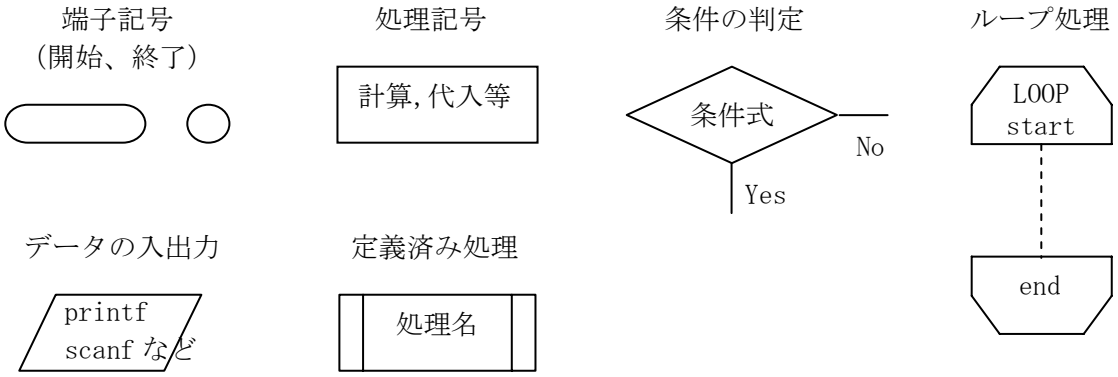
フローチャート（流れ図）

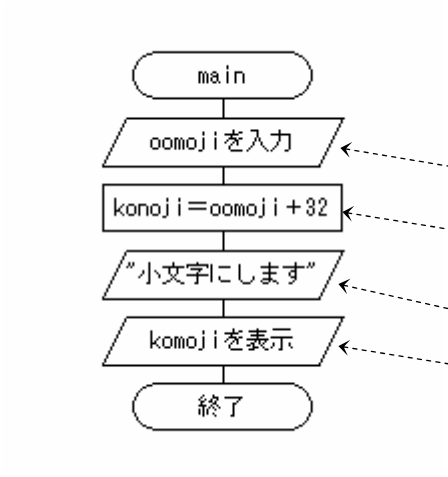
プログラムの処理手順（アルゴリズム）を図示したもの。記号の種類は下記のとおり。



◇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) --
```



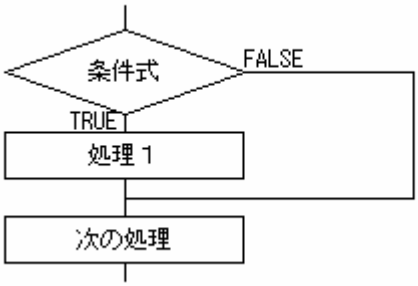

```
/*  -----  大文字から小文字への変換  -----  */
#include <stdio.h>
main()
{
        char oomoji,komoji;

        printf("大文字を入力して下さい--->");
        scanf("%c",&oomoji);

        komoji = oomoji + 32;

        printf("¥n");
        printf("小文字に変換します　→　");
        printf("%c¥n",komoji);
}
```

◇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 oomoji 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命令が、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）。

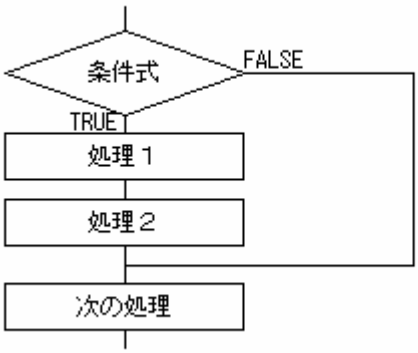
if 文

①単純 if 型（もし～ならば、○○○を行う）




