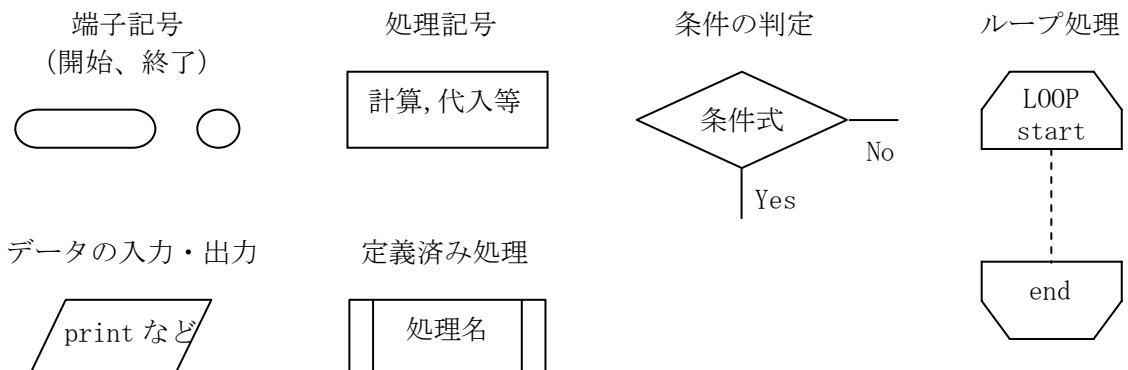
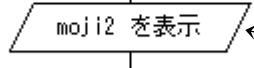
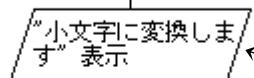
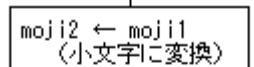
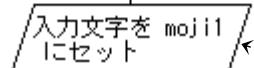
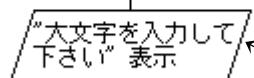
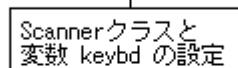
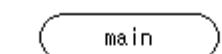
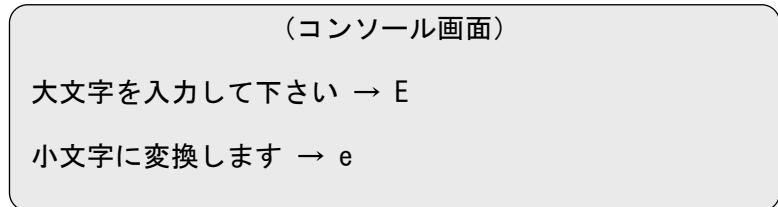


## フローチャート（流れ図）

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)



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

class Flowchart_sample_moji_henkan {
    public static void main(String[] args)
    {
        char moji1, moji2;

        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();
        moji1=keybd. charAt(0);

        moji2=Character. toLowerCase(moji1);

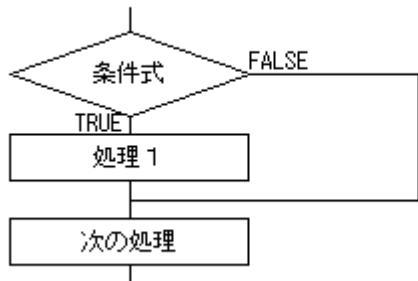
        System. out. print("\n 小文字に変換します → ");
        System. out. print(moji2+"\n");
    }
}
```

## ◇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、書くようとする。
- 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

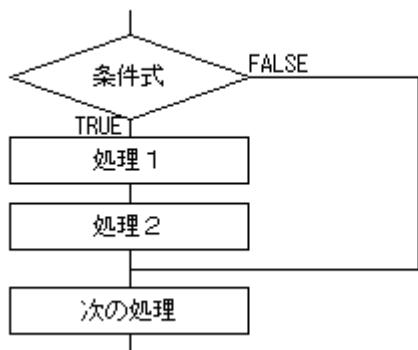
①単純 if 型（もし～ならば、○○○を行う）



