2023 年 5 月 25 日（木）実施

プログラムの制御構造

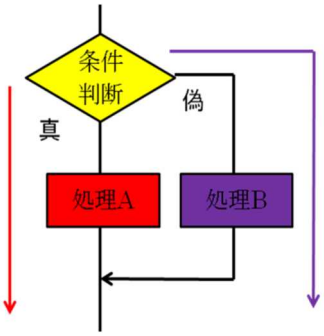
1960 年代後半にダイクストラが提唱した**構造化プログラミング**という考え方では，手続き型のプログラムを記述する際には，**順次**，**選択**，**反復**という標準的な**制御構造**のみを用い，先ずプログラムの概略構造を設計し，その大まかな単位を段階的に詳細化して処理を記述していくものであるが，C#言語で Windows フォームアプリケーションを作成する場合にも，個々のイベントハンドラに記述される処理にはこの考え方が適用出来る。

順次構造

順次構造とは，プログラムの文を記述した順に処理していく構造である。これまで扱ったプログラムは，全て順次構造によって記述されたものであり，最も基本的な制御構造と言える。

プログラムの処理の流れを図示する手法の一つに**流れ図**がある。この流れ図で順次構造を表すと右図の様になる。(色矢印は処理の流れを補足)

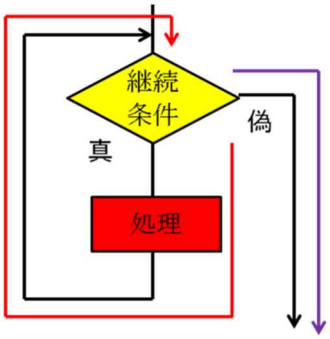
選択構造



選択構造とは，条件や式の値によってプログラムの処理の流れを分ける構造である。選択構造の基本は 2 分岐と呼ばれる構造で，この構造を流れ図で表すと右図の様になる。

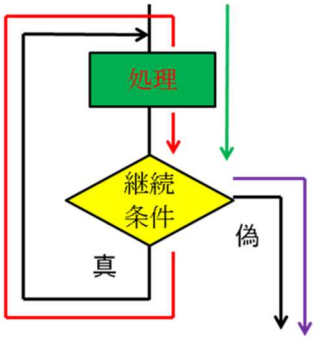
また，式の値によって，幾つもの異なる処理が必要なときには，**多分岐**という選択構造も利用可能である。

反復構造



反復構造とは，**継続条件**が満たされている間，定められた範囲内の文に記述された処理を繰り返して実行する構造である。(C#言語以外のプログラム言語では，**終了条件**が満たされない間，文を繰り返して実行する構造を持つものもある)

なお，反復構造には，右の流れ図で表される 2 種類の場合がある。



1）0 回以上の繰り返し（右上図）

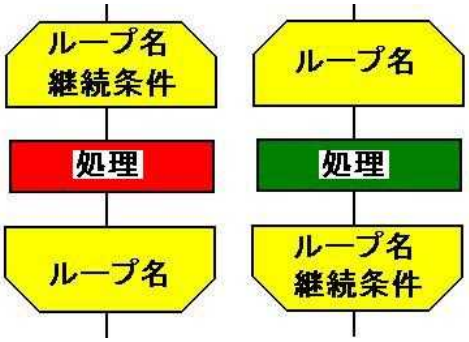
先ず継続条件が判定され，真であれば定められた範囲内の文に記述された処理を実行する。始めから継続条件が満たされない場合には，文は全く実行されないため，0 回以上の繰り返しと呼ばれる。

2）1 回以上の繰り返し（右下図）

先ず定められた範囲内の文に記述された処理が実行され，その後に継続条件が判定される。始めから継続条件が満たされない場合でも，最初の 1 回は定められた範囲内の文に記述された処理が実行されるため，1 回以上の繰り返しと呼ばれる。

＊ 反復構造に対する流れ図に表れる閉線図形をループと呼ぶことから，反復構造のプログラム構造をループと称することがある。

複雑な処理は，順次構造，選択構造，反復構造の組み合わせで実現される。プログラムの構造は，最も大掴みにした概略構造で見ると順次構造となる。選択構造や反復構造を中間の概略構造と看做した場合，その詳細構造として，それぞれの構造の流れ図で『処理』と書かれた箇所に，選択構造や反復構造を埋め込んだ構造も可能である。そこで，反復構造の開始位置と終了位置とを『ループ端』によって明示し，構造を見易くした右のような流れ図も利用される。



