第4章 数据的组织结构（一）

4.1 数组类型

4.2 利用一维数组组织数据的应用实例

4.3 字符串的组织

4.4 常用的字符串标准函数及应用实例

4.5 二维数组

4.1 数组类型

§ 数组类型的应用背景

（1）同时存在若干个用来描述同一性质且不同个体的数据(**同质数据**)。

（2）只有将这些数据组织在一起形成**批量数据**，共同参与处理，很多操作才具有实际意义。

- 例如：在某个部门中，需要由全体职工推选一名办公室主任。假设有10名候选人准备参与竞选。希望编写一个程序，统计每个候选人的得票数量及选举结果。

§一维数组类型的定义

- 定义格式：

  <元素类型> <数组变量名>[<元素数量>];

  例如：　int vote[10];

- C语言规定：数组的下标从0开始，因此，表示这10个数据的下标为0~9

- 变量一经定义，系统就要为它分配相应的存储空间。在C程序中，系统将会为每个数组型变量分配一片连续的存储空间，所需要分配的存储空间总数将取决于包含的元素个数和每个元素需要的存储空间。

§ 一维数组的初始化

- 基本格式为：

  <元素类型><数组变量名>[<元素数量>]={<元素初值1>，<元素初值2>，.....，<元素初值n>};

- 例如：float score[5] = {9.2, 9.1, 8.7, 9.1, 8.5};

§ 说明：

1）为数组型变量中的每一个元素都提供了一个初始值。此时，可以省略方括号内的数组元素数量。系统将根据花括号中包含的初值数目推测出数组含有的元素数量。

```
float score[ ] = {9.2, 9.1, 8.7, 9.1, 8.5};
```

2）对数组型变量的前面若干个元素赋予初值。此时可以使用下面这种书写形式：

```
int letter[26] = {10, 9, 8, 7};
```

它的执行结果是：将10、9、8、7分别赋予letter数组中下标为0、1、2、3的元素，后面的所有元素赋予初值0。

3）将数组型变量中的每一个元素赋予初值0。此时，可以使用下面这种简化的书写形式：

```
int vote[10] = {0};
```

§ 一维数组元素的引用及基本操作

1. 数组元素的引用

   - <数组变量名>[<下标表达式>]

2. 数组的赋值

   - 利用赋值语句为数组赋值

```
for (i=0; i<10; i++){
    vote[i] = 0;
}
```

   - 调用标准输入函数为数组赋值

```
for (i=0; i<13; i++) {
    scanf("%f", &score[i]);
}
```

3. 数组的输出

```
for (i=0; i<10; i++) {
    printf("%5d", vote[i]);
}
```


4.2 利用一维数组组织数据的应用实例

§按照条件对数据进行筛选

1 在遇到的许多问题中，经常需要从众多的数据中挑选出来满足一定条件的数据，这就是数据的筛选操作。在C程序中，参与筛选操作的批量数据可以采用一个一维数组型变量组织，筛选的条件用逻辑表达式表示。