```

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

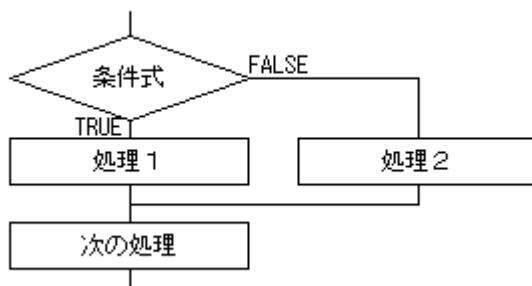


```

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

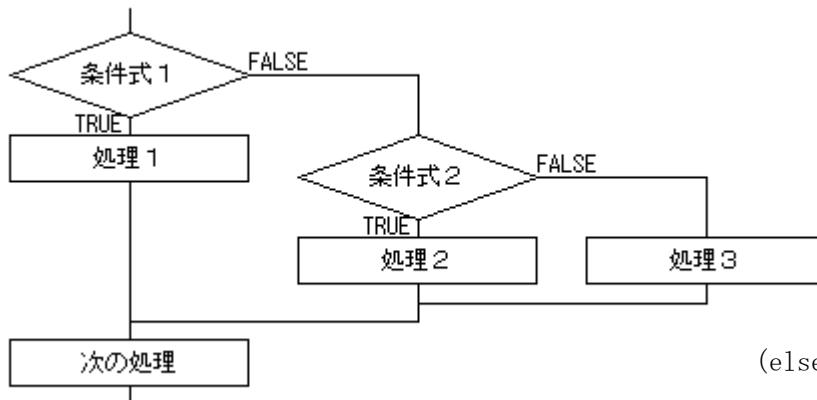
②if-else 型（もし～ならば○○○を行い、そうでないときは×××を行う）



```

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)

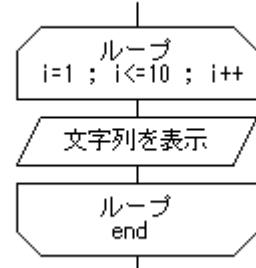
## 繰り返し型（その1）for文

```
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

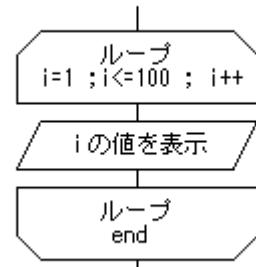
(例1) 同じ文字列を10個表示するプログラム

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



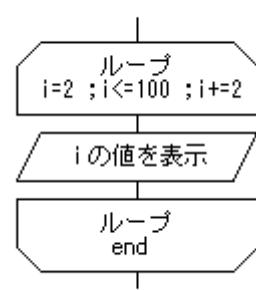
(例2) 1から100までの数を表示するプログラム

