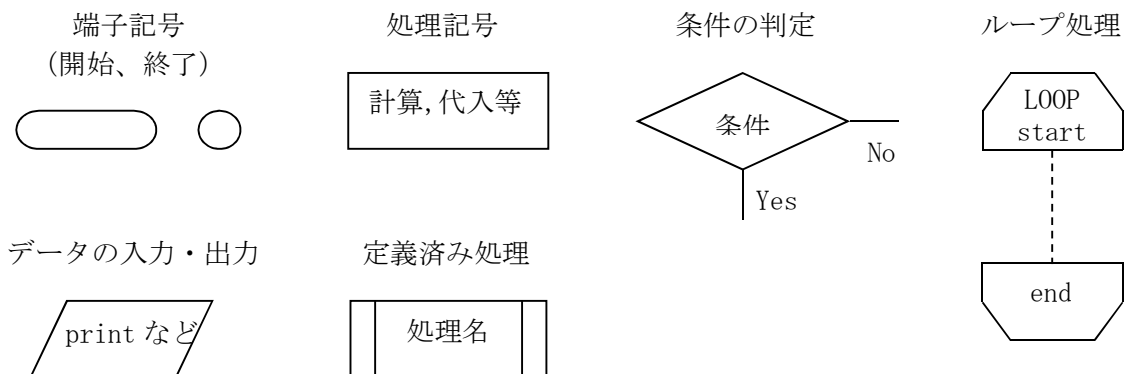


### フローチャート（流れ図）

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。

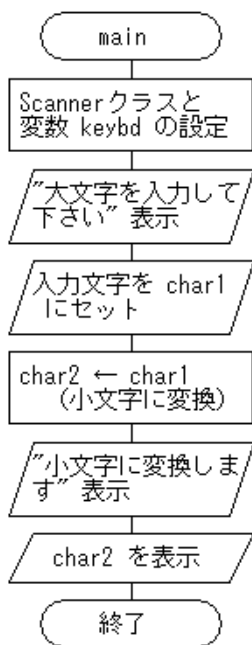


◇サンプルプログラム  
(大文字 → 小文字変換)

(コンソール画面)

大文字を入力して下さい → E

小文字に変換します → e



```
// 大文字から小文字への変換
import java.util.*;

class FlowchartSampleChar {
    public static void main(String[] args)
    {
        char char1, char2;

        Scanner sc = new Scanner(System.in);
        String keybd;

        System.out.print("大文字を入力してください → ");
        keybd=sc.nextLine();
        char1=keybd.charAt(0);

        char2=Character.toLowerCase(char1);

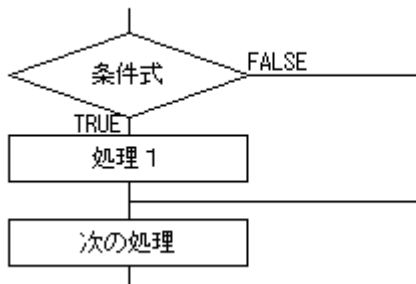
        System.out.print("\n小文字に変換します → ");
        System.out.print(char2+"\n");
    }
}
```

◇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 char1 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)。

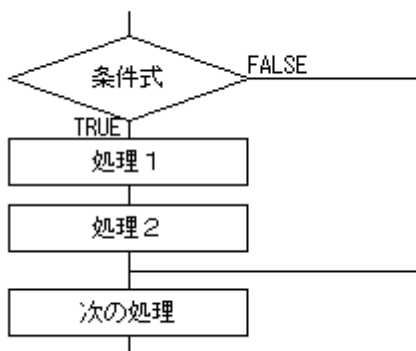
## if 文

①単純 if 型 (もし~ならば、〇〇〇を行う)



```
if (条件式)
    処理 1 ;
次の処理 ;
```

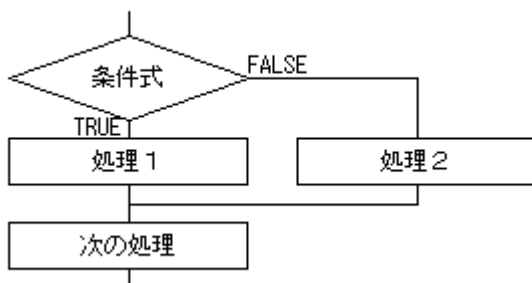
- ・条件式のあとに ; はつけない
- ・if 中の処理はインデントすると見やすい



