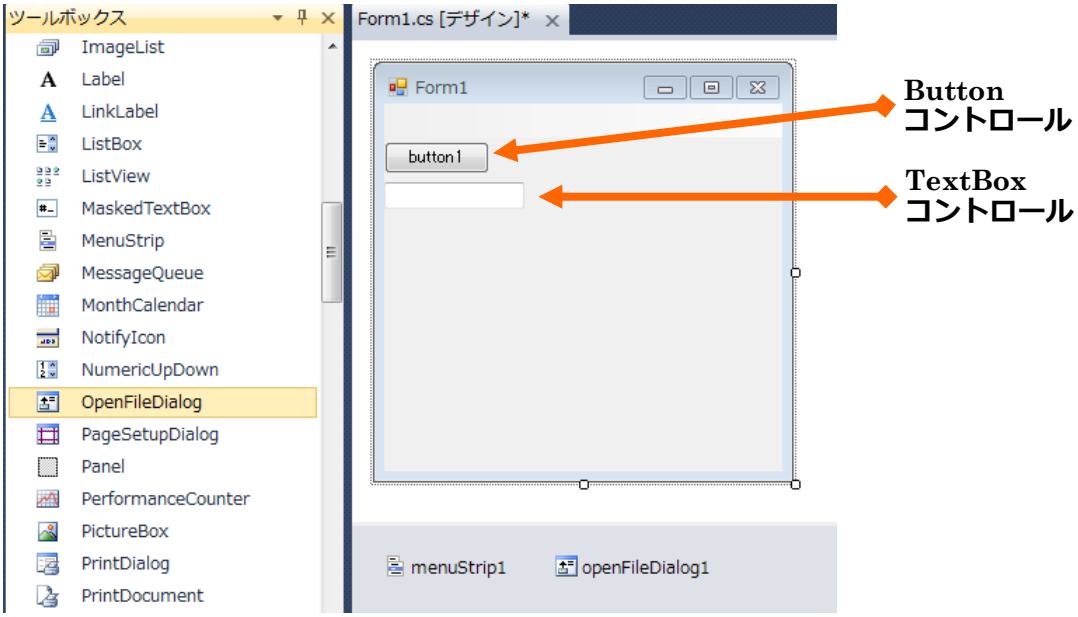
3 Visual Studio 2010 の起動と新規プロジェクトの作成

- 1回目でやったとおり、Visual Studio 2010 を起動
  [1]「スタート」→ [2]「すべてのプログラム」→ [3]「Visual Studio 2010」のフォルダ
  → [4]「Visual Studio 2010」のアイコン

- 新規プロジェクトも1回目の手順で作成
  [1] メニュー →「ファイル」→ [2]「新規作成」→ [3]「プロジェクト」
  [4]「Visual C#」→ [5]「Windowsフォームアプリケーション」
  [6]「プロジェクト名」を入力 **（今回は JKJ05）**
  [7]「参照…」をクリックして、プロジェクトを保存する場所を選択
  [8]「ソリューションのディレクトリを作成」のチェックは はずす
  [9]「OK」をクリック

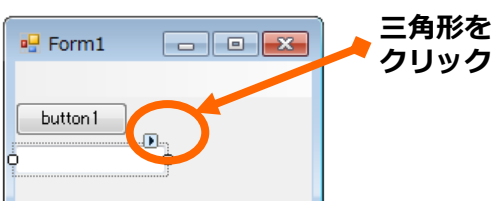
4 メインのフォームのコントロールの配置

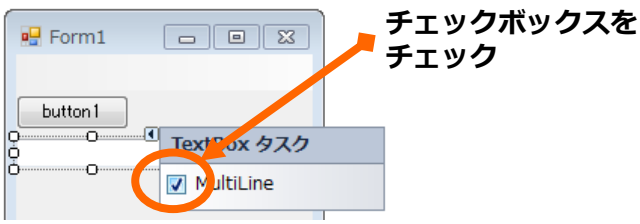
- **Button コントロール、TextBox コントロール** をツールボックスから
  ドラッグ&ドロップして配置
- **MenuStrip コントロール、OpenFileDialog コントロール**を
  ダブルクリックして追加する。
  （MenuStrip コントロール、OpenFileDialogコントロールもフォーム上には配置されない）



- **フォームを縦長の形にする。**
- **ボタンの位置をメニューのすぐ下に**
- **ボタンのすぐ下に TextBox を**
- **TextBox は MultiLine を設定して、フォームの大きさに合わせる**

MultiLine の設定方法：



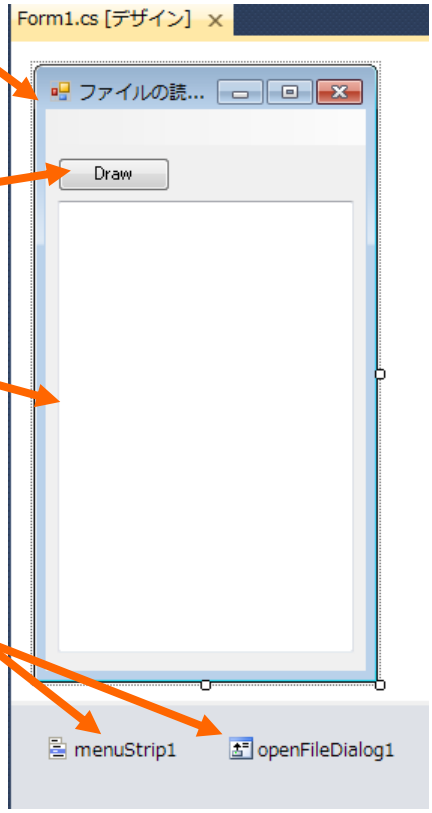


■それぞれのコントルールのプロパティを次のように設定する

Name	Form1
Text	ファイルの読み込み

Name	btnDraw
Text	Draw

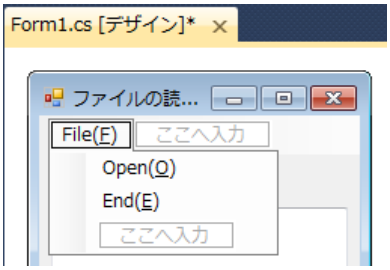
Name	tbPosition



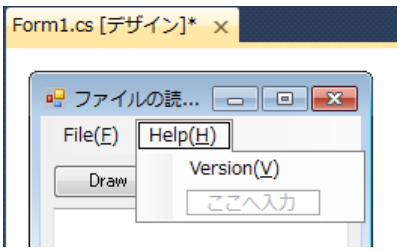
（ 変更なし ）

5 メニューの作成

■前回と同様にして、メニューを次のように作成する



File(&F)
├ Open(&F)
└ End (&E)