2 分岐のプログラム

if 文

C#言語で 2 分岐のプログラムを実現するための文として，if 文が用意されている。if 文の構文は次の様になる。

```
if ( 条件式 ) { 真文 } [ else { 偽文 } ]
```

{}は複数の文のブロック

ここで，[]内は省略可能であり，省略時は条件式が偽のときは何もしない場合を表す。また，真文または偽文は複数の文で構成することが出来，単独の文の場合は{}の括弧を省略可能である。

条件式

条件式では，条件が満たされる（真となる）場合には，値が true となり，条件が満たされない（偽となる）場合には，値が false となる。true 及び false は論理型（型名は bool）のリテラルである。(bool 型に関しては，第 3 回教材の p.1 参照)

条件式で用いられる**関係演算子**及び**等値演算子**を次に挙げる。

関係演算子	書式	意味
<	x < y	x が y より小さければ true，それ以外は false
>	x > y	x が y より大きければ true，それ以外は false
<=	x <= y	x が y より小さいか，両者が等しければ true，それ以外は false
>=	x >= y	x が y より大きいか，両者が等しければ true，それ以外は false

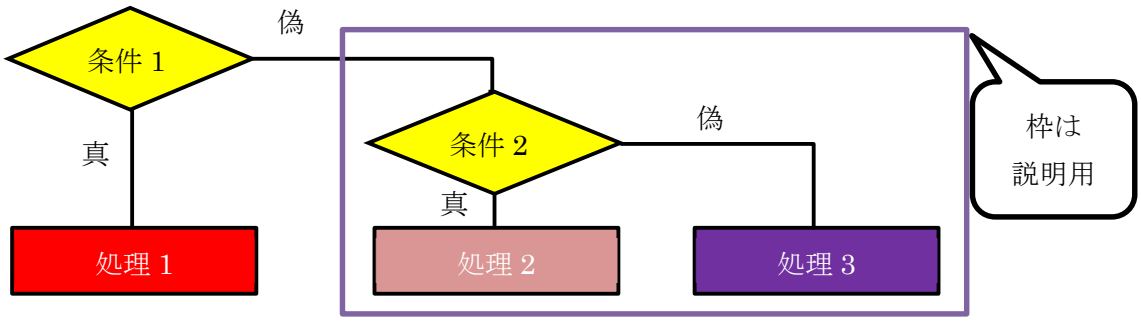
等値演算子	書式	意味
==	x == y	x と y とが等しければ true，それ以外は false
!=	x != y	x と y とが等しくなければ true，それ以外は false

また，複数の条件式を組み合わせるために用いられる**論理演算子（論理否定演算子，条件 AND 演算子**及び**条件 OR 演算子**）を次に挙げる。なお，論理否定演算子は第 4 回教材の p.1 に登場した単項演算子の一つであり，条件 AND 演算子及び条件 OR 演算子は加法演算子よりも優先順位が低い。

論理演算子	書式	意味
!	!式	式が true であれば false，式が false であれば true
&&	式 1 && 式 2	式 1 及び式 2 が共に true であれば true，それ以外は false
||	式 1 || 式 2	式 1 または式 2 のどちらか一方が true であれば true（両者が true の場合も含む），それ以外は false

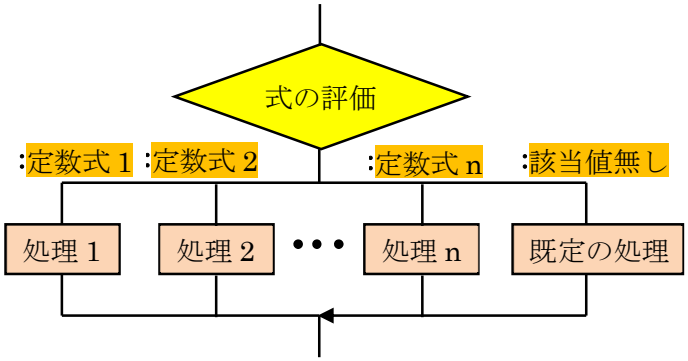
入れ子の if 文

if 文では，真文または偽文として if 文を用いることが可能である。これを"**入れ子（ネスト）の if 文**"と呼ぶ。if 文は 2 分岐であるが，入れ子の if 文を用いることによって，3 以上の分岐**(多分岐)**を表現することが可能となる。次の図は，偽文として if 文を用いた入れ子の if 文の流れ図である。