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

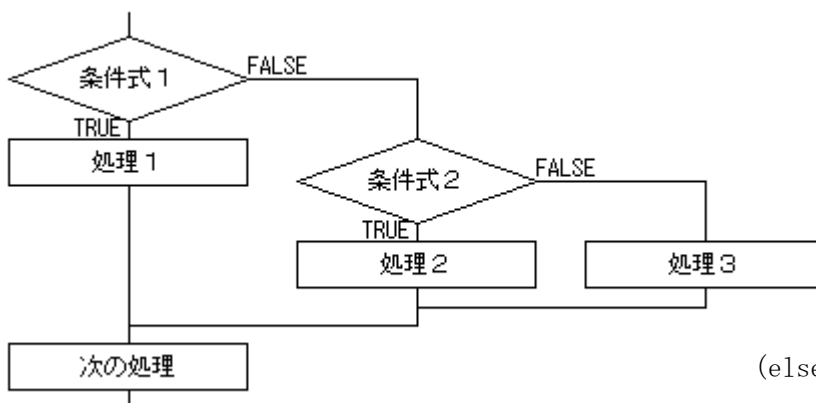
- ・処理が 2 つ以上ある時は、{ } をつけてブロックにする

②if-else 型 (もし~ならば〇〇〇を行い、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)

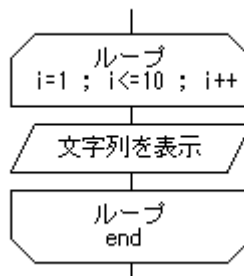
繰り返し型 (その1) for 文

```
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

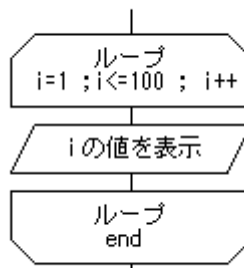
(例1) 同じ文字列を10個表示するプログラム

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



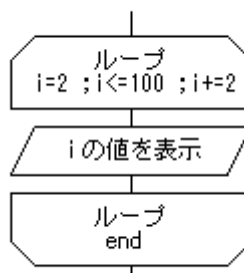
(例2) 1から100までの数を表示するプログラム

```
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

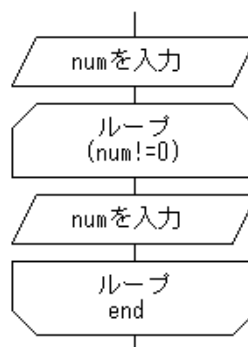
```
while ( 繰り返し条件 ) {
    処理の内容
}
```

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-->");
num=sc.nextInt();

while ( num!=0 ){
    System.out.print("数を入れて下さい-->");
    num=sc.nextInt();
}
```



この条件は、while ( !(num==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）。

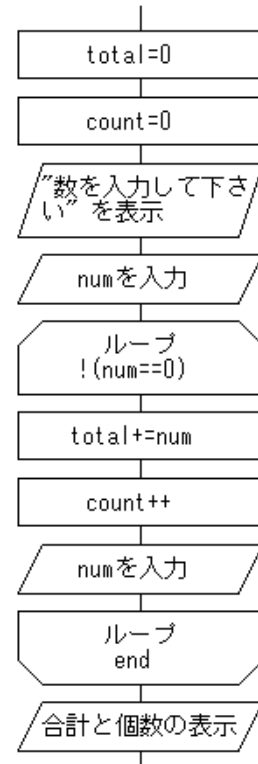
(コンソール画面)