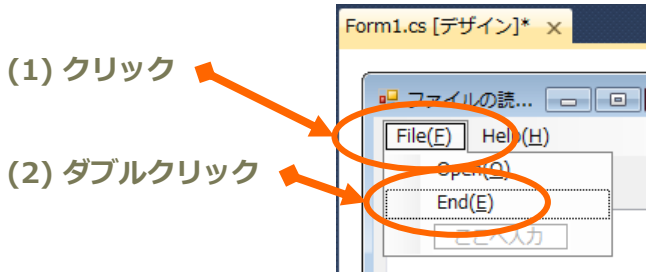


Help(&H)
└ Version (&V)

6 メニューを選択したときのプログラムを入力

■ メニューの「End(E)」をクリックされたときの処理を入力

(1) File(F) をクリックしてメニューを出す
(2) End(E) をダブルクリックしてプログラムのタグを出す



(3) プログラムを終了させる関数 Close(); を入入力

自動的に入力された部分

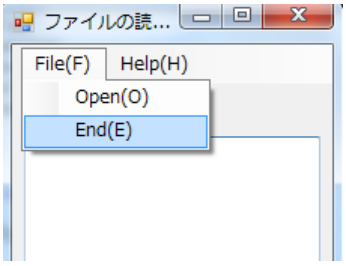
```
namespace JKJ05
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void endEToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

自動的に入力された部分

この１行を入力

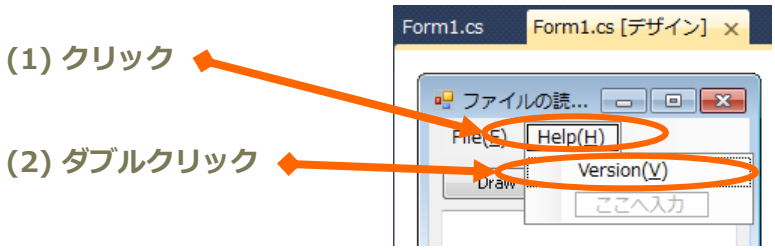
■ プログラムを実行してみる。

メニューの「End(E)」をクリックすると、プログラムが終了するか確認



■ メニューの「Version(V)」をクリックされたときの処理を入力

(1) Help(H)をクリックしてメニューを出す
(2) Version(V) をダブルクリックしてプログラムのタグを出す



(3) メッセージボックスを表示する関数 MessageBox.Show を記述

自動的に入力された部分

```
private void versionVToolStripMenuItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("星座を描くプログラム¥n入力:占部弘治", "バージョン情報");
}
```

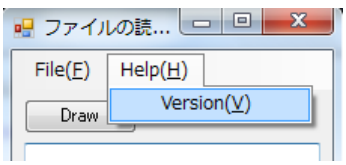
この１行を入力

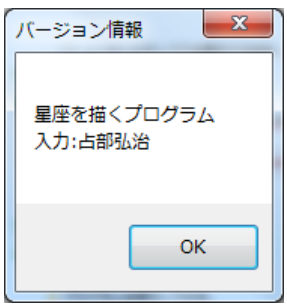
自分の名前に変更

自動的に入力された部分

■ プログラムを実行してみる。

メニューの「Version(V)」をクリックすると、バージョン情報を掲載したメッセージボックスが表示されることを確認





7 テキストファイルからの入力

- メニューの「 Open(O) 」 → ファイルを選ぶウィンドウ → ファイルを読み込むまでの流れ

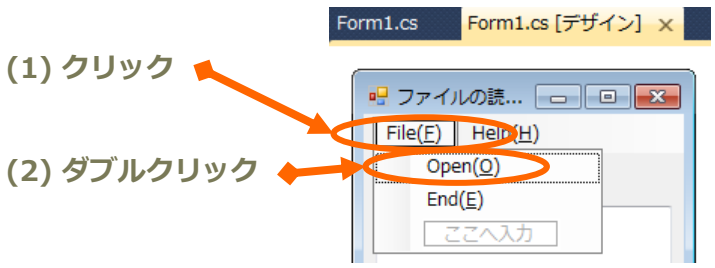
1) メニューの「 Open(O) 」がクリック ← 「 Open(O) 」がクリックされたイベント発生
2) ファイルを選ぶウィンドウを表示 ← 「 Open(O) 」がクリックされたイベントで処理する内容
3) ファイルを選ぶ
4) 「開く」がクリックされる ← 「開く」がクリックされたイベント発生
5) 選択されたファイルが存在するかチェックする
6) ファイルを Open する
7) ファイルを読み込む
8) ファイルを Close する

5)〜8) → 「開く」がクリックされたイベントで処理すること

- メニューの「 Open(O) 」 がクリックされたときのプログラムを入力

ファイルを選択するウィンドウは Windows API で用意されており、Visual C# では **OpenFileDialog コントロール**を使って利用することができる。
今回、すでにフォームのは OpenFileDialog コントロールの openFileDialog1 が読み込まれているので、メニューの「Open(O)」がクリックされたとき、openFileDialog1 を表示するようにする。

(1) FILE(F)をクリックしてメニューを出す
(2) Open(O)をダブルクリックしてプログラムのタグを出す



(3) OpenFileDialog を表示する関数を記述

自動的に
入力された
部分