```
if（条件式）
        処理１；
次の処理；
```

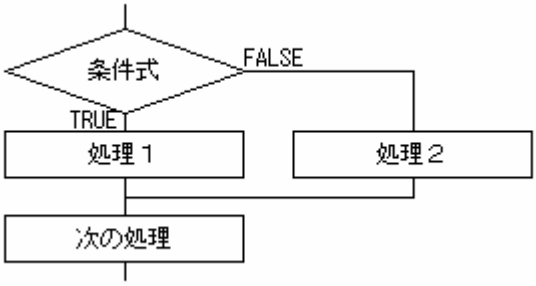
・条件式のあとに；はつけない
・if の中の処理はインデントすると見やすい




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

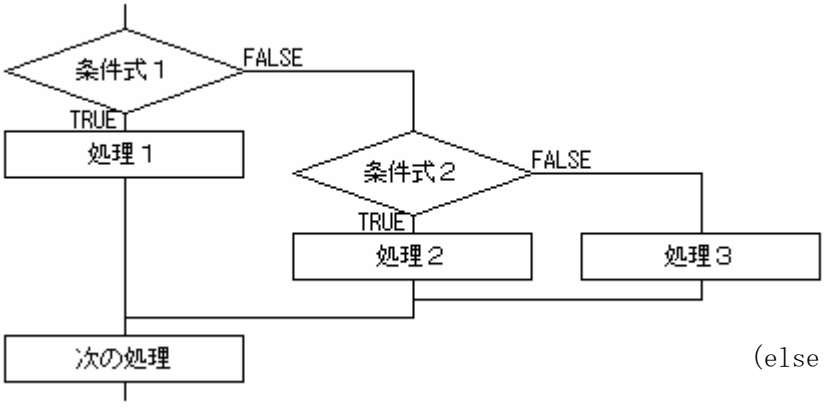
・処理が２つ以上ある時は、{　} を
つけてブロックにする

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）

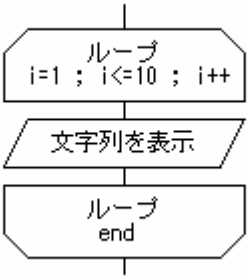
繰り返し型（その１）for 文

```
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

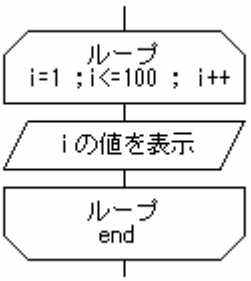
（例１）同じ文字列を 10 個表示するプログラム

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



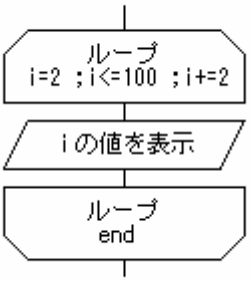
（例２）１から１００までの数を表示するプログラム

```
for ( i=1 ; i<=100 ; i++ ) {
    printf(" %d ",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 ) {
    printf(" %d ",i);
}
```



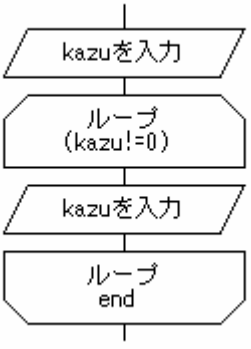
繰り返し型（その２）while 文

```
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）

```
printf("数を入れて下さい");
scanf("%d",&kazu);
while ( kazu!=0 ) {
    printf("数を入れて下さい");
    scanf("%d",&kazu);
}
```



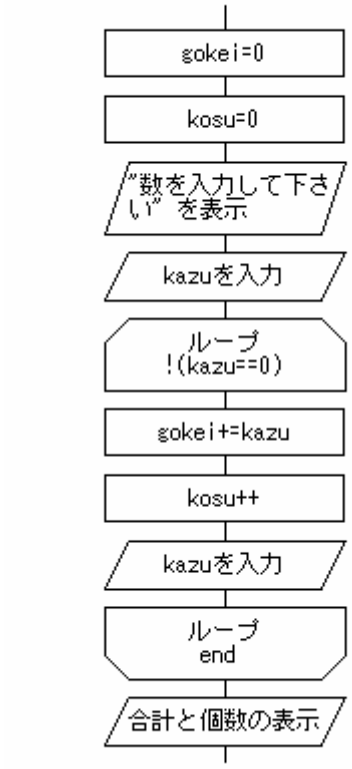
この条件は、while (!(kazu==0)) と書いても同じ。(意味は「～になるまで繰り返す」)

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
```