```

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)

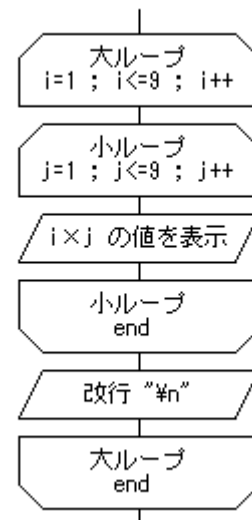


(2) 二重ループ (九九の表示)

(コンソール画面)

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)



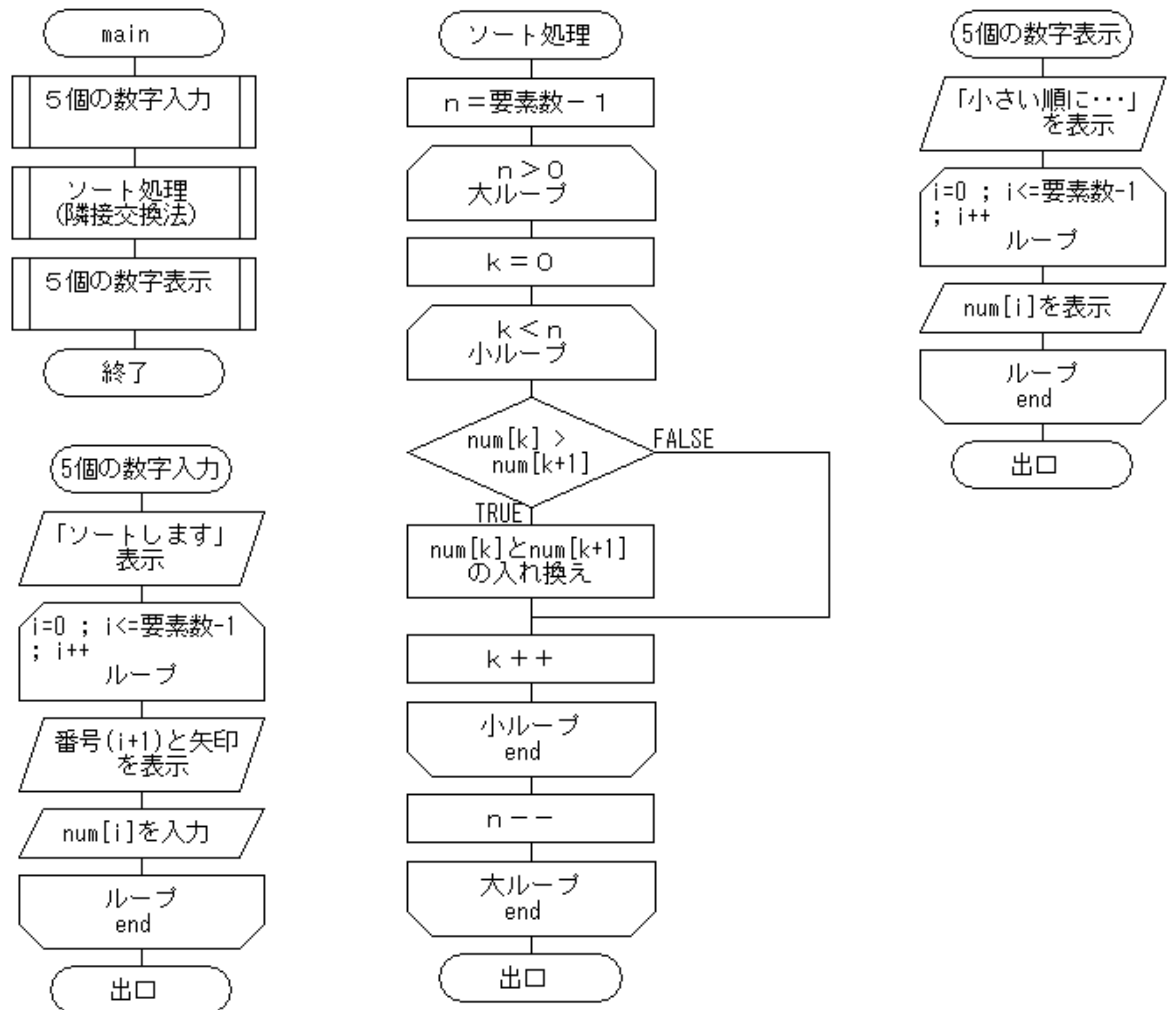
```

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 (Bubble Sort)



## (コンソール画面)

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

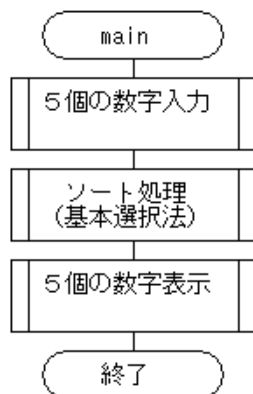
10 20 40 60 90

## ソート処理の手順 (隣接交換法)

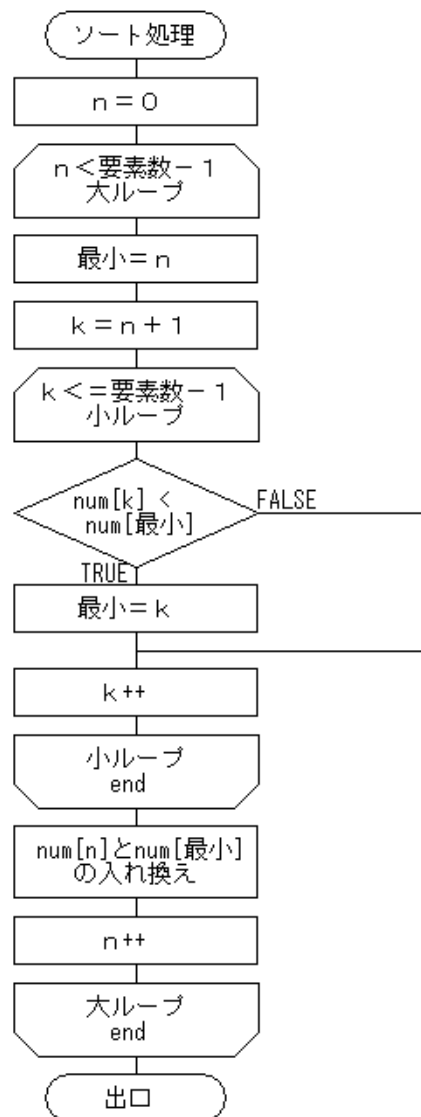
- ①配列の 1 番目の要素と 2 番目の要素を比較する。
- ② 1 番目の要素の方が大きければ、両者を交換する。