```
private void openOToolStripMenuItem_Click(object sender, EventArgs e)
        {
            openFileDialog1.ShowDialog();
        }
```

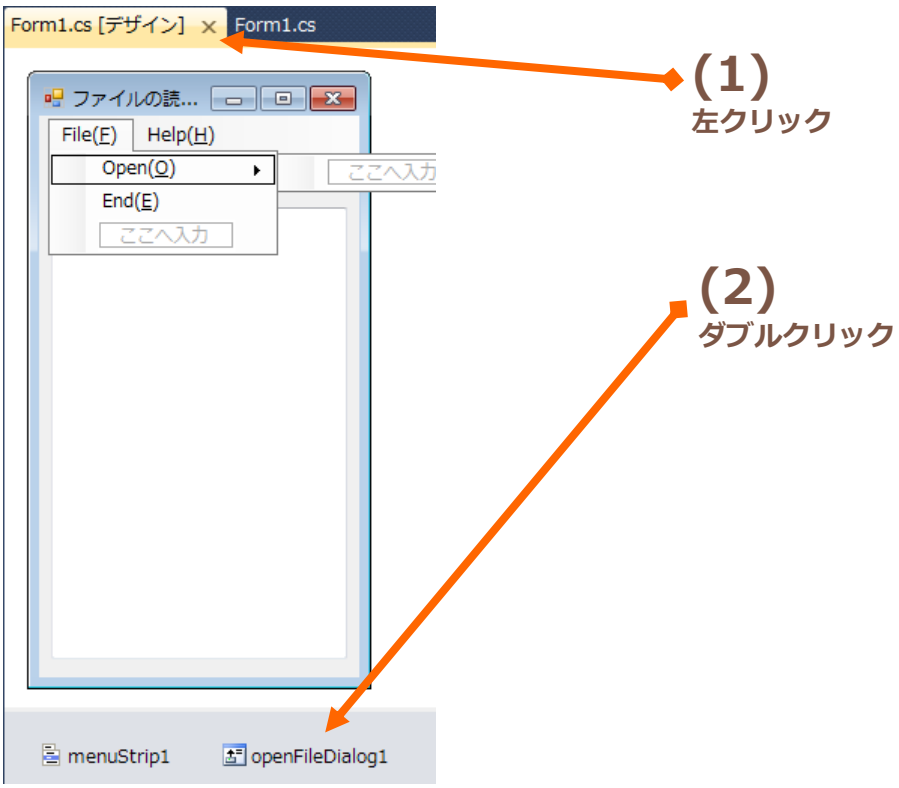
この１行を入力

自動的に
入力された
部分

- **OpenFileDialog コントロール**はフォルダの移動、新規作成やファイルの選択を実施する。
  こちらで用意しないといけないのは「開く」をクリックしたときの処理である。

- openFileDialog1 の「開く」がクリックされたときの処理を入力する。

(1) Form1.cs [デザイン] をクリックして、コード入力のタグからフォームのデザイン
    するタグへ戻る
(2) openFileDialog1 をダブルクリックする



(3) ファイル入力に必要な Stream を使うための設定 を入力

プログラムの最初のほうに using System～ とたくさん並んでいる場所に
次の１行を追加する。

すでに入力されている部分

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

using System.IO;

namespace JKJ05
{
```

この１行を入力

すでに入力されている部分

(4) openFileDialog1 の「開く」がクリックされたときの処理を入力

自動的に入力された部分

```
private void openFileDialog1_FileOk(object sender, CancelEventArgs e)
    {
        if (File.Exists(openFileDialog1.FileName) == false) return;

        tbPosition.Text = "";

        StreamReader sr = new StreamReader(openFileDialog1.FileName, Encoding.Default);

        while (true)
        {
            string line = sr.ReadLine();

            if (line == null)
            {
                break;
            }

            tbPosition.Text += line + "¥r¥n";
        }
    }
```

この部分を入力

自動的に入力された部分

プログラムの解説

```
if (File.Exists(openFileDialog1.FileName) == false) return;
```
openFileDialog1 で選択されたファイルが存在しているかどうかのチェック

```
tbPosition.Text = "";
```
tbPosition の表示をクリアする

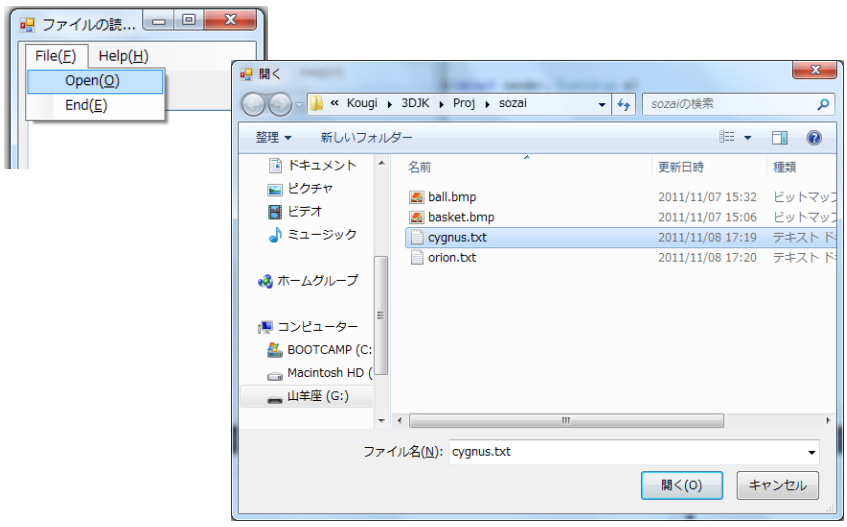
```
StreamReader sr = new StreamReader(openFileDialog1.FileName, Encoding.Default);
```
openFileDialog1 で選択されたファイル を StreamReader sr で開く

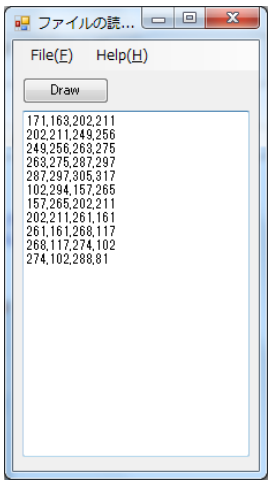
```
while (true)
 {
      string line = sr.ReadLine();
```
StreamReader sr から 1 行読み込んで、文字列 line へ格納

```
      if (line == null)
      {
          break;
      }
```
文字列 line がからっぽ＝ファイルの最後まで読み込んだ
なのでループから脱出

```
      tbPosition.Text += line + "¥r¥n";
}
```
文字列 line と 改行 を tbPosition の表示へ追加