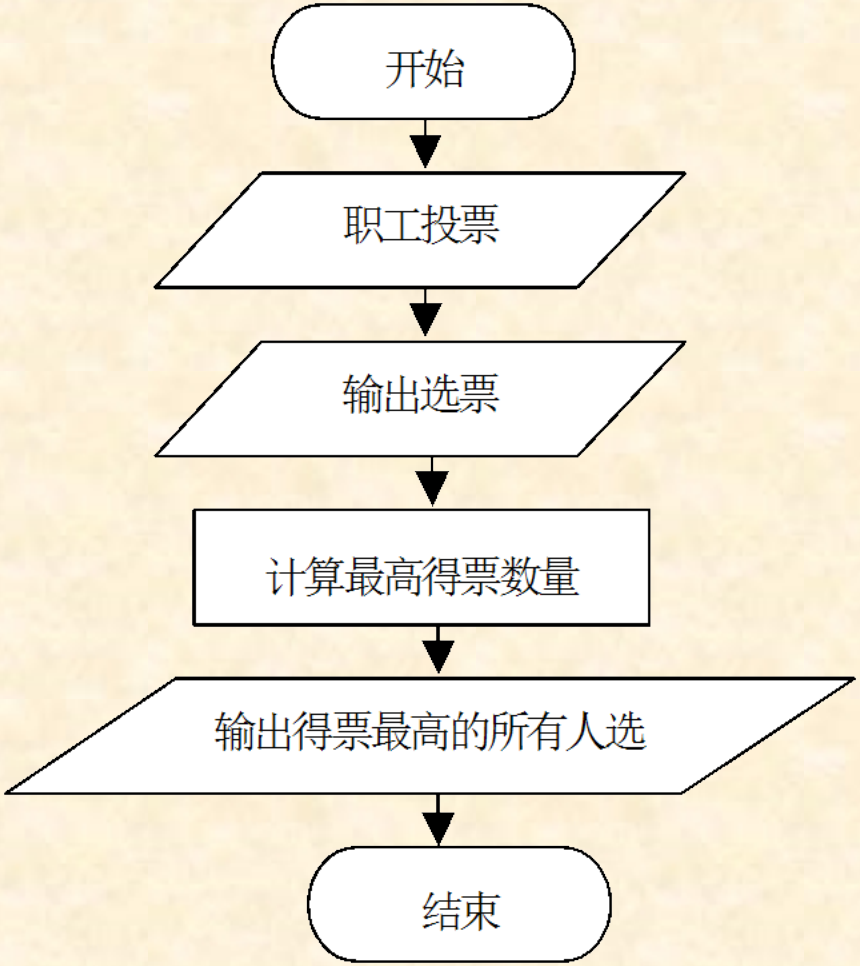
例1：在某个公司中，计划由职工们推选一名办公室主任。假设有10名候选人准备参与竞选。希望编写一个程序，输入一组选举人的投票信息，统计每个候选人的得票数目及选举结果。

§ 问题分析

- 用一维数组记录每位候选人的得票数量。
- 投票通过循环输入介于1~10之间的整型数值来模拟的。

  例如，输入3代表某个职工选举编码为3的候选人。
- 找出最多的得票数量之后，从所有的候选人中筛选出得票数量与最高得票数量相同的人。

§算法描述



程序代码

```
#include <stdio.h>
#define NUM 10                          /* 候选人人数 */
main( )
{
   int vote[NUM] = {0};
   int code, i, winner;
   /* 职工投票 */
   printf("\nEnter your selection<0 end>:\n");
   do {
      scanf("%d", &code);
      if (code<0 || code>NUM) {              /* 检验输入的编码是否有效 */
            printf("\nInvalid vote.");
      } else {
        if (code!=0)
                  vote[code-1] = vote[code-1]+1;      /* 累加票数 */
         }
    } while (code!=0);
```

```
/* 输出选票 */
   printf("\n The amount of votes is :");
   for (i=0; i<NUM; i++) {
       printf("%4d", vote[i]);
   }
   /* 计算最高得票数量 */
   winner = 0;
   for (i=1; i<NUM; i++) {
       if (vote[i]>vote[winner])
                winner = i;
   }
   /* 输出得票最高的所有候选人 */
   printf("\nThe winner :");
   for (i=winner; i<NUM; i++) {
      if (vote[i]==vote[winner])
                printf("%3d",i+1);
   }
}
```