多分岐のプログラム

前節では多段階の 2 分岐を組み合わせて 3 種類以上の場合分けを実現したが，式の値の評価によって一度に多種類の場合分けを行う**多分岐**の利用によって見通しのよいプログラムを作成出来る場合がある。（流れ図は右図）



switch 文

C#言語で多分岐のプログラムを実現するための文として，switch 文が用意されている。switch 文の構文は次の様になる。

```
switch ( 制御式 ) {
    case 定数式 1: [[文 1-1 [文 1-2] ・・・ ] break;]
```

```
        case 定数式 2: [[文 2-1 [文 2-2] ・・・ ] break;]
        ・・・
        case 定数式 n: [[文 n-1 [文 n-2] ・・・ ] break;]
        [default: [既定文 1 [既定文 2] ・・・] break;]
    }
```

ここで，[]内は省略可能である。switch 文の機能は，制御式を評価してその値が定数式（例えば，10，'a'の様なリテラルや 1+2+3 の様な値の定まった式）の何れかの値に一致したとき，その case ラベルに続く文を break 文に出会うまで実行する。break 文に到達すると switch 文から抜け出す。もし，一致した case ラベルに続く文及び break 文が省略されている場合には，その下の case ラベルに続く文を break 文に出会うまで実行する。

また，switch 文では，制御式を評価してその値が定数式の何れの値にも一致しないときは，default ラベルが書かれていれば，それに続く文を実行する。

Java 言語との違いは，default ラベルにも break 文を伴うこと，及び case ラベルに続く文がある時に break 文を省略して次の case ラベルに続く文も実行するという**フォールスルー**が禁じられていることである。

本日の課題

今回の授業では，選択構造（2 分岐，多分岐）の使い方について学ぶ。

手順

1）コントロールの追加

JimboCalc.sln を開く。フォームデザイナーで，直前の入力を取り消す（**クリアエントリー**）ボタンを追加する。プロパティは次の通りである（Text プロパティの値は半角文字で与える）。

【ボタンのプロパティ】

button18	
Width	40
Height	35
Text	CE
X	72
Y	153

2) コーディング

今回は，演算子のボタンが押されたら変数 num2 に代入すべき数値の元となる，数字を並べた文字列を新たに受け付ける様に改良する。そのためには，先ず，演算子のボタンが押されたかどうかという状態を判定するためにフラグとして用いる int 型の変数 flag を宣言して，初期値を 0 としておく。この変数の値を演算子のボタンが押された際に 1 に変える（これを『**フラグを立てる**』と言う）。また，どの演算子のボタンが押されたのかを記録するための char 型の変数 type を宣言して，加算，減算，乗算，除算のそれぞれについて，'a'～'d' の値を割り当てる。

更に，クリアエントリーボタンを機能させるために，直前の数字を並べた文字列，数式，フラグを残しておくための変数，predeg，preform，preflag をそれぞれ宣言する。

なお, フラグが立っているかどうかの条件判定には if 文を用い, 演算子の種類の判定には switch 文を用いる。

【手順】

先ず，上述の 5 個の変数をクラスのフィールドに追加して宣言する。

```
13      3 個の参照
        public partial class Form1 : Form
14      {
15          string degits = "";
16          string formula = "";
17          double num1;
18          double num2;
19          double result;
20
21          int flag = 0;
22          char type;
23          string predeg = "";
24          string preform = "";
25          int preflag = 0;
26
27          1 個の参照
            public Form1()
28          {
```

num2 のデータ型を変更

次に，button1～button11 のイベントハンドラに直前の状態を残しておくための代入文を追加する。

```
32          1 個の参照
            private void button1_Click(object sender, EventArgs e)
33          {
34              predeg = degits;
35              preform = formula;
36              preflag = flag;
37
38              degits += button1.Text;
39              formula += button1.Text;
40              label2.Text = degits;
41              label1.Text = formula;
42          }
```

続けて，演算子のボタン button12～button15 のイベントハンドラの処理の記述を，if 文を用いて書き換える。ここでは，num1 には数字を並べた文字列を解析して，数値化したものを代入

