第5章 数组程序设计

数组是C语言的一种重要数据结构，使用数组可以实现一组同类型数据的连续存储和有效处理。本章介绍使用数组的程序设计，包括一维数组和二维数组的定义、初始化、在计算机中的存储及使用方法，字符串的输入与输出操作及常用的字符串操作函数，并通过大量实例介绍数组应用程序的设计方法。

5.1 一维数组程序设计

本节首先通过批量处理数据的一个简单示例说明数组在数据处理中的作用，然后逐步介绍一维数组程序设计的基本知识。

5.1.1 一维数组程序示例

下面是一个简单的数据处理程序，其功能是从键盘输入10个整数，然后按照与输入相反的顺序依次将它们输出。

```
/* program e5-0.c */
#include<stdio.h>
int main()
{
    int a,b,c,d,e,f,g,h,i,j;
    printf("Input Data:  ");
    scanf("%d%d%d%d%d%d%d%d%d%d",&a,&b,&c,&d,&e,&f,&g,&h,&i,&j);
    printf("Output Data: ");
    printf("%d %d %d %d %d %d %d %d %d %d\n",j,i,h,g,f,e,d,c,b,a);
    return 0;
}
```

这是一个正确的程序，但并不是一个好程序。如果处理的数据规模进一步扩大，有成千上万个数据时，变量的这种表示方法显然是不适用的，很难想象以类似的方式定义和使用成千上万的变量时程序会是什么样子。

本章要讨论的数组将有效解决上述问题。数组是包含多项同类数据的一种数据结构，它能将一系列相同类型的数据组织起来，使用同一个名称，再用下标进行分量标识。例如a[0]、a[1]、a[2]、…、a[9]等，当下标用一个变量i表示时i的不同取值即对应不同的分量，使用a[i]即可访问这一组数据的任何一个分量，这里的a就是一个数组。以下是用数组解

决上述问题的程序。

【例 5-1】 从键盘输入 10 个整数，然后按照与输入相反的顺序依次将它们输出。

程序如下：

```
#include<stdio.h>
int main()
{
    int i,a[10];                          /*定义 a 数组*/
    printf("Input: ");
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);                /*输入 a[0]、a[1]、a[2]、…、a[9]*/
    printf("Output: ");
    for(i=9;i>=0;i--)
        printf("%d ",a[i]);               /*输出 a[9]、a[8]、a[7]、…、a[0]*/
    return 0;
}
```

程序执行结果：

```
Input:0 1 2 3 4 5 6 7 8 9↵
Output: 9 8 7 6 5 4 3 2 1 0
```

该程序中定义了一维数组 a，用 a[i]表示数组 a 的一个元素，当 i 为 0 时即为 a[0]，当 i 为 5 时即为 a[5]。“scanf("%d",&a[i]);”语句执行 10 次，i 依次取值 0 到 9，故依次为 a[0]到 a[9]输入数据。“printf("%d",a[i]);”语句也执行 10 次，i 依次取值 9 到 0，故依次输出 a[9]到 a[0]的值。

5.1.2 一维数组的定义及元素引用

1. 一维数组的定义

一维数组在使用之前必须先定义，一般格式如下：

```
数据类型  数组名[数组长度]
```

例如：

```
int a[10];
```

该语句定义了数组名为 a 的 int 型数组，该数组有 10 个元素，能够存储 10 个整数值。

```
char name[20];
```

该语句定义了数组名为 name 的 char 型数组，该数组有 20 个元素，能够存储 20 个字符。

说明：

(1) 数组的数据类型即数组元素的数据类型，它可以是 C 语言允许的任何数据类型。

(2) 数组长度是数组能够包含的数组元素的个数，通常用一个整数值表示，也可以是常量表达式，但不允许是包含变量的表达式。例如，下面对数组的定义方法是错误的，其原因是定义数组长度时使用了变量 n。

```
int n = 10;
float a[n];
```