- この段階で実行してみる。
  メニューから File(F) → Open(O) へクリックし、OpenFileDialog を開いて、
  cygnus.txt を読み込むと、cygnus.txt の内容が tbPosition へ読み込まれる

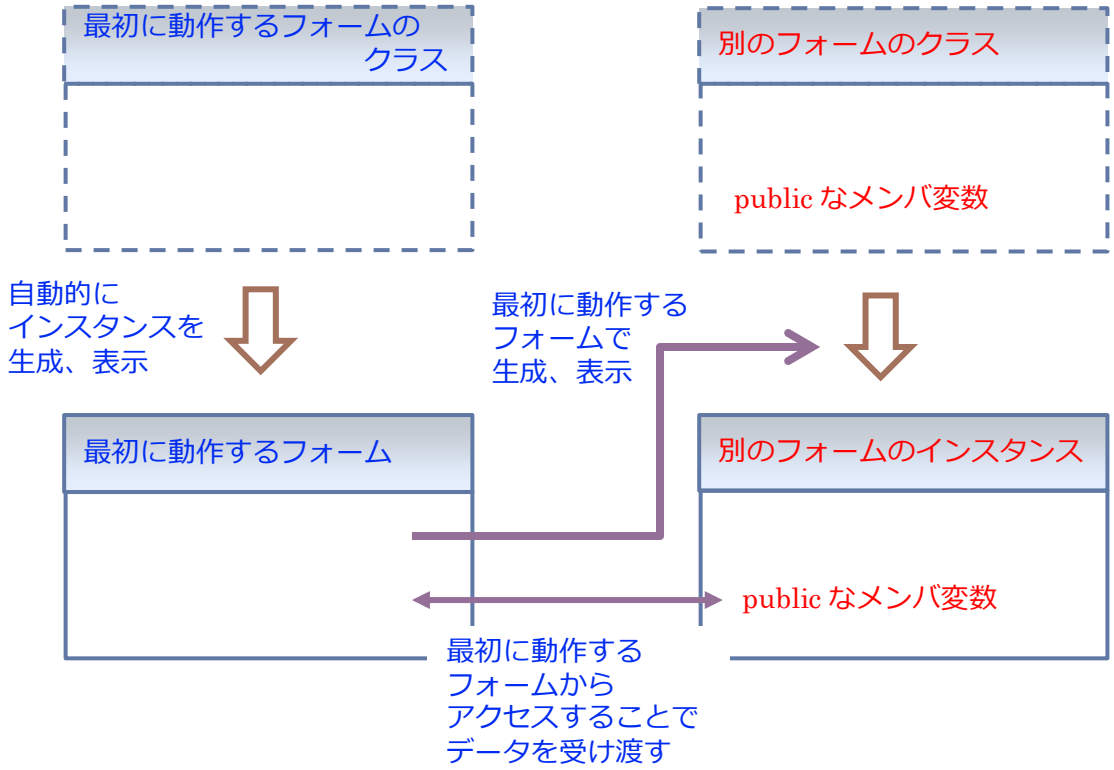




8 別のウィンドウを開くために

別のウィンドウを開くということは、別のフォームを用意して呼び出すということである。
別のフォームを用意するということは、フォームのクラスを別に用意するということである。
フォームのクラスが別になるということは、

- 別のフォームのクラスを用意し、そのフォームでの処理はそちらに記述する
- 動作しているフォームから別のフォームを呼び出すためには、
  フォームクラスに対するインスタンスを生成し、呼び出す必要がある。
  （最初に動作するフォームは Visual Studio によって自動的にインスタンスを生成し、呼び出している）
- 動作しているフォームから別のフォームへデータを渡すためには、
  別のフォームのクラスに public なメンバ変数を用意し、
  インスタンスを生成してから、その public なメンバ変数へ値を渡す



今回のプログラムは次のように二つのフォームを関係付ける。

・星座を描くフォーム frmDraw は 星座を描く 処理しか行わない

・星座を描くデータは 線の始点と終点の座標の配列と 線が最大本数 とし、これは
ファイルを読み込んだ元のフォーム Fomr1 の tbPositon.Text より読み込む

・frmDraw から tbPosition は読めない。
そこで、元のフォーム Form1 でtbPosition.Text のテキスト解析を行い、
frmDraw の座標の配列と最大本数へ格納させる

・frmDraw の座標と配列と最大本数は public なメンバとして設定しないといけない

・Form1 の btnDraw がクリックされたら、frmDraw のインスタンスが生成されて、
frmDraw のメンバ変数に値が与えらえれ、frmDraw が表示される。

