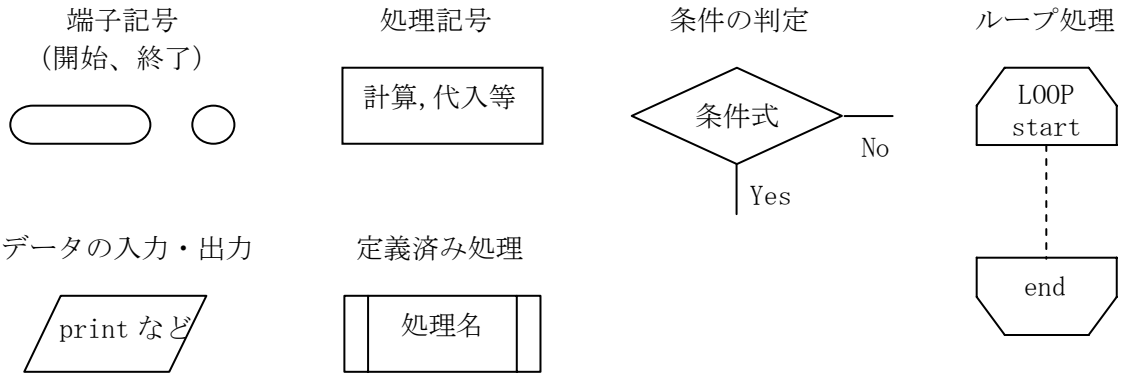
フローチャート（流れ図）

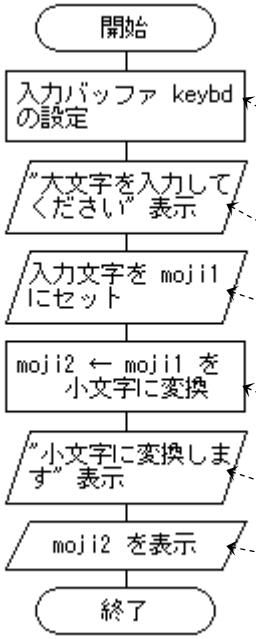
プログラムの処理手順（アルゴリズム）を図示したもの。記号の種類は下記のとおり。



◇サンプルグログラム
（大文字 → 小文字変換）

（実行画面）

```
大文字を入力して下さい → E

小文字に変換します → e
-- Press any key to exit (Input "c" to continue) --
```



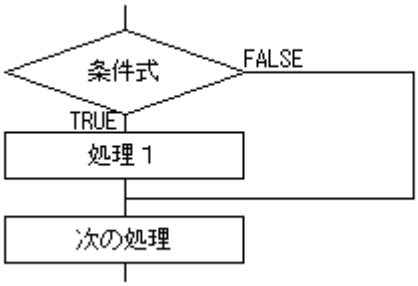

```
//  大文字から小文字への変換
import java.io.*;
class Flowchart_sample_moji_henkan {
   public static void main(String[] args) throws Exception
   {
      char moji1,moji2;

      BufferedReader inp=new BufferedReader
        (new InputStreamReader(System.in));
      String keybd;

      System.out.print("大文字を入力してください → ");
      keybd=inp.readLine();
      moji1=keybd.charAt(0);

      moji2=Character.toLowerCase(moji1);

      System.out.print("¥n 小文字に変換します → ");
      System.out.print(moji2+"¥n");
   }
}
```

◇補足説明

- 変数の宣言は、フローチャートに書かなくてもよい（char moji1 など）。ただし初期化など、値を代入する場合は、書くようにする。
- プログラムの１行が、必ずしも記号１つに対応するとは限らない。
- アルゴリズム上あまり重要ではない処理は、流れ図を省略してもよい（単なる改行の¥n など）。