- ③要素の 2 番目と 3 番目を比較して、同様の処理を行う。
- ④以上を、配列の最後の要素まで繰り返す。  
(この段階で、配列の右端が最大値となる ---> 確定)
- ⑤上記の①~④までを、配列の最後から 1 つ手前の要素まで行う。

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 (Selection Sort)



「5個の数値入力」  
 「5個の数値表示」  
 は隣接交換法と同じ。



## (コンソール画面)

5 個の数値をソートします

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

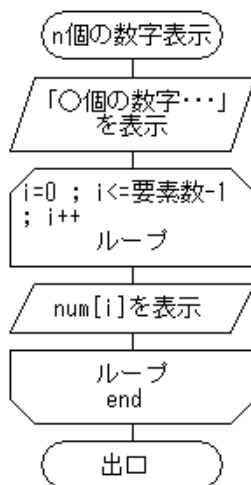
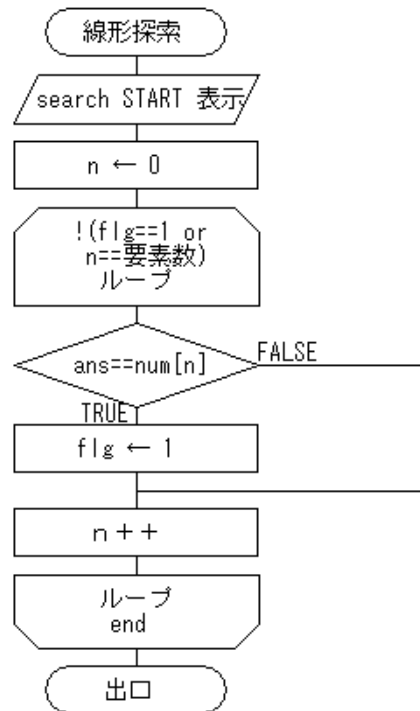
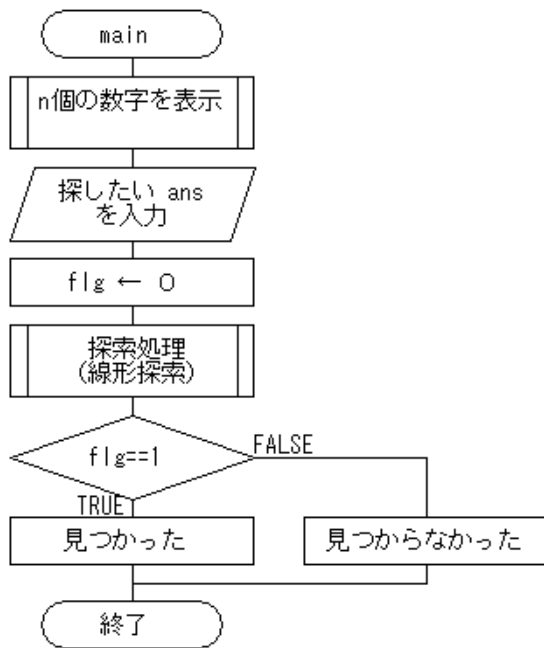
10 30 50 60 80

## 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。

探索処理 (1) 線形探索 (Linear Search)



(実行画面)