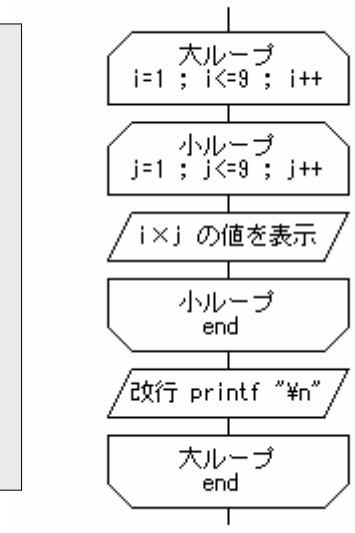


（while 文を使用する）

（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



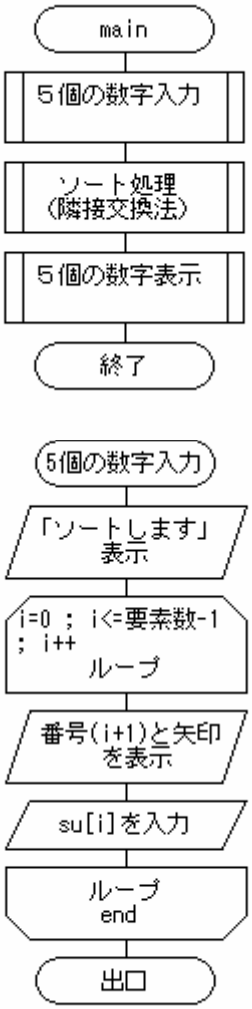
（for 文を二重に使用する）

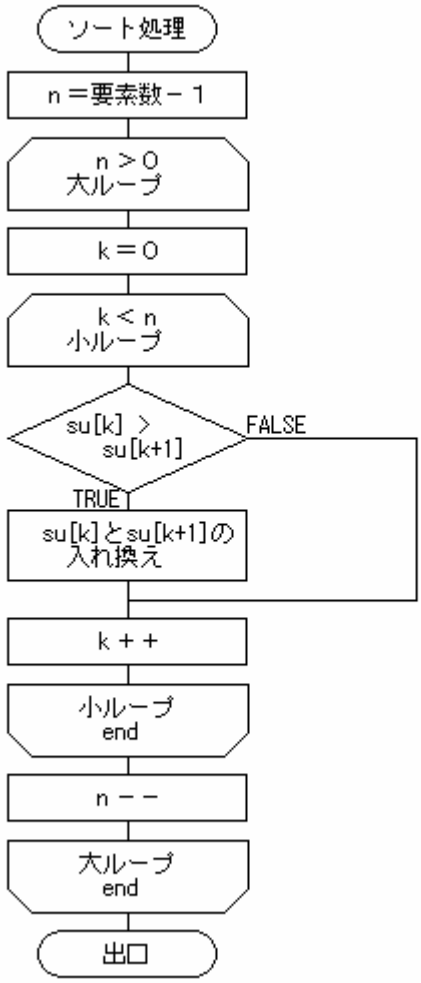
```
for (i=1 ; i<=9 ; i++) {
    for( j=1 ; j<=9 ; j++ ) {
        printf(" %d ",i*j);
    }
    printf("¥n");
}
```