§根据需求对数据进行统计

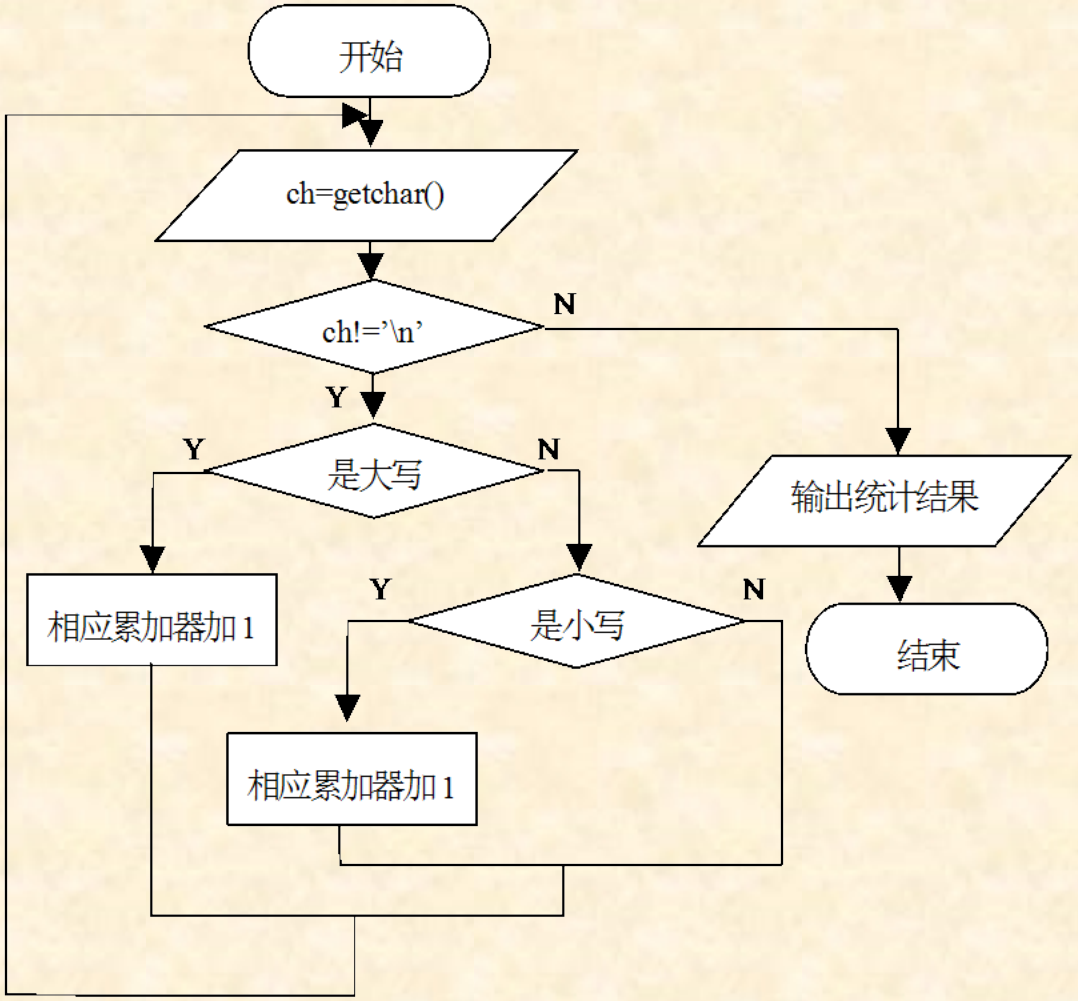
1 为了满足特定的需要，对一组数据的某些特征进行统计是一项经常遇到的基本操作。例如，统计一段文本中某个字符出现的频率；统计学生考试的平均成绩等等都属于统计操作。统计操作的结果往往是通过对所有数据进行扫描、判断或综合加工得到的。在C程序中，参与统计操作的批量数据可以用一维数组来组织，具体统计过程可以通过逻辑判断、累计、算术运算等基本操作手段实现。