2. 一维数组的元素引用

一维数组的输入、输出、运算等一般通过数组元素进行。数组元素的引用形式如下：

```
数组名[下标]
```

数组元素的下标从0开始，当数组长度为n时最后一个元素的下标是n－1。例如，上述a数组的各元素依次为a[0]、a[1]、a[2]、…、a[9]，上述name数组的各元素依次为name[0]、name[1]、name[2]、…、name[19]。

在实际使用中，下标可以是一个整数型常量，也可以是整数型表达式，最常用的是用一个整数型变量表示元素的下标。例如上述a数组，其任意元素可表示为a[i]，当指定i的值后a[i]则表示a数组的一个特定元素。当i＝0时，a[i]即代表a[0]元素；当i＝1时a[i]即代表a[1]元素；以此类推。

5.1.3 数值型一维数组的输入和输出

数值型数组的输入和输出是通过每个数组元素的输入和输出实现的。数值型一维数组的元素都是一些简单变量，输入和输出按照简单变量的方法进行。

例如对于上述a数组，输入a[5]的值时使用如下语句：

```
scanf("%d",&a[5]);
```

输出a[5]的值时使用如下语句：

```
printf("%d",a[5]);
```

【例5-2】 向数组输入10个整数，通过相邻元素比较、交换的方法，将最大值移到数组末尾，然后输出该数组中所有元素的值。

1) 算法设计

设用数组a存储数据，它的10个元素分别为a[0]、a[1]、a[2]、…、a[9]，这10个元素比较9次即可将最大值移到最后，即a[9]的位置。每次比较时，进行比较的两个相邻元素分别为a[i]和a[i＋1]，比较示意图如图5-1所示，算法流程图如图5-2所示。

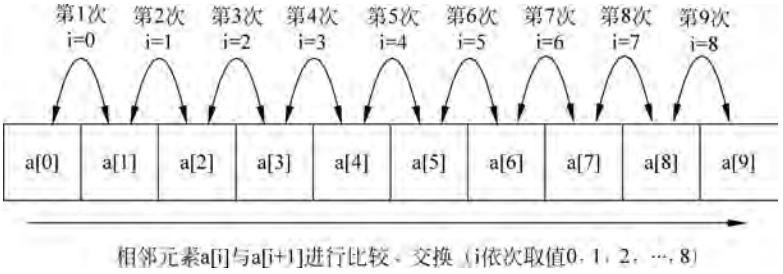

图5-1 比较示意图

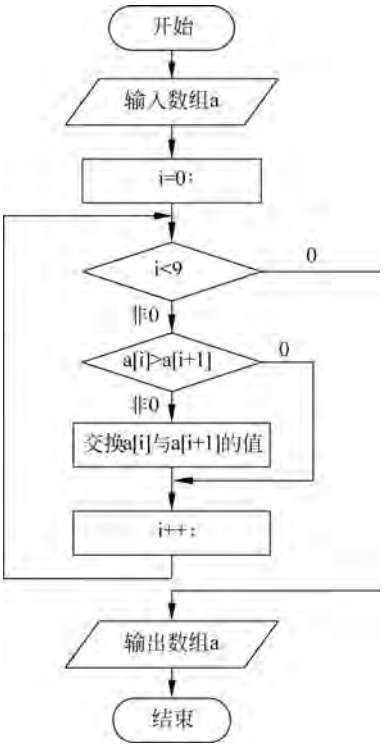

图 5-2　例 5-2 算法流程图

2）实现程序

程序如下：

```
#include<stdio.h>
#define N 10
int main()
{
    int a[N],i,temp;                    /*定义整数型数组 a*/
    for(i=0;i<N;i++)                    /*为数组 a 输入数据*/
        scanf("%d",&a[i]);
    for(i=0;i<N-1;i++)                  /*把数组 a 中的最大数后移*/
        if(a[i]>a[i+1])
        {                               /*相邻元素前者大、后者小时交换两个元素的值*/
            temp=a[i];
            a[i]=a[i+1];
            a[i+1]=temp;
        }
    for(i=0;i<N;i++)                    /*输出数组 a 的所有元素的值*/
        printf("%d ",a[i]);
    return 0;
}
```