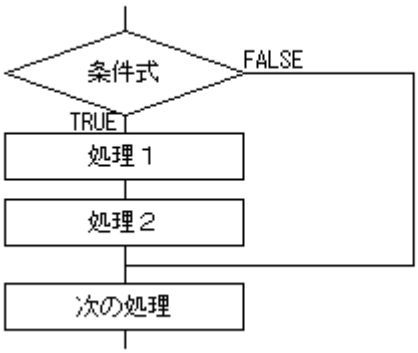
if 文

①単純 if 型（もし～ならば、○○○を行う）




```
if（条件式）
　　処理１；
次の処理；
```

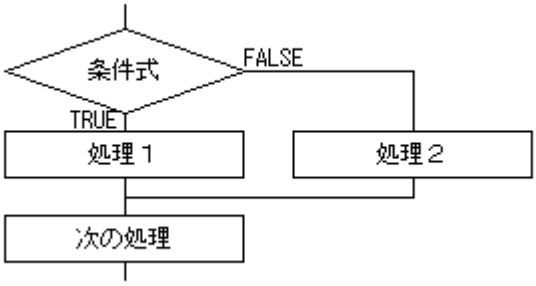
・条件式のあとに；はつけない
・if の中の処理はインデントすると見やすい




```
if（条件式）　{
　　処理１；
　　処理２；
}
次の処理；
```

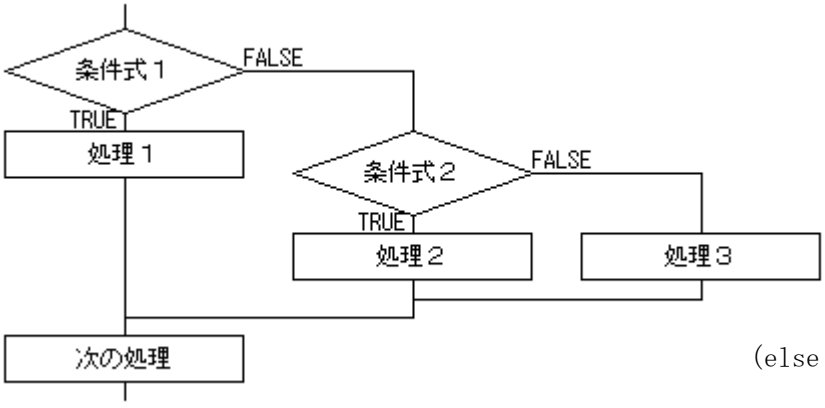
・処理が２つ以上ある時は、{　} を
　つけてブロックにする

②if-else 型（もし～ならば○○○を行い、そうでないときは×××を行う）




```
if（条件式）
　　処理１；
else
　　処理２；
次の処理；
```

③if-else if 型（else のあとに、さらに if を続けて書くことができる）




```
if（条件式１）
　　処理１；
else if（条件式２）
　　処理２；
else
　　処理３；
次の処理；
```

（else if は続けて何個でも書ける）

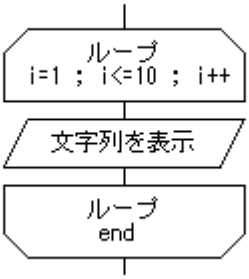
繰り返し型（その１）for 文

```
for ( 初期値 ; 繰り返し条件 ; 増分値 ) {
        処理の内容
}
```

初期値　　→ ループの前処理
繰り返し条件 → ループを継続する条件
増分値　　→ １回のループの後処理

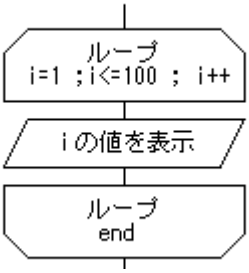
（例１）同じ文字列を 10 個表示するプログラム

```
for ( i=1 ; i<=10 ; i++ ) {
    System.out.print("任意の文字列 ¥n");
}
```



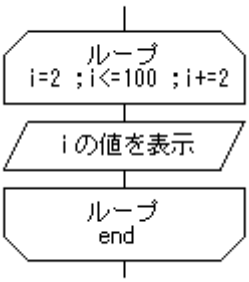
（例２）１から１００までの数を表示するプログラム

```
for ( i=1 ; i<=100 ; i++ ) {
    System.out.print(i+"  ");
}
```



（例３）２から１００までの偶数を表示するプログラム

```
for ( i=2 ; i<=100 ; i+=2 ) {
    System.out.print(i+"  ");
}
```