```
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

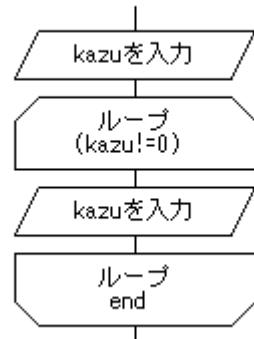
```
while ( 繰り返し条件 ) {
    処理の内容
}
```

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

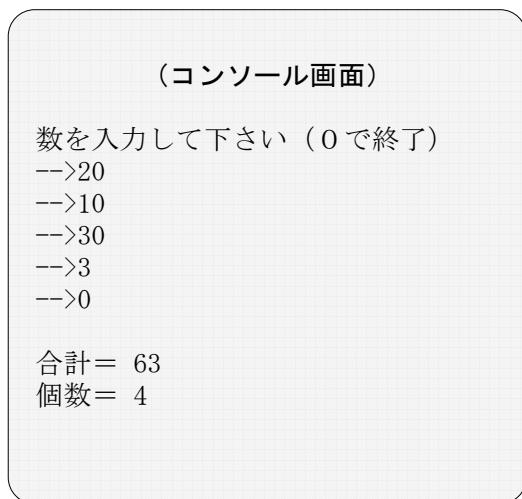
while ( kazu!=0 ){
    System.out.print("数を入れて下さい-->");