10 個の数字を探索します (線形探索 LinearSearch)  
3 25 60 48 13 80 40 12 7 99

探したい値を入れて下さい --> 25

----- Search START -----  
見つかりました

(実行画面)

10 個の数字を探索します (線形探索 LinearSearch)  
3 25 60 48 13 80 40 12 7 99

探したい値を入れて下さい --> 26

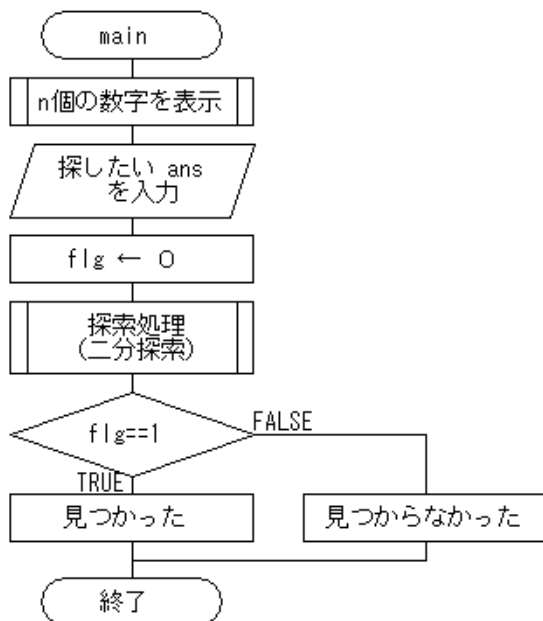
----- Search START -----  
データが見つかりません

線形探索の手順

フラグ(fl g)の意味 : 1 =探索成功 0 =不成功

- ①フラグを0にする。
- ②先頭の要素から、1件ずつ順番に値を比較する。
- ③目的の値(answer)が見つかったら、フラグを1にする。
- ④フラグが1になるか、またはすべての要素の検索が終わるまでループを繰り返す。
- ⑤ループ終了後にフラグを判定する (1ならば探索成功、0ならば不成功)。

探索処理 (2) 二分探索 (Binary Search)



(実行画面)

10 個の数字を探索します (二分探索 BinarySearch)  