例2：在一段文本中，可能会出现各式各样的字符。编写一个程序，从键盘读入一行文本，完成统计每个英文字母出现频率的操作。

§ 问题分析

1. 用一维数组构造26个用于记录每个字母出现次数的累加器。
1. 对于输入的文本字符，可以在读取时检查一下是否为英文字母，而不需要将其存储起来。

§算法描述



§ 程序代码

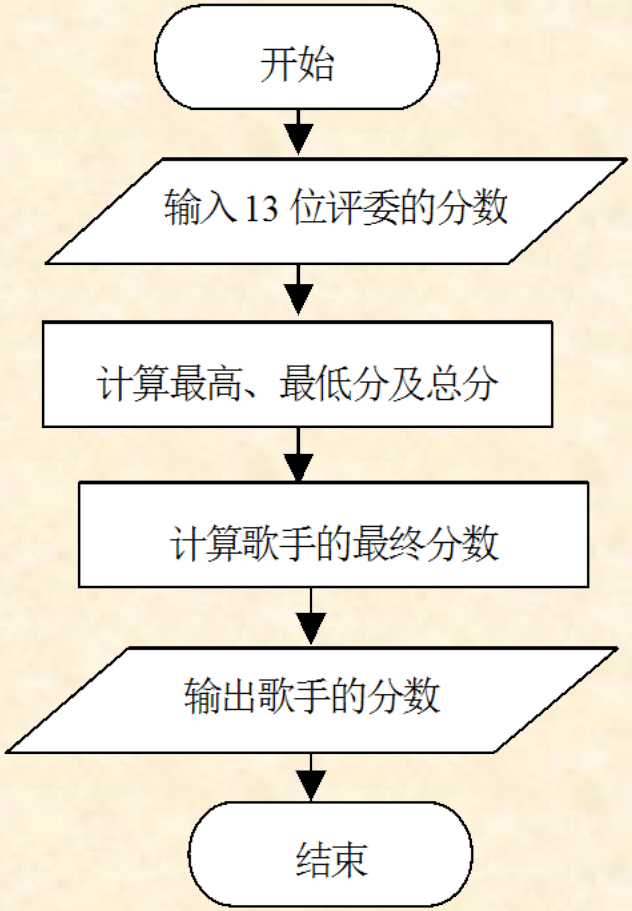
```
#include <stdio.h>
#define NUM 26
main( )
{
   int  letter[NUM] = {0};
   char ch;
   int i;
   printf("\nEnter text line\n");
   while ((ch=getchar()) != '\n') {
     if ('A'<=ch && ch<='Z') {                    /* 检测是否为大写字母 */
         letter[ch-'A'] = letter[ch-'A']+1;
     } else {
       if ('a'<=ch && ch<='z')                    /* 检测是否为小写字母 */
                  letter[ch-'a'] = letter[ch-'a']+1;
     }
   }
   /* 输出每个英文字母出现的次数 */
   for (i=0; i<NUM; i++){
     printf("\n\'%c\':%d", 'A'+i, letter[i]);
   }
}
```

例3：每年中央电视台都要举办青年歌手大奖赛。假设有13位评委参与评分工作。计算每位歌手最终得分的方法是：首先去掉一个最高分和一个最低分，然后计算剩余11个分数的平均值，所得结果就是选手的最终得分。希望编写一个程序，帮助工作人员计算每个歌手的分数。

§问题分析

- 用一维数组存储 13位评委给出的分数
- 寻找最高分和最低分
- 计算剩余11个分数的平均分

§算法描述



§ 程序代码

```
#include <stdio.h>
#define NUM 13
main( )
{
   float score[NUM]; */
   int i, minValue, maxValue;
   float sum;

   /* 输入13位评委给出的分数 */
   printf("\nEnter 13 score:");
   for (i=0; i<NUM; i++){
       scanf("%f", &score[i]);
   }
```

```
/* 找出最高分、最低分，并同时累加13个分数的总和 */
   minValue = score[0];
   maxValue = score[0];
   sum = score[0];
   for (i=1; i<NUM; i++) {
        if (score[i]<minValue)
                minValue = score[i];
        if (score[i]>maxValue)
                maxValue = score[i];
        sum = sum+score[i];
   }

   /* 计算并输出歌手的最终得分 */
   sum = (sum- minValue-maxValue)/(NUM-2);
   printf("\nFinal score is %6.2f", sum);
}
```