    kazu=sc.nextInt();
}
```



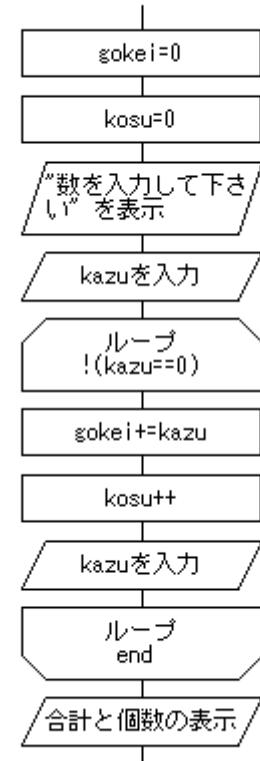
この条件は、while ( !(kazu==0) ) と書いても同じ。（意味は「0になるまで繰り返す」）

## 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）。



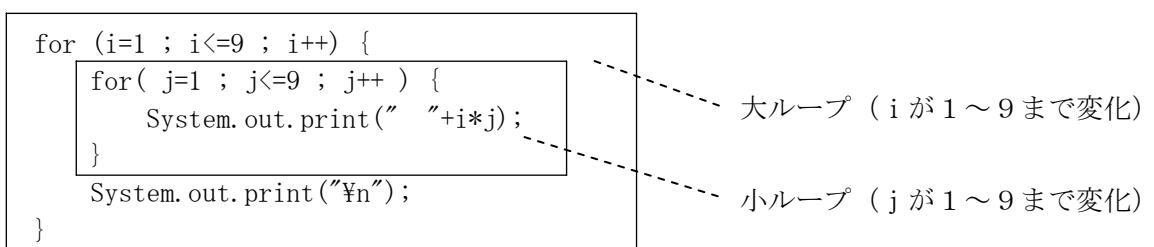
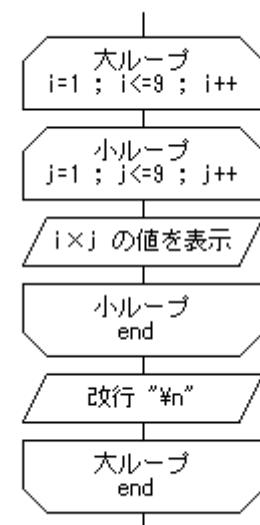
(while文を使用する)