しているが，num2 用の数字を並べた文字列の入力を受け付けるために，degits に空の文字列を代入している。また，フラグを立て（flag = 1;），演算子の種類を設定している（type = 'a'; 等）。button13～button15 に対する type の値に関しては，この節の冒頭の説明の通りである。

```
164         private void button12_Click(object sender, EventArgs e)
165         {
166             if (flag == 0)
167             {
168                 predeg = degits;
169                 preform = formula;
170                 preflag = flag;
171 
172                 num1 = double.Parse(degits);
173                 formula += button12.Text;
174                 label2.Text = "";
175                 label1.Text = formula;
176                 degits = "";
177                 flag = 1;
178                 type = 'a';
179             }
180         }
181 
182         private void button13_Click(object sender, EventArgs e)
183         {
184             if (flag == 0)
185             {
186                 predeg = degits;
187                 preform = formula;
188                 preflag = flag;
189 
190                 num1 = double.Parse(degits);
191                 formula += button13.Text;
192                 label2.Text = "";
193                 label1.Text = formula;
194                 degits = "";
195                 flag = 1;
196                 type = 'b';
197             }
198         }
```

```
200         private void button14_Click(object sender, EventArgs e)
201         {
202             if (flag == 0)
203             {
204                 predeg = degits;
205                 preform = formula;
206                 preflag = flag;
207 
208                 num1 = double.Parse(degits);
209                 formula += button14.Text;
210                 label2.Text = "";
211                 label1.Text = formula;
212                 degits = "";
213                 flag = 1;
214                 type = 'c';
215             }
216         }
```

```
218     private void button15_Click(object sender, EventArgs e)
219     {
220         if (flag == 0)
221         {
222             predeg = degits;
223             preform = formula;
224             preflag = flag;
225
226             num1 = double.Parse(degits);
227             formula += button15.Text;
228             label2.Text = "";
229             label1.Text = formula;
230             degits = "";
231             flag = 1;
232             type = 'd';
233         }
234     }
```

結果表示のボタン button16 のイベントハンドラでは，次の様な記述を加える。フラグが立っている場合には num2 に数字を並べた文字列を解析して，数値化したものを代入し，switch 文で演算子の種類を判定して，それぞれの演算子に対応した結果を result に代入する。また，フラグが立っていない場合には数字を並べた文字列を解析して，数値化したものを result に代入する。

```
        1 個の参照
236     private void button16_Click(object sender, EventArgs e)
237     {
238         if (flag == 1)
239         {
240             num2 = double.Parse(degits);
241
242             switch (type)
243             {
244                 case 'a': result = num1 + num2; break;
245                 case 'b': result = num1 - num2; break;
246                 case 'c': result = num1 * num2; break;
247                 case 'd': result = num1 / num2; break;
248             }
249             degits = "";
250             formula = "";
251             flag = 0;
252         }
253         else
254         {
255             result = double.Parse(degits);
256         }
257
258         label2.Text = result.ToString();
259     }
```