下面是程序的执行结果：

```
19 7 21 6 25 8 17 20 11 10 ↵
7 19 6 21 8 17 20 11 10 25
```


程序中使用了3个并列的for循环。第1个for循环用于输入10个整数,其中的a[i]是数组a的元素的一般表示,&a[i]是它的地址形式。第2个for循环对数组a的10个元素从头开始进行两两比较,将较大值后移,循环结束时数组a的最大值存储在最后一个元素中。N个元素共需比较N−1次。第3个for循环用于输出数组a的全部元素。

问题思考

该题目的数组a经过这样一趟(共N−1次)比较、交换后,最大值移到了数组的最后位置。如果对其余元素也同样进行一趟比较、交换,会出现什么结果?

5.1.4 数值型一维数组的初始化

数组的初始化是指在定义数组时对数组元素赋初值。通常有两种情况,即全部元素的初始化和部分元素的初始化。

1. 全部元素的初始化

其一般格式如下:

```
数据类型  数组名[数组长度]={数组全部元素值表}
```

"数组全部元素值表"是用逗号分隔的各数组元素的初值。

例如:

```
int a[6]={10,20,30,40,50,60};
```

该语句定义了整数型数组a,它有6个元素a[0]～a[5],初值依次为10、20、30、40、50、60。

当需要对全部元素初始化时数组长度允许省略,格式如下:

```
数据类型  数组名[]={数组全部元素值表}
```

当用这种省略方式初始化数组时,数组的长度由"数组全部元素值表"中值的个数确定。

例如:

```
float r[]={12.5,-3.11,8.6};
```

该语句定义了长度为3的float型数组r,其数组元素r[0]、r[1]、r[2]的初值分别为12.5、−3.11、8.6。

2. 部分元素的初始化

其使用较多的情况是对前部元素的初始化。一般格式如下:

```
数据类型  数组名[数组长度]={数组前部元素值表}
```

例如:

```
int b[10]={1,2,3};
```

该语句定义了整数型数组b,它有10个元素,前3个元素b[0]、b[1]、b[2]的初值分别为1、2、3,其余元素的初值为0。若b未进行初始化,则各元素的初始值由数组b的存储属性决定。变量的存储属性将在后续内容中介绍。

读者需要注意，当只对数组的部分元素初始化时数组长度的说明是不能省略的。

另外，由于数组元素本身是一个变量，因此可以使用赋值语句对其单独赋值。例如：

```
int array[10];
array[5] = 26;
array[7] = 38;
```

【例 5-3】 将 Fibonacci 数列的前 20 项存储在一维数组中，然后输出这些数据。

1）算法设计

在第 4 章已经讨论过 Fibonacci 数列问题。读者将会发现，用数组处理这个问题也是一种有效的方法。

首先定义一个 int 型一维数组 fib 并初始化：

```
int fib[20] = {1,1};
```

初始化后，元素 fib[0]和 fib[1]的值即是 Fibonacci 数列前两项的值。

那么，Fibonacci 数列自第 3 项起每一项可按以下方式求值，并存储在 fib 数组中。

```
fib[i] = fib[i - 1] + fib[i - 2] (i = 2,3,4,…,19)
```