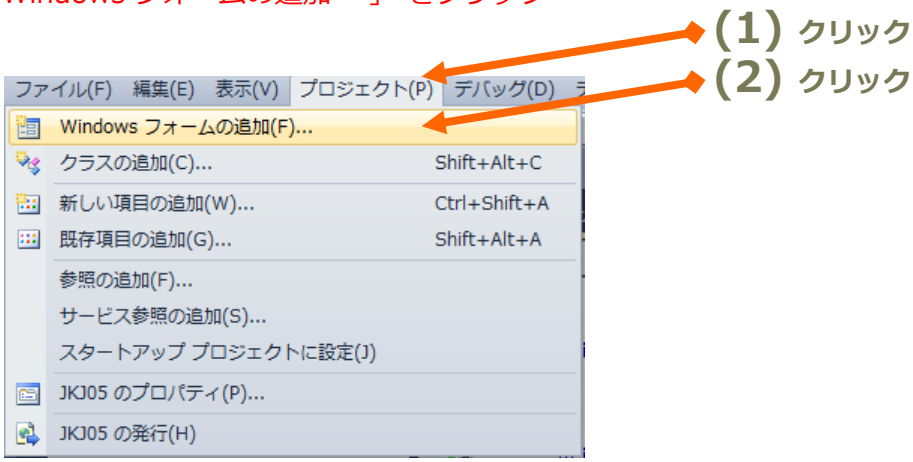
・frmDraw は メンバ変数の値を元に星座を描く

そこで、次の手順でプログラムを作成する

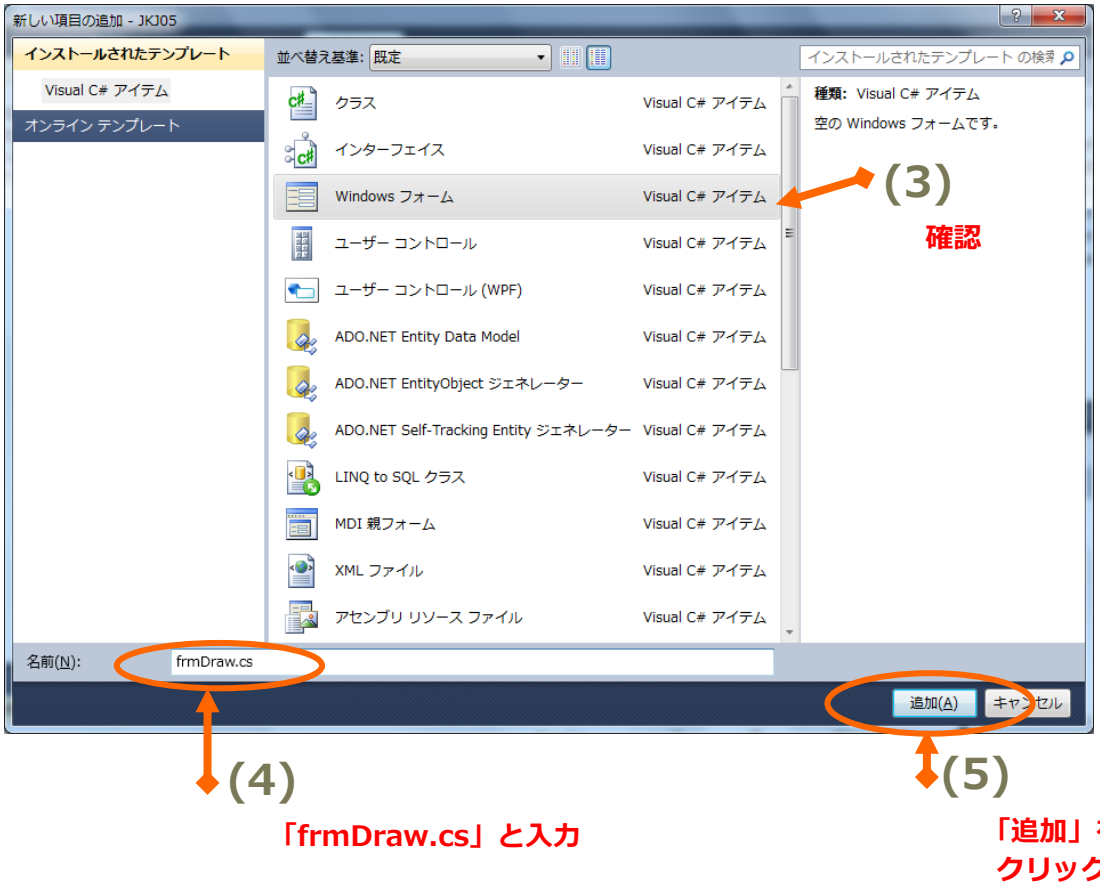
[1] フォーム frmDraw を新規作成する
[2] 新規作成した frmDraw に public なメンバ変数を設定する
[3] Form1 の btnDraw ボタンを押されたときの処理を記述する
・フォーム frmDraw のインスタンス frmDraw1 を生成する
・tbPosition.Text を解析し、フォーム frmDraw1 のメンバ変数に値を与える
・フォームのインスタンス frmDraw1 を表示する
[4] フォーム frmDraw のプログラムを入力する
・再描画イベントで星座を描く

9 フォームを新規作成する

(1) Visual Studio のメニューの「プロジェクト」をクリック
(2) 「Windows フォームの追加 …」 をクリック

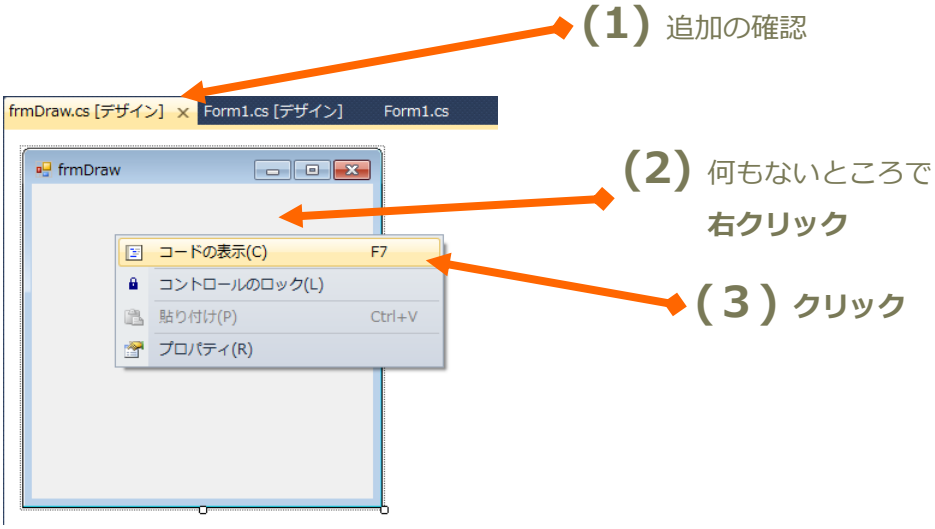


(3) 「Windows フォーム　Visual C# アイテム」が選択されているかを確認
(4) 名前に「frmDraw.cs」と入力
(5) 「追加」をクリック