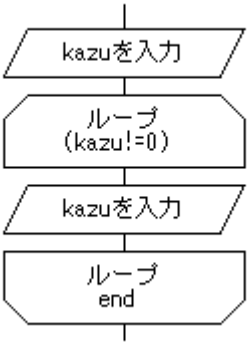
繰り返し型（その２）while 文

```
while ( 繰り返し条件 ) {
        処理の内容
}
```

繰り返し条件 → ループを継続する条件

（例４）入力を繰り返すプログラム（０で終了）

```
System.out.print("数を入れて下さい");
keybd=inp.readLine();
kazu=Integer.parseInt(keybd);
while ( kazu!=0 ) {
    System.out.print("数を入れて下さい");
    keybd=inp.readLine();
    kazu=Integer.parseInt(keybd);
}
```



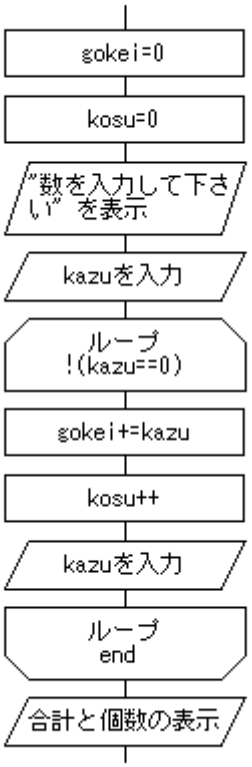
この条件は、while (!(kazu==0)) と書いても同じ。(意味は「０になるまで繰り返す」)

繰り返し型の応用例

（1）数を連続して入力し、「合計」と「個数」を表示するプログラム（0を入力したら終了)。

（実行画面）

```
数を入力して下さい（0で終了）
-->20
-->10
-->30
-->3
-->0

合計＝ 63
個数＝ 4
```

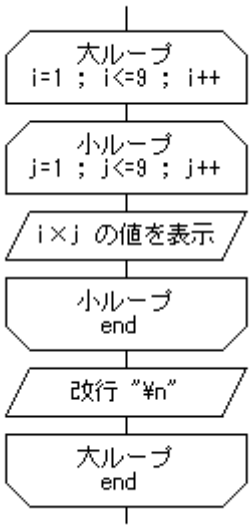


（while 文を使用する）

（2）二重ループ　（九九の表示）

（実行画面）

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81



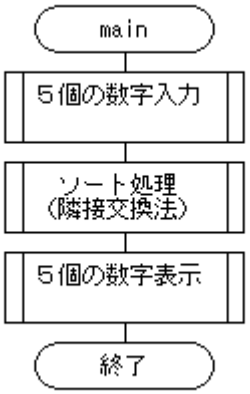
（for 文を二重に使用する）

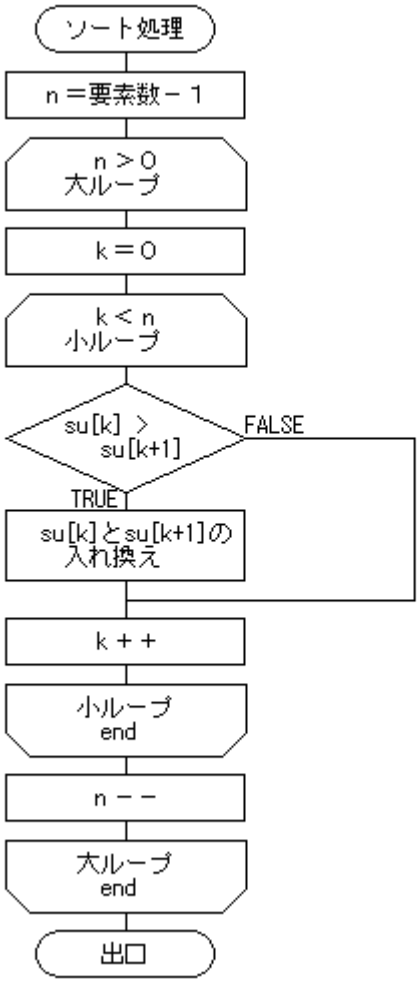
```
for (i=1 ; i<=9 ; i++) {
    for( j=1 ; j<=9 ; j++ ) {
        System.out.print("  "+i*j);
    }
    System.out.print("¥n");
}
```