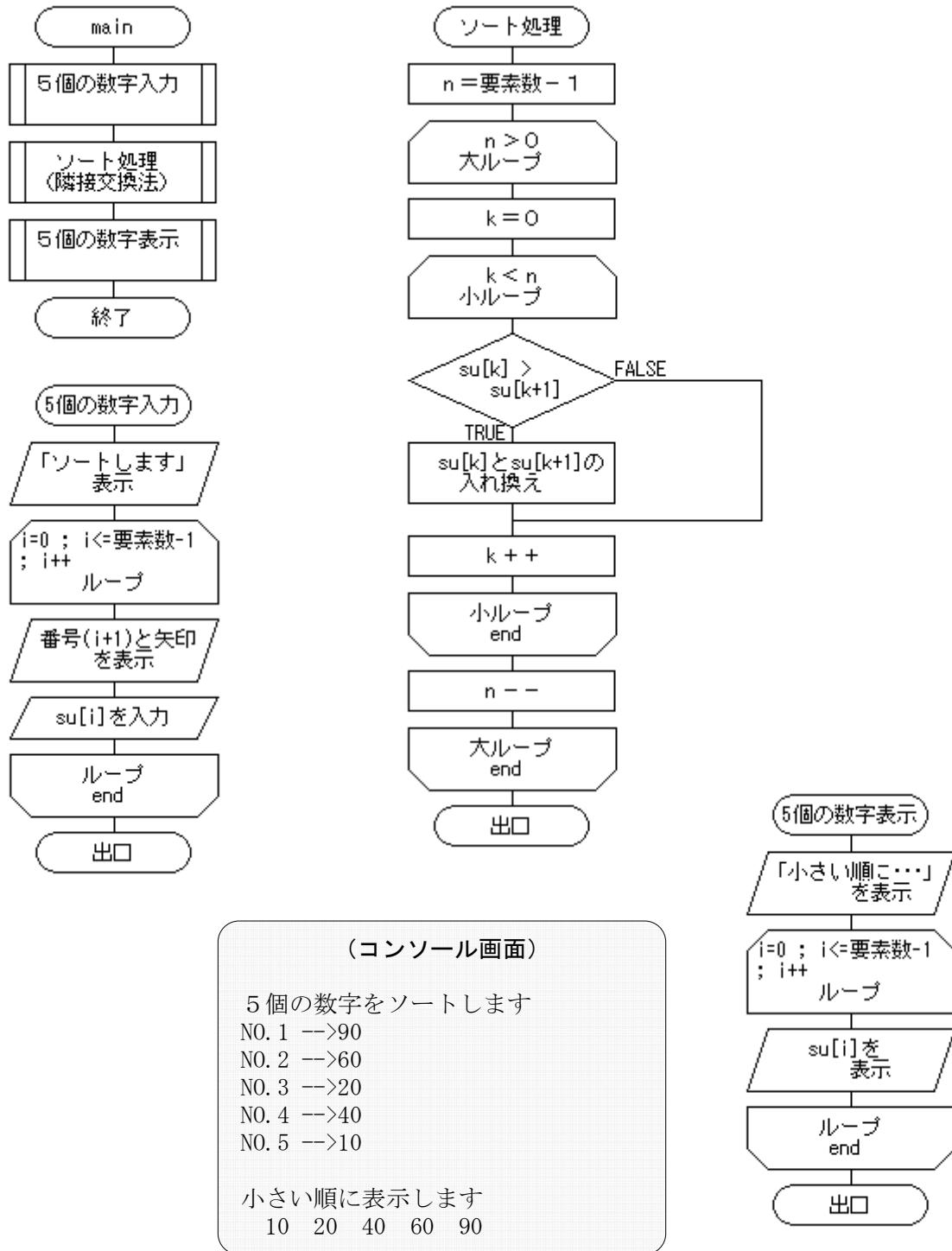
(2) 二重ループ (九九の表示)



(for文を二重に使用する)



## ソート処理 (1) 隣接交換法

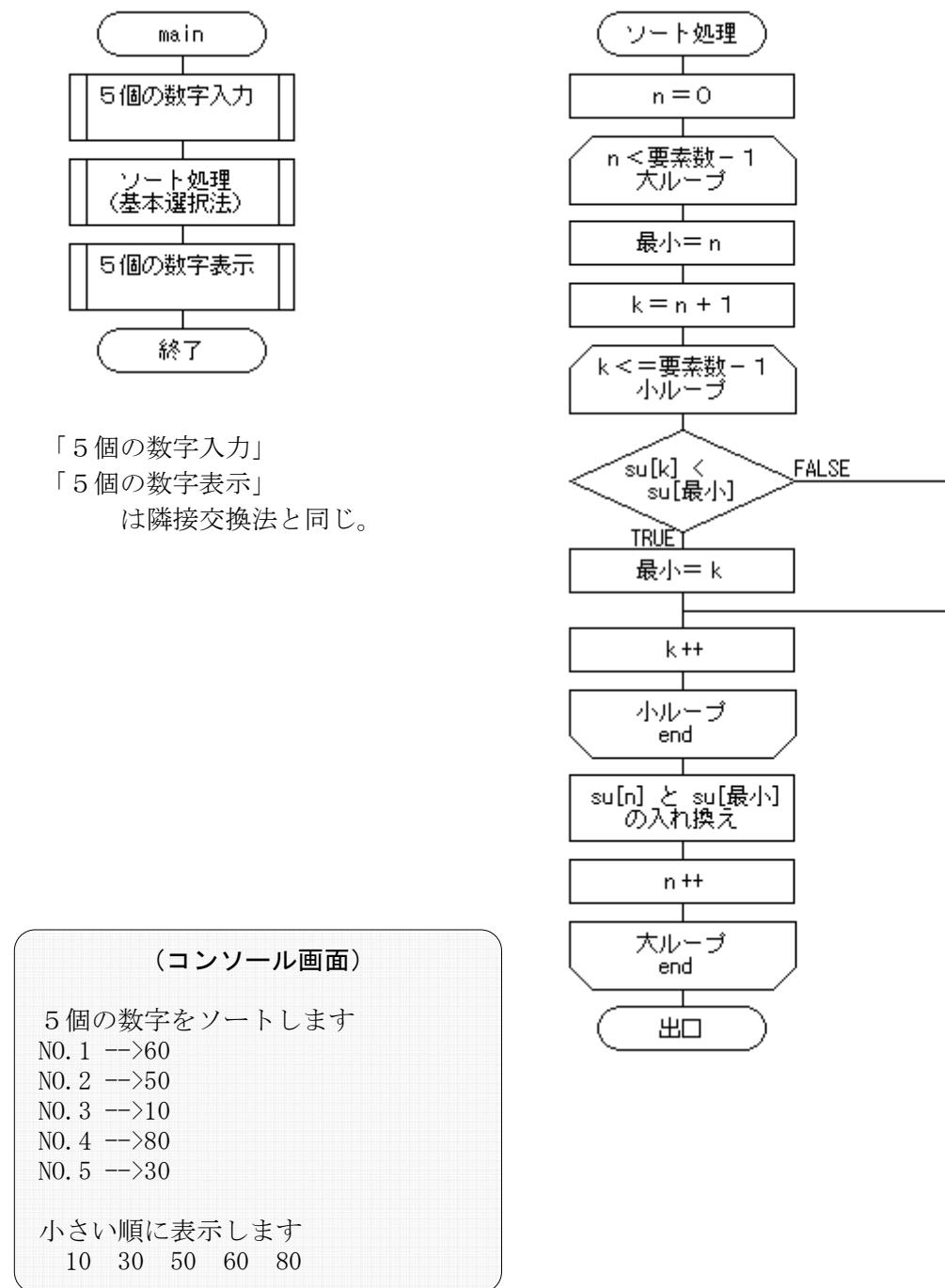


## ソート処理の手順 (隣接交換法)

- 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。

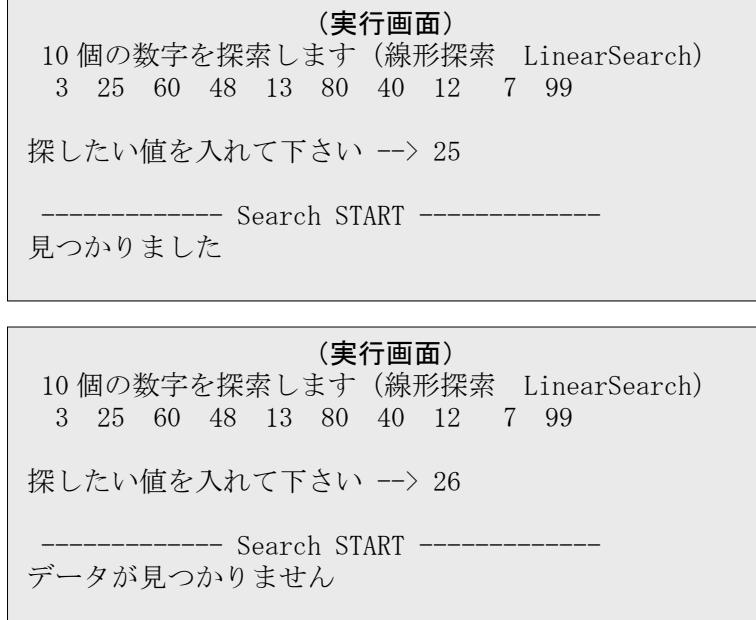
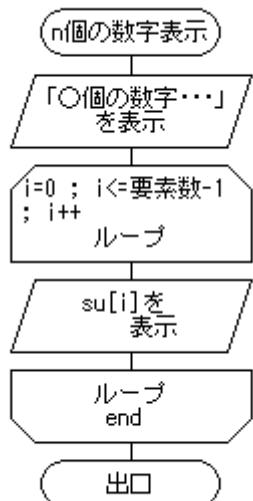