１０ 追加したフォームに public なメンバ変数を追加

(1) frmDraw.cs [デザイン] のタグが追加される
(2) フォームの何もないところで、右クリックをしてメニューを表示
(3) 右クリックメニューを表示の「コードの表示」をクリック



(4) 次の public なメンバ変数の宣言を入力

すでに入力されている部分

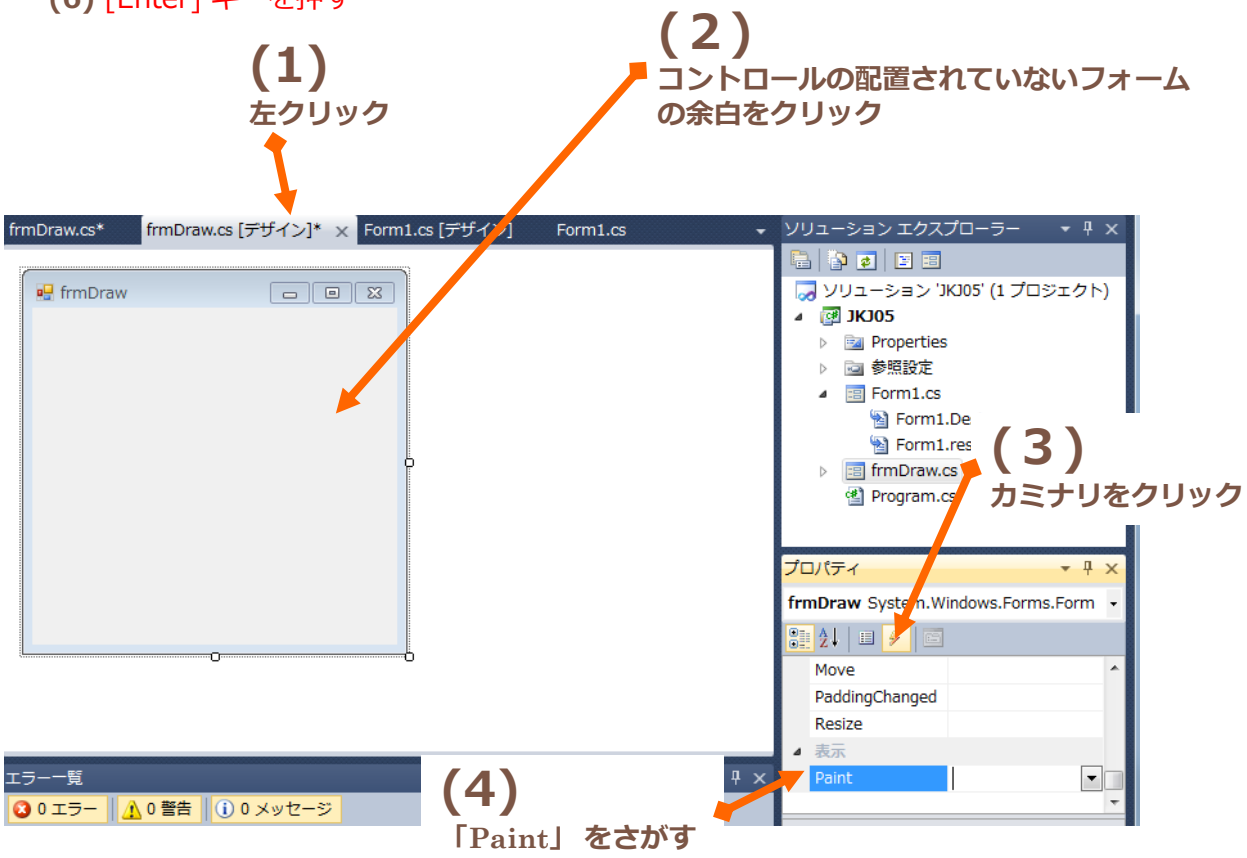
```
namespace JKJ05
{
    public partial class frmDraw : Form
    {
        // 線の最大本数
        public int max;
        // 線の始点の座標を格納する配列
        public float[] xs = new float[100];
        public float[] ys = new float[100];
        // 線の終点の座標を格納する配列
        public float[] xe = new float[100];
        public float[] ye = new float[100];

        public frmDraw()
        {
            InitializeComponent();
        }
    }
}
```

この部分を入力

すでに入力されている部分

11 追加したフォームの再描画イベントの入力

(1) frmDraw.cs [デザイン] をクリックして、コード入力のタグからフォームのデザインするタグへ戻る
(2) frmDraw.cs のプロパティを選ぶ。（フォームのコントロールの配置してない場所をクリック）
(3) プロパティウィンドウのカミナリのところをクリックしてイベント一覧を表示
(4) イベント一覧の中から「Paint」を探す
(5) Paint の右側のテーブルをクリックしてカーソルが点滅した状態にする
(6) [Enter] キーを押す



(5)「Paint」 の右側のセルをクリック

(6)「Paint」 の右側のセルでカーソルが点滅していたら、[Enter]キーを押す

(7) frmDraw_Paint という関数が作られる。
この frmDraw_paint が再描画のときに実行される関数である。

この関数にすでに配列に読み込まれている座標から星座をグラフィックで描く部分を入力する

自動的に入力された部分

```
private void frmDraw_Paint(object sender, PaintEventArgs e)
    {
        // フォーム用の Graphics を生成
        Graphics g = this.CreateGraphics();

        for (int i = 0; i < max; i++)
        {
            // 始点と終点を線で結ぶ
            g.DrawLine(Pens.Black, xs[i], ys[i], xe[i], ye[i]);

            // 始点に半径 2 の円を描く
            g.DrawEllipse(Pens.Black, xs[i] - 2, ys[i] - 2, 4, 4);

            // 終点に半径 2 の円を描く
            g.DrawEllipse(Pens.Black, xe[i] - 2, ye[i] - 2, 4, 4);
        }

        // フォーム用の Graphics である g を消滅させる
        g.Dispose();
    }
```