クリアボタン button17 のイベントハンドラでは，4 個の変数の値を初期値に戻す文を加える。

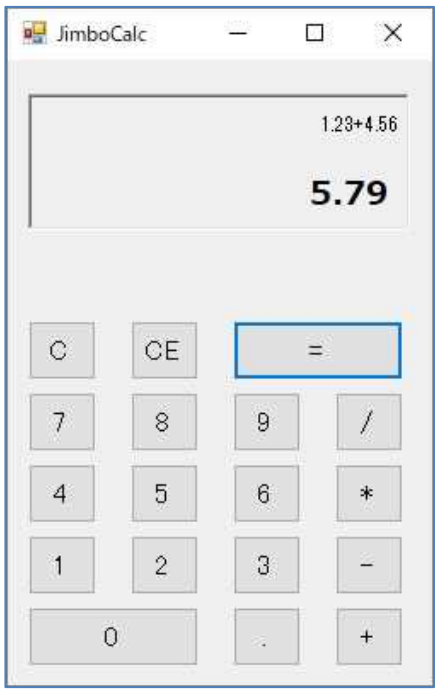
（図は次のページ）

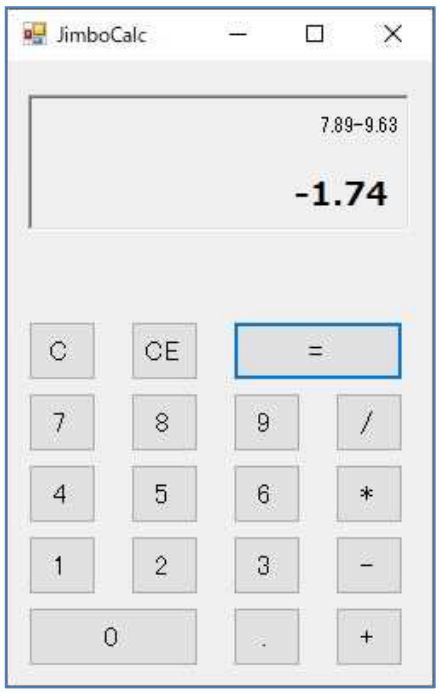
```
260
            1 個の参照
261         private void button17_Click(object sender, EventArgs e)
262         {
263             predeg = "";
264             preform = "";
265             preflag = 0;
266             flag = 0;
267
268             degits = "";
269             formula = "";
270             label1.Text = "";
271             label2.Text = "";
272         }
```

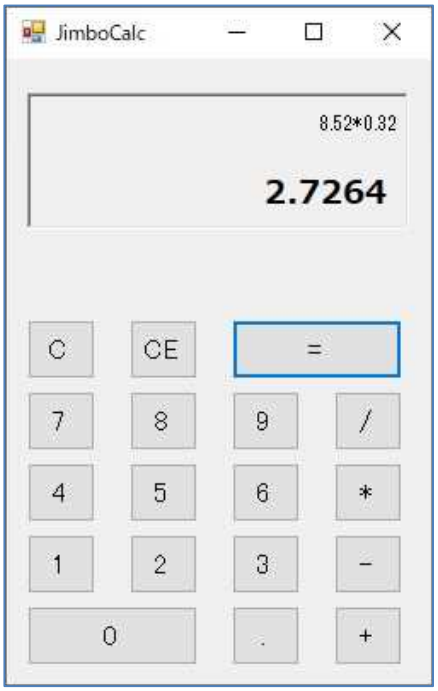
更に，フォームデザイナーに戻って，クリアエントリーのための『CE』と書かれた button18 をダブルクリックし，イベントハンドラを作成し，処理を記述する。

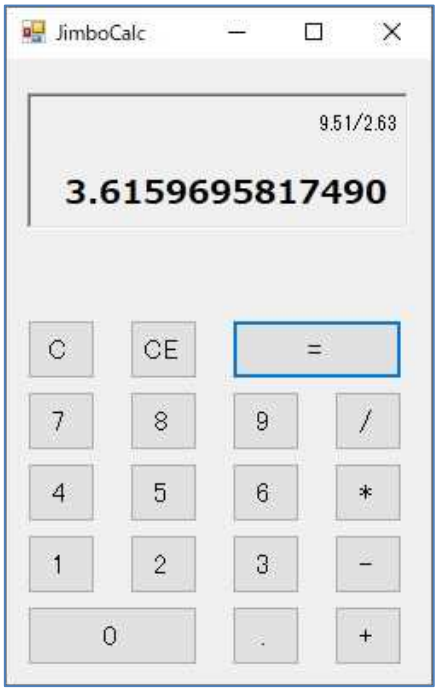
```
279
            1 個の参照
280         private void button18_Click(object sender, EventArgs e)
281         {
282             degits = predeg;
283             formula = preform;
284             flag = preflag;
285             label2.Text = degits;
286             label1.Text = formula;
287         }
288     }
289 }
```

ここで，『Form1.cs の保存』ボタンを押してから，『開始』ボタンを押して，プログラムを実行して，四則演算（加算，減算，乗算，除算）及び CE のそれぞれについて，動作を確かめる。









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) コードエディタで編集したソースファイル Form1.cs をメールに添付して提出する。
3) 実行の加算の結果のスクリーンショット第 5 回加算実行結果.jpg（.png も可）をメールに添付して提出する。
4) 実行の減算の結果のスクリーンショット第 5 回減算実行結果.jpg（.png も可）をメールに添付して提出する。
5) 実行の乗算の結果のスクリーンショット第 5 回乗算実行結果.jpg（.png も可）をメールに添付して提出する。
6) 実行の除算の結果のスクリーンショット第 5 回除算実行結果.jpg（.png も可）をメールに添付して提出する。
7) 質問を記述したファイル Prog2_Questions_5th.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。