§ 查找问题

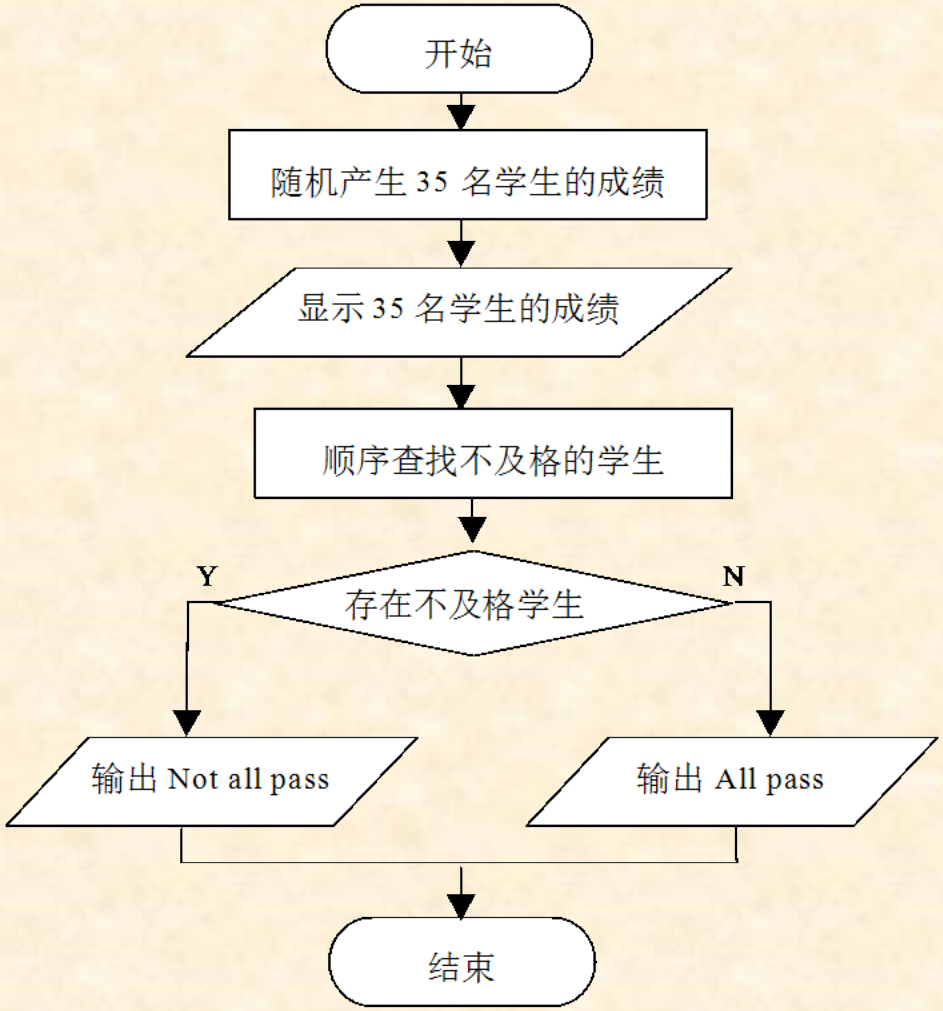
1 所谓查找是指根据某个给定的条件，在一组数据中搜索是否存在满足该条件的数据的过程。如果存在，则表示查找成功，给出成功的标志；否则表示查找不成功，给出失败的标志。在程序中，查找操作的结果经常被用来作为是否执行某项后续操作的决策依据。

例4：已知某个班级35名学生的某门课程的考试成绩。请编写一个程序，查看在这个班级中是否存在不及格的学生。

§ 问题分析

1. 用一维数组记录每位学生的考试成绩，下标表示每个学生的编号，元素内容表示考试成绩。
1. 查找可以通过从前往后依次查看每个元素内容的过程实现。

§ 算法描述


开始
随机产生 35 名学生的成绩
显示 35 名学生的成绩
顺序查找不及格的学生
Y
存在不及格学生
N
输出 Not all pass
输出 All pass
结束

```
#include <stdio.h>
#include <stdlib.h>
#define NUM 35   /*学生人数*/
main( )
{
        int score[NUM];
        int i;
          /* 随机产生35个考试成绩 */
         randomize( );
         for (i=0; i<NUM; i++) {
                score[i] = random(100);
        }
        /*显示35名学生的考试成绩*/
        for (i=0; i<NUM; i++) {
               printf("\nNo.%d: %d", i+1, score[i]);
        }
```