この部分を入力

自動的に入力された部分

１２ 元のフォームから frmDraw フォームを生成、表示

btnDraw がクリックされたとき、

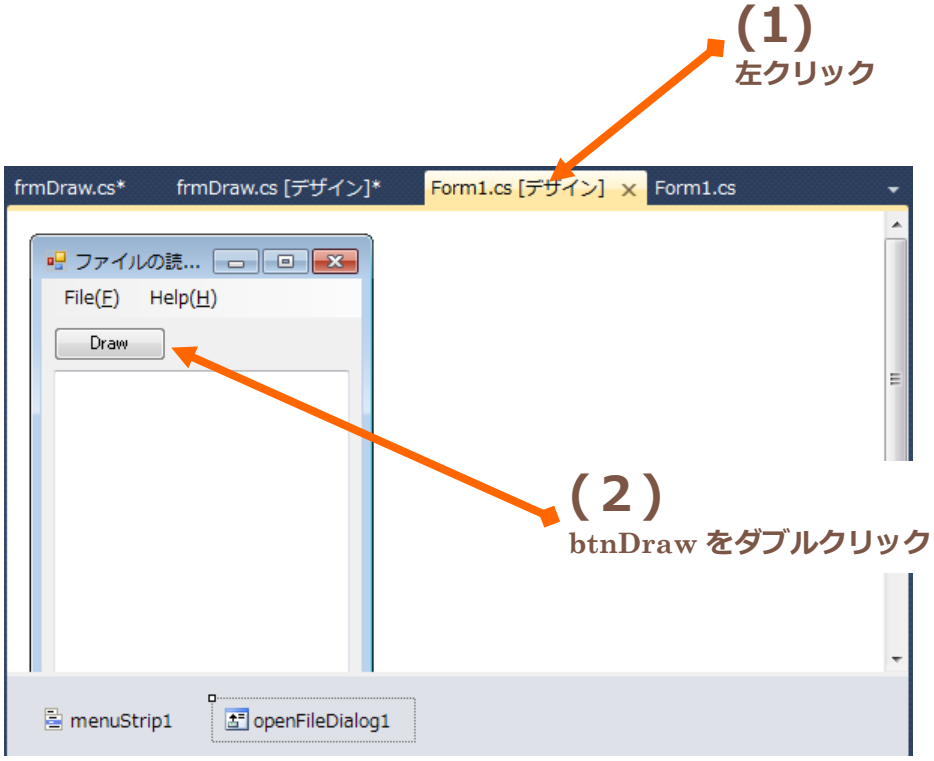
・Form のクラス frmDraw のインスタンス frmDraw1 を生成
・tbPosition.Text のテキストで書かれた座標データを frmDraw1 の配列に格納
・frmDraw1 を表示

という処理が実施される。

frmDraw1 が表示されれば、frmDraw1の再描画イベントによって座標の配列より、星座が描かれる

btnDraw がクリックされたときに呼び出される関数を設定するため、次の操作を行う。

(1) From1のデザイナーのタグを選択
(2) btnDraw でダブルクリック



(3) btnDraw_Click という関数が作られる。
この **btnDraw_Click** が btnDraw がクリックされたときに実行される関数である。
この関数はまず、フォーム frmDraw のインスタンス frmDraw1 を生成する
つぎに tbPosition.Text 配列に読み込まれているテキストデータから
座標を検出して、frmDraw1 のメンバに与える。
最後に frmDraw1 を表示する。

自動的に入力された部分

```
private void btnDraw_Click(object sender, EventArgs e)
    {
        // frmDraw のインスタンス frmDraw1 を生成
        frmDraw frmDraw1 = new frmDraw();

        // tbPosition.Text の内容を Stream sr として利用
        StringReader sr = new StringReader(tbPosition.Text);

        int i = 0;

        while (true)
        {
            // Stream sr より一行読み込んで、 文字列 line へ
            string line = sr.ReadLine();

            //  line に何も入っていなかったら、ファイルが終了したとして、ループを出る
            if (line == null)
            {
                break;
            }

            //  line に入っている文字列をカンマで区切って配列 sp へ格納
            string[] sp = line.Split(',');

            //  配列 sp に入っている文字列を実数に変換して座標の配列へ
            frmDraw1.xs[i] = float.Parse(sp[0]);
            frmDraw1.ys[i] = float.Parse(sp[1]);
            frmDraw1.xe[i] = float.Parse(sp[2]);
            frmDraw1.ye[i] = float.Parse(sp[3]);

            //  行数をカウント、ループ終了時に行の数が i に格納される
            i++;
        }

        //  最大値を frmDraw のメンバ変数 max 格納
        frmDraw1.max = i;
        //  frmDraw1 を表示する
        frmDraw1.ShowDialog();
    }
```