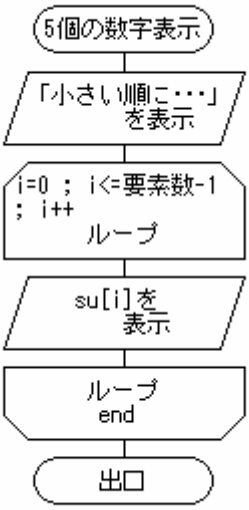
大ループ（ｉが１～９まで変化）

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

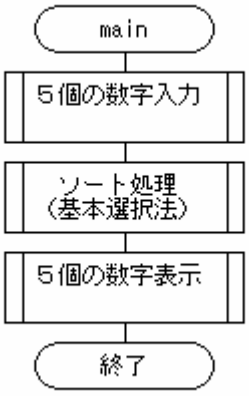
小さい順に表示します
10　20　40　60　90

ソート処理の手順　（隣接交換法）

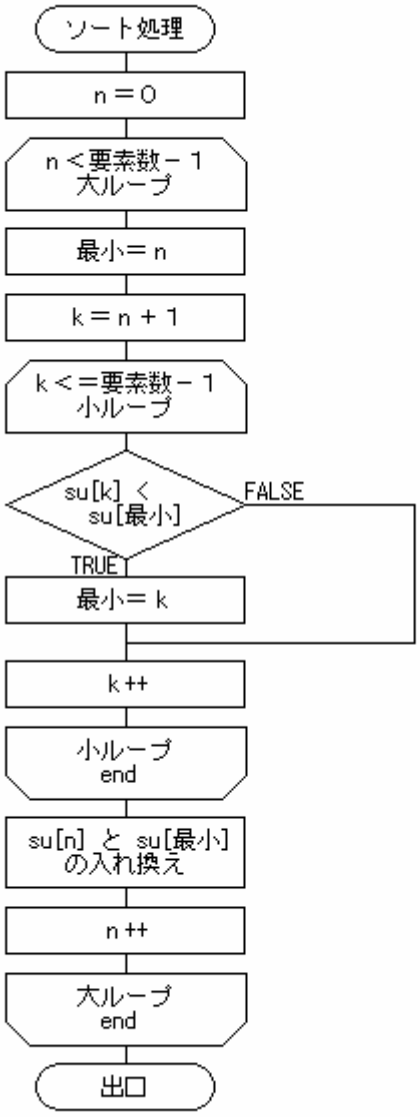
①配列の１番目の要素と２番目の要素を比較する。
②１番目の要素の方が大きければ、両者を交換する。
③要素の２番目と３番目を比較して、同様の処理を行う。
④以上を、配列の最後の要素まで繰り返す。
　（この段階で、配列の右端が最大値となる ---> 確定）
⑤上記の①～④までを、配列の最後から１つ手前の要素まで行う。