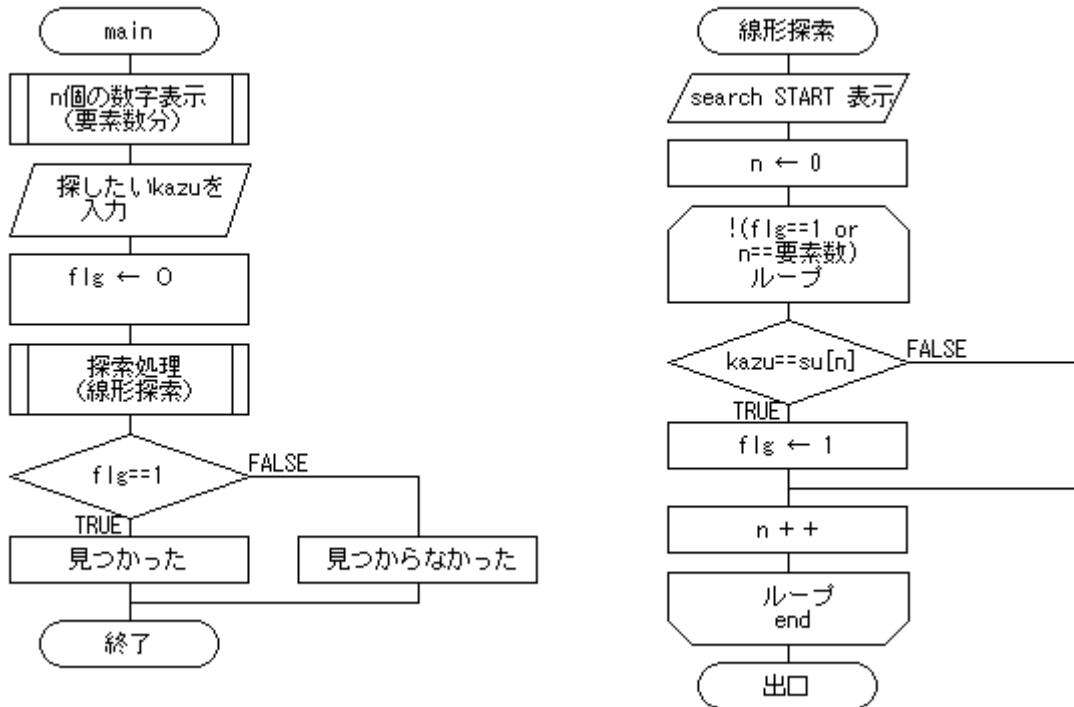
## ソート処理 (2) 基本選択法



## 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なかから、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)

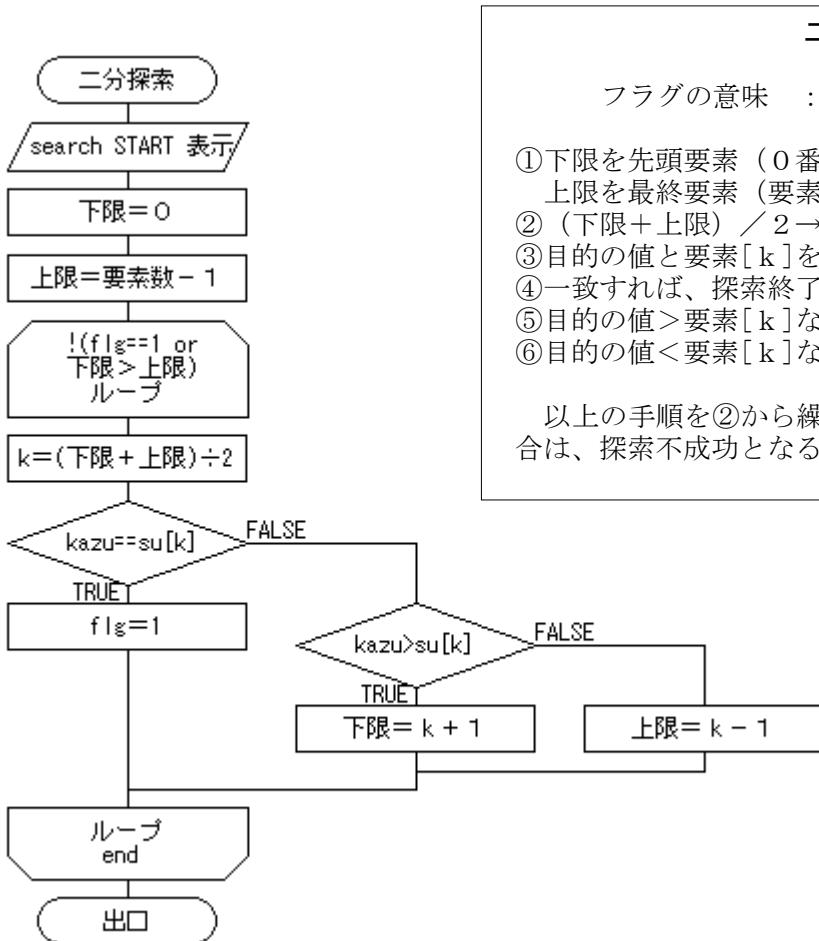
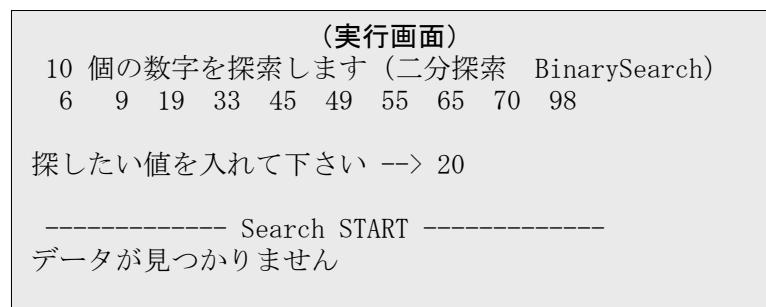
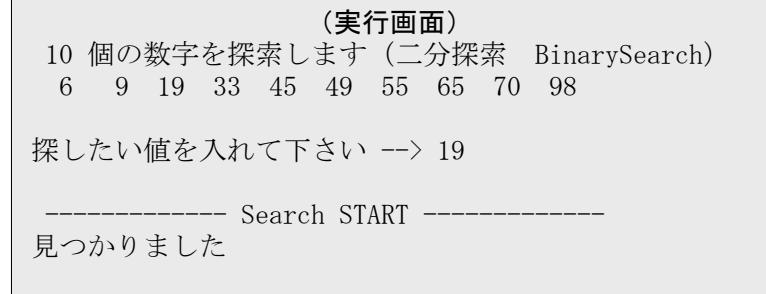