 6 9 19 33 45 49 55 65 70 98

探したい値を入れて下さい --> 19

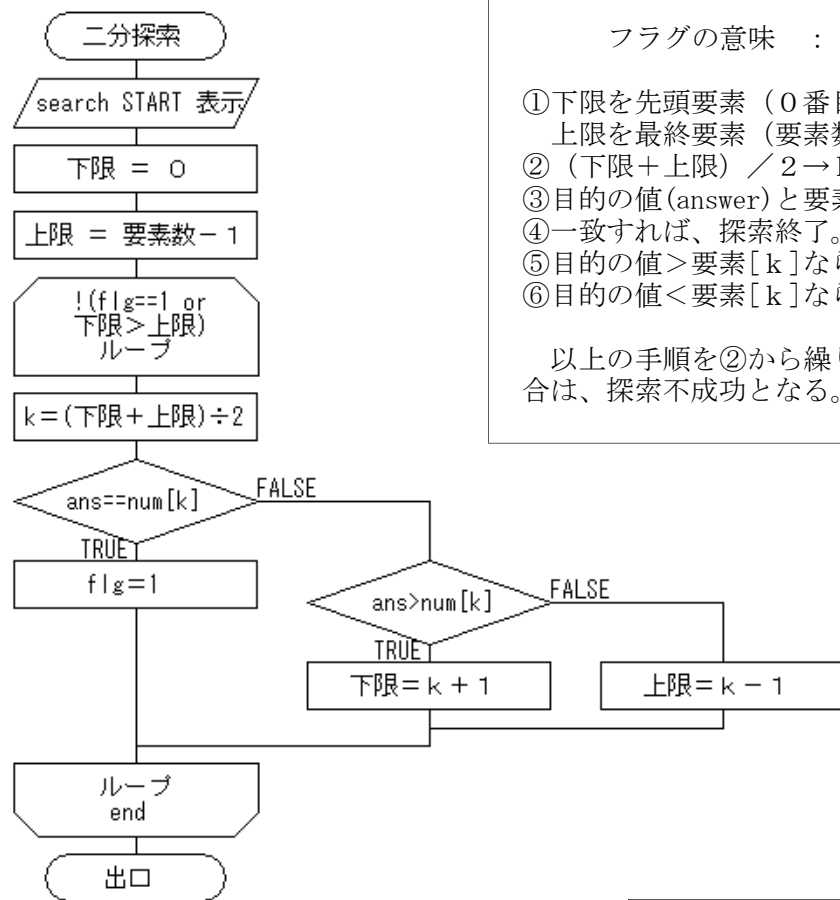
----- Search START -----  
 見つかりました

(実行画面)

10 個の数字を探索します (二分探索 BinarySearch)  
 6 9 19 33 45 49 55 65 70 98

探したい値を入れて下さい --> 20

----- Search START -----  
 データが見つかりません



二分探索の手順

フラグの意味 : 1 = 探索成功 0 = 不成功

- ① 下限を先頭要素 (0 番目)、  
 上限を最終要素 (要素数 - 1 番目) に合わせる。
- ② (下限 + 上限) / 2 → k とする。
- ③ 目的の値 (answer) と要素 [k] を比較する。
- ④ 一致すれば、探索終了。
- ⑤ 目的の値 > 要素 [k] ならば、下限を k + 1 に合わせる。
- ⑥ 目的の値 < 要素 [k] ならば、上限を k - 1 に合わせる。

以上の手順を②から繰り返す。ただし下限 > 上限になった場合は、探索不成功となる。

「n 個の数字表示」のフローチャートは省略