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。



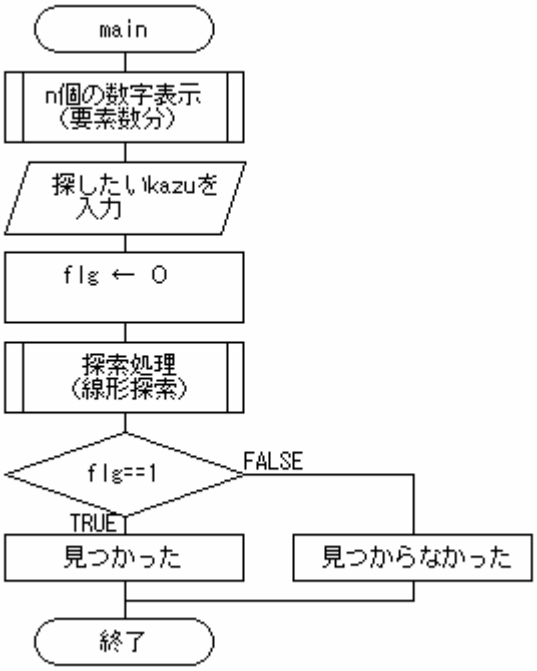
（実行画面）

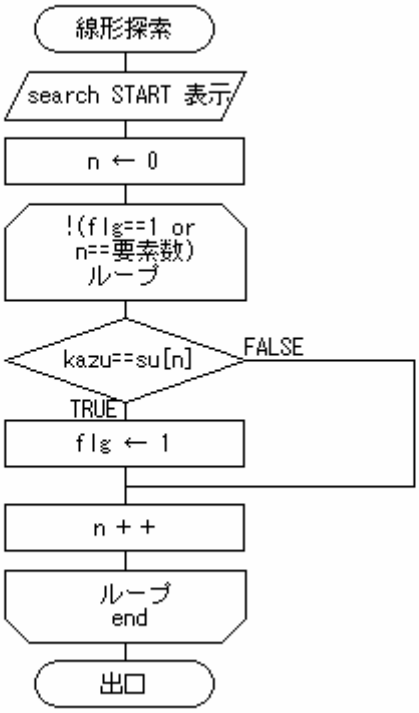
```
５個の数字をソートします
NO.1 -->60
NO.2 -->50
NO.3 -->10
NO.4 -->80
NO.5 -->30

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

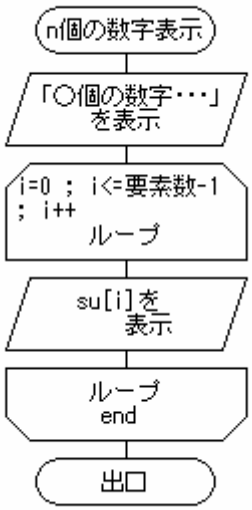
ソート処理の手順　（基本選択法）

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。

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）

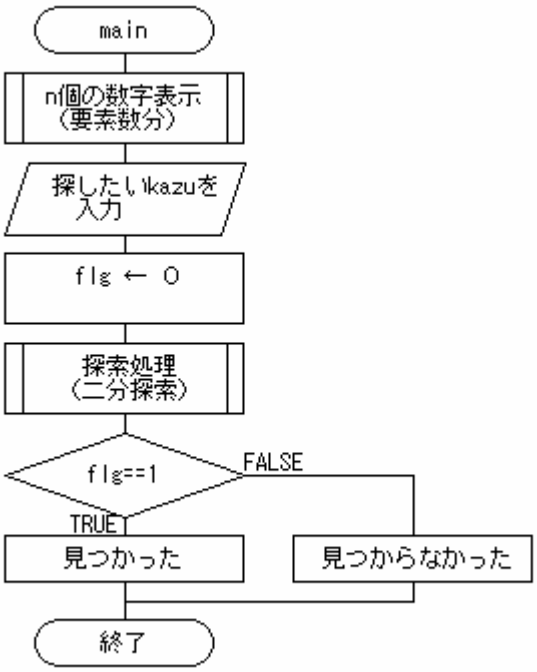
```
10 個の数字を探索します（線形探索　LinearSearch）
  3   25   60   48   13   80   40   12    7   99