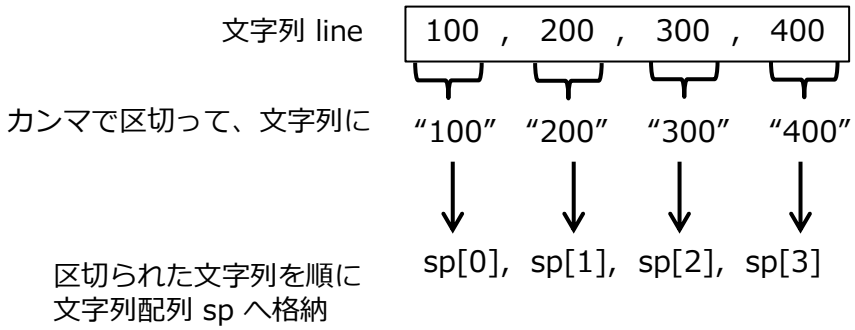
この部分を入力

自動的に入力された部分

```
string[] sp = line.Split(',');
```

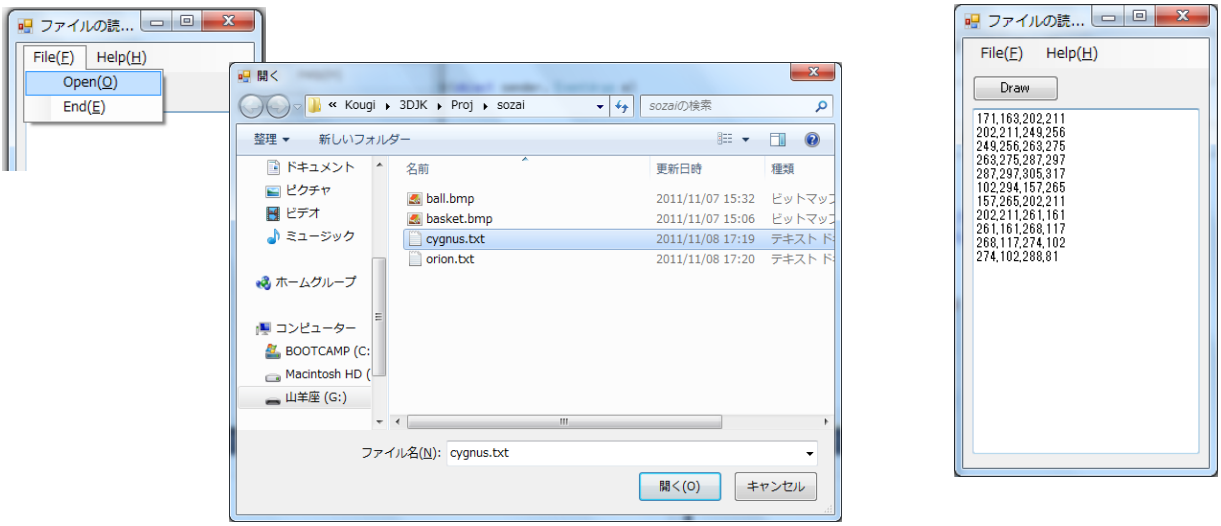
文字列 line を シングルコーテーション（’ , Shift + 7 で入力）で囲まれた文字カンマ（, ）で区切って、区切られた文字列を配列 sp へ格納する

(1) カンマ , で文字列を区切る
(2) 区切られた文字列はそれぞれ文字列配列 sp へ格納される

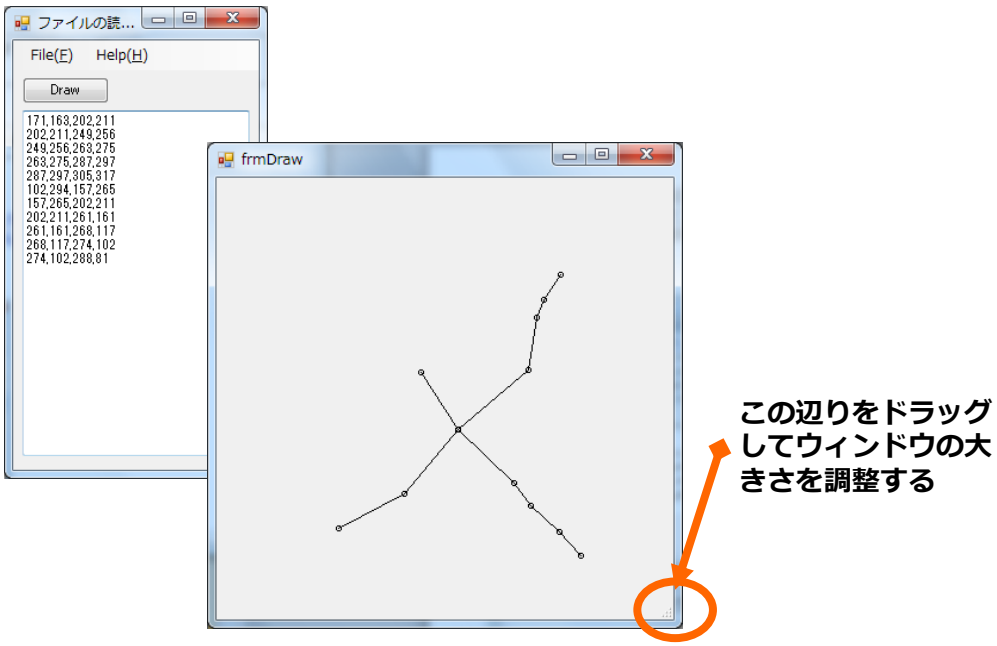


13 とりあえず完成

- メニューの「Open(O)」をクリックすると、OpenFileDialog を呼び出す
- 星座データの入ったファイルを選択して、「開く」
- ファイルが読み込まれ、テキストボックスにその内容を表示



- 「Draw」ボタンをクリックすると、別のウィンドウが開いて、星座が表示される



１４ まとめ

- ファイルの選択は OpenFileDialog コントロール
  - メンバの ShowDialog() を呼び出すだけでファイル選択のウィンドウが扱える
  - 「開く」をクリックしたときのイベントがあるので、そこで読み込み処理を実施
  - メンバの FileName に選択したファイル名が格納される

- ファイルからの読み込みには StremReader を利用
  - 利用するためにはプログラムの最初に using System.IO; が必要
  - StreamReader sr = new StreamReader(ファイル名, Encoding.Default);
    で sr を Stream として利用可能に
  - メンバの ReadLine() で 1 行ごとの文字列を取り出すことができる
  - StringReader だと文字列を Stream のように扱うことができる

- 他のウィンドウを開く
  - 「プロジェクト」→「Windows フォームの追加」で新しいフォームを追加
  - これに追加するさえるのは クラスとしてのフォーム
  - 呼び出すフォームの中で、このフォームのインスタンスを生成して、表示しないと
    新しいウィンドウが表示されない
  - 呼び出し元のインスタンスが参照できないので、追加したフォームに
    public なメンバを追加し、これを用いてデータのやり取りを行う。

- 今回使ったコントロール
  - TextBox コントロール
  - Button コントロール
  - MenuStrip コントロール（メニューを管理する）
  - OpenFileDialog コントロール（ファイルを選択して開く）**<- NEW!**

１５ 追加課題

1 星座を表示した時、ウィンドウの大きさを自動的に調整するようにする。

2 入力するテキストファイルに線の色、丸の大きさや色のカラムを増やし、線の色や丸の大きさを変更することのできるようにする

3 テキストファイルをテキストボックス tbPosition へドラッグ＆ドロップすることで、ファイルを開いて読み込めるようにする。