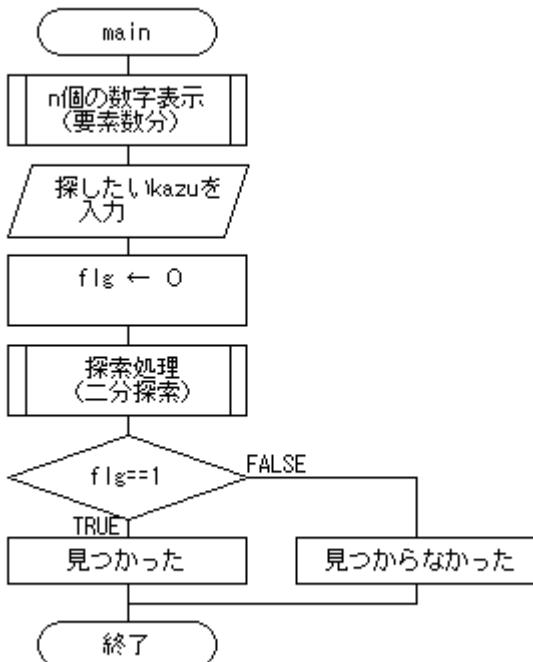


## 線形探索の手順

フラグの意味 : 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)



## 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  $(\text{下限} + \text{上限}) / 2 \rightarrow k$  とする。
- ③目的の値と要素  $[k]$  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