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）



（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）

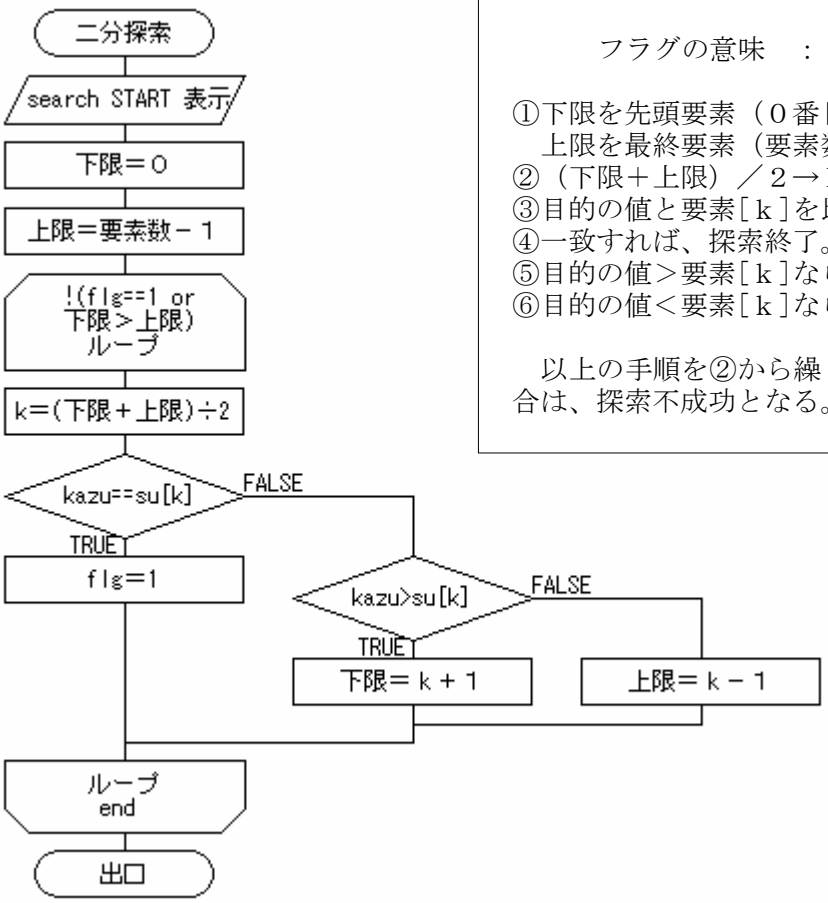
```
 10 個の数字を探索します（二分探索　BinarySearch）
  6    9   19   33   45   49   55   65   70   98

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

２次元配列

「配列の定義」→ データ型　配列名［行数］［列数］

配列要素のイメージ

int tbl[3][5]　　——→　列

↓ 行					
	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

初期値の設定

```
int tbl[3][5]={
        {65,45,85,40,100},
        {66,98,65,95,75},
        {88,32,96,82,79}
};
```

	0	1	2	3	4
0	65	45	85	40	100
1	66	98	65	95	75
2	88	32	96	82	79

（例）tbl[1][2]は１行２列目の「65」を表す

文字型の２次元配列

```
char simei[3][7]={
        {"bunkyo"},
        {"taro"},
        {"hanako"}
};
```

	0	1	2	3	4	5	6
0	b	u	n	k	y	o	¥0
1	t	a	r	o	¥0		
2	h	a	n	a	k	o	¥0

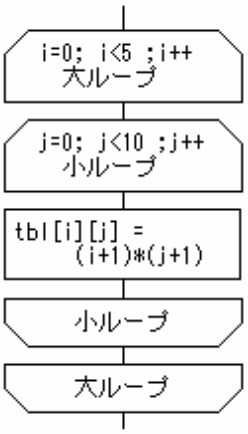
（例）simei[1][2]は文字の 'r' を表す
simei[2]は文字列 "hanako" を表す

「２次元配列の格納と表示」

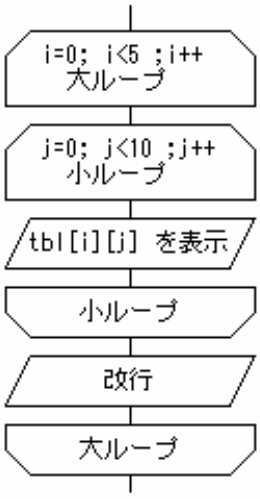
tbl[5][10]に１から５の倍数を、それぞれ 10 個格納して表示する。

（実行結果）

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50



要素へ格納



表示処理

「テストの点数と平均点の計算」

int tensu[5][3]

	英語	数学	国語
生徒１	60	80	100
生徒２	40	90	65
生徒３	10	20	60
生徒４	70	70	100
生徒５	90	100	90

人数＝５、科目数＝３

（実行結果）

```
点数を一覧表示します
  英　数　国
  60　80 100
  40　90　65
  10　20　60
  70　70 100
  90 100　90

生徒の平均点を表示します
  生徒 001 番　80.0 点
  生徒 002 番　65.0 点
  生徒 003 番　30.0 点
  生徒 004 番　80.0 点
  生徒 005 番　93.3 点

科目別の平均点を表示します
  英語　54.0 点
  数学　72.0 点
  国語　83.0 点
```