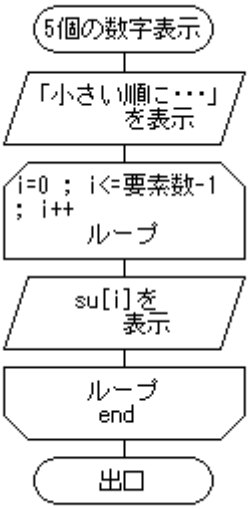
大ループ（ｉが１～９まで変化）

小ループ（ｊが１～９まで変化）

ソート処理　（１）隣接交換法







（実行画面）

５個の数字をソートします
NO.1 -->90
NO.2 -->60
NO.3 -->20
NO.4 -->40
NO.5 -->10

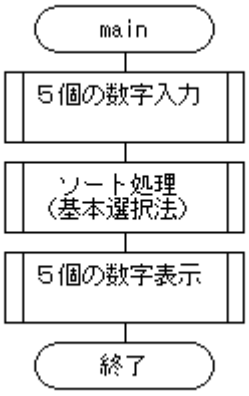
小さい順に表示します
10　20　40　60　90

ソート処理の手順　（隣接交換法）

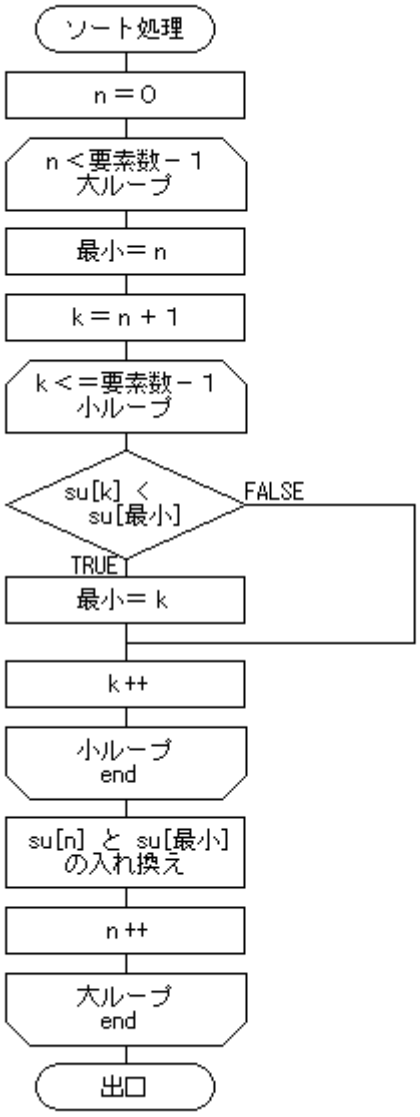
①配列の１番目の要素と２番目の要素を比較する。
②１番目の要素の方が大きければ、両者を交換する。
③要素の２番目と３番目を比較して、同様の処理を行う。
④以上を、配列の最後の要素まで繰り返す。
　（この段階で、配列の右端が最大値となる ---> 確定）
⑤上記の①～④までを、配列の最後から１つ手前の要素まで行う。

以上を繰り返し、２番目の要素が確定するまで（すなわち全要素が確定するまで）行う。

ソート処理　（２）基本選択法



「５個の数字入力」
「５個の数字表示」
　　は隣接交換法と同じ。



（実行画面）

```
５個の数字をソートします
NO.1 -->60
NO.2 -->50
NO.3 -->10
NO.4 -->80
NO.5 -->30

小さい順に表示します
  10   30   50   60   80
```

ソート処理の手順　（基本選択法）

①すべての配列の要素の中から、最小値の要素を見つける。
②１番目の要素と見つけた最小値の要素を交換する。（１番左の要素が確定）
③２番目の要素から最終要素までの中から、最小値を見つける。
④２番目の要素と見つけた最小値の要素を交換する。（左から２番目が確定）
　以上を、最後のデータの１つ前まで繰り返す。
　　↓
　並べ替え終了。