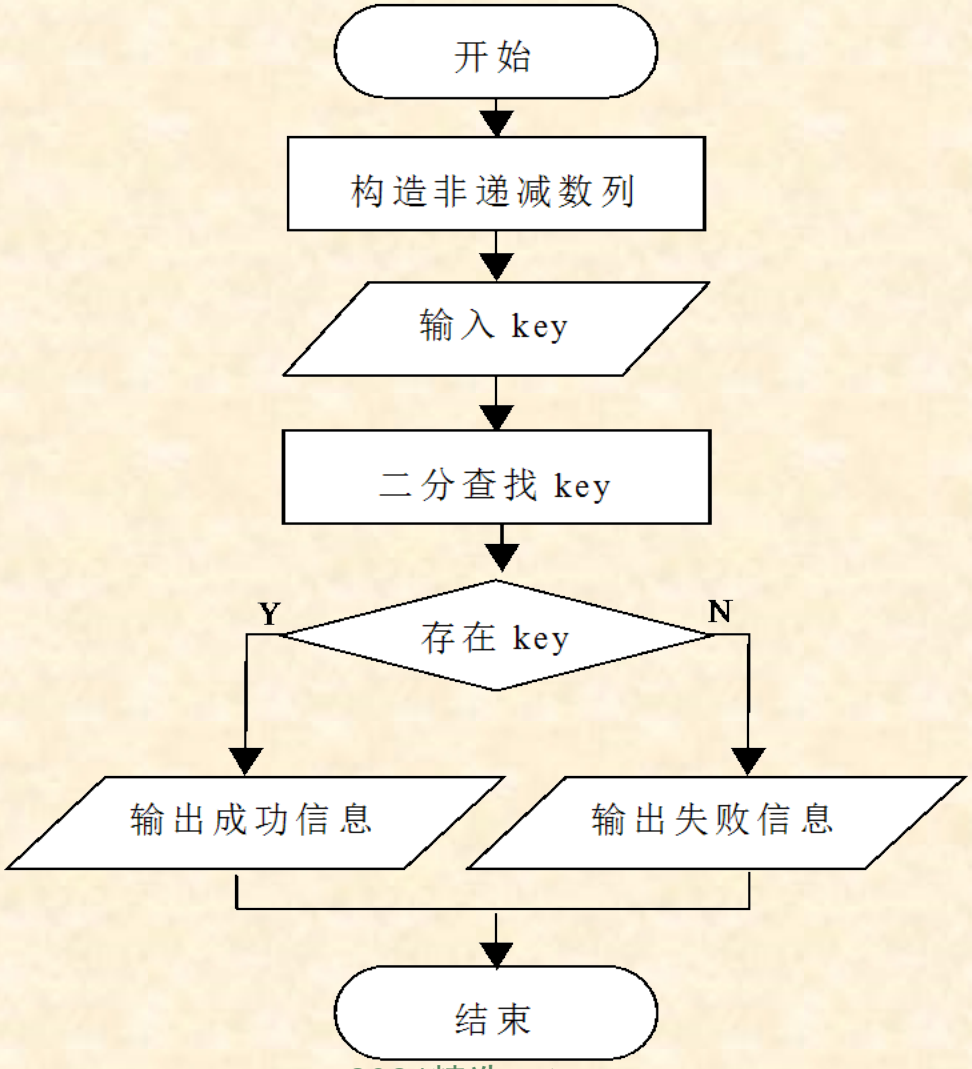
```
    /*顺序查找是否存在不及格的学生*/
    for (i=0; i<NUM; i++) {
        if (score[i]<60)  break;
    }

    /*输出查找结果*/
    if (i<NUM)
        printf("\nNot all pass.");
    else
        printf("All pass.");
}
```

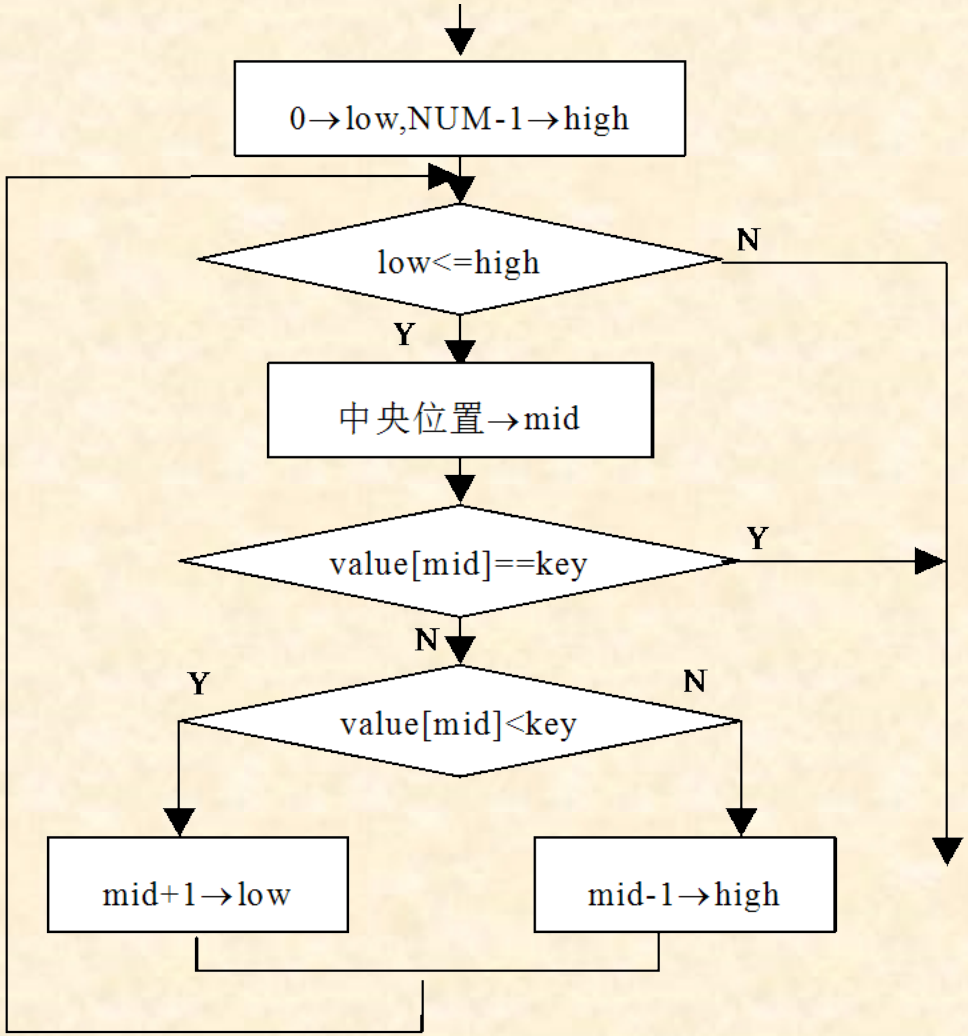
例5：已知一个按非递减有序排列的整型数列（12, 23, 3
3）。请编写一个程序，查找其中是否存在与给定key相等的数值。

§ 问题分析

- 二分查找是指每次用key与位于查找区间中央位置的元素进行比较，比较结果将会产生下面三种情形之一：
  1. 如果相等，说明查找成功。
  2. 如果key小于中央位置的元素，说明如果存在这样的元素，应该位于查找区间的前半部分。此时可以将查找区间缩减为原来的一半，并在这一半的区间中继续用相同的方式查找。
  3. 如果key大于中央位置的元素，说明如果存在这样的元素，应该位于查找区间的后半部分。同样可以将查找区间缩减为原来的一半，并在这一半的区间中继续用相同的方式查找。
- 可以看出，用key与当前查找区间中央位置的元素比较后，不是找到了，就是将查找区间缩小了一半。直到查找区间不存在了，说明没有要找的key。

§算法描述




0→low,NUM-1→high
low<=high
N
Y
中央位置→mid
value[mid]==key
Y
N
Y
value[mid]<key
N
mid+1→low
mid-1→high

```
#include <stdio.h>
#define NUM 10
main( )
{
   int value[NUM] = {12, 23, 30, 45, 48, 50, 67, 82, 91, 103}; /* 非递减整型数列 */
   int low, high, mid, key;
   printf("\nEnter a key:");                    /* 输入查找的数值 */
   scanf("%d", &key);
   low = 0;              high = NUM-1;
   while (low<=high) {
            mid = (low+high)/2;
            if (value[mid]==key)              break;
            if (value[mid]<key)
                     low = mid+1;
            else
                  high = mid-1;
   }
   if (low<=high)
            printf("\n%d is found at %d.", key, mid);          /* 确认break出口 */
   else
            printf("\n%d is not found.", key);                 /* 确认循环正常出口 */
}
```

用递归函数实现二分查找

```
int search(int value[ ], int key, int low, int high)
{                                                    /* 二分查找的递归函数 */
   int mid;

   if (low>high)
                  return -1;                         /* 查找区间为空 */
   mid = (low+high)/2;                 /* 求中间位置 */
   if (value[mid]==key)
                  return mid;                        /* 得到查找的数据位置 */
   if (key<value[mid])
         return search(value, key, low, mid-1);  /* 在下半区查找 */
  else
      return search(value, key, mid+1, high);    /* 在上半区查找 */
}
```