求解 Fibonacci 数列的算法流程图如图 5-3 所示。

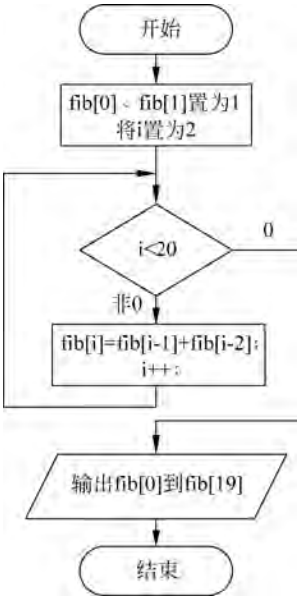

图 5-3 求解 Fibonacci 数列算法流程图

2）实现程序

程序如下：

```
#include <stdio.h>
int main()
{
    int fib[20] = {1,1},i;                    /* fib 数组初始化 */
    for(i = 2;i < 20;i++)
        fib[i] = fib[i - 1] + fib[i - 2];     /* 第 i 项为其紧邻的前两项之和 */
    for(i = 0;i < 20;i++)                     /* 控制输出每个值 */
    {
        printf("%-10d",fib[i]);               /* 输出 1 个 Fibonacci 数列中的值 */
        if((i + 1) % 5 == 0)                  /* 每输出 5 个数之后换行 */
            printf("\n");
    }
    return 0;
}
```

程序执行结果：

```
1     1     2     3      5
8     13    21    34     55
89    144   233   377    610
987   1597  2584  4181   6765
```

程序执行后,Fibonacci 数列的前 20 个数在 fib 数组中的存储情况如图 5-4 所示。

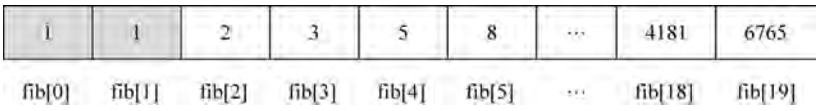

图 5-4　存储在 fib 数组中的 Fibonacci 数列

将 Fibonacci 数列前两项的值存储在 fib[0]、fib[1]中的操作也可由以下赋值语句实现。

```
fib[0] = fib[1] = 1;
```

问题思考

在该程序中,生成 Fibonacci 数列和输出 Fibonacci 数列是分开进行的,能否修改程序将这两个过程合并在一个步骤中?

5.1.5　字符型一维数组的初始化

字符型数组是数据类型为 char 型的数组,用于存储字符串,每个元素存储一个字符。字符型数组与数值型数组在本质上没有区别,但在具体使用时还是有其自身的特点。

(1) 使用字符常量对字符数组初始化。例如:

```
char string[8] = {'e','x','a','m','p','l','e','\0'};
```

上面的语句定义了字符数组 string,该数组共有 8 个元素,string[0]到 string[6]这 7 个元素的初始值分别为字符常量'e'、'x'、'a'、'm'、'p'、'l'、'e',string[7]的初始值为转义字符常量'\0'。'\0'是 C 语言字符串的结束标志,在进行字符串处理时它标志一个字符串的结束。

(2) 使用字符串对字符数组初始化。例如:

```
char string[8] = "example";
```

当使用这种方式对字符数组初始化时,系统自动在字符串尾部增加一个结束标志'\0',使元素 string[7]自动获得'\0'结束符,各元素的初始化情况与(1)相同。

(3) 在初始化时可省略数组长度说明,数组的实际长度由系统根据初始化的形式确定。例如:

```
char string[ ] = "example";
```

对于这种情况,系统将根据存储长度自动设置数组 string 的长度为 8。

5.1.6　一维数组的存储

任何一个一维数组在内存中都占用一段连续的存储空间,依次存储它的各元素的值。5.1.4 节中定义的数组 a 及 5.1.5 节中定义的数组 string 的存储情况如图 5-5 所示,数组元素占用的字节数由数组的数据类型决定。数组 a 的每个元素为 int 型,VC++ 6.0 编译系统为其分配 4 字节的存储空间;数组 string 的每个元素为 char 型,分配 1 字节的存储空间。

图 5-5　一维数组的存储

5.2　字符串操作

字符串在数据处理中有重要的应用，例如统计一段文字中单词的数量、按学生姓名查找学生信息等都是典型的字符串处理问题。C 语言使用字符数组存储字符串，并且为方便字符串处理，专门设计了功能丰富的字符串操作函数。

5.2.1　字符串的输入和输出

C 语言提供了多个函数支持字符串的输入和输出操作，例如专门的字符串输入输出函数 gets()和 puts()、格式化输入输出函数 scanf()和 printf()等。

1. 使用 gets()函数和 puts()函数输入、输出字符串

实现字符串的输入和输出操作最常用的方法是使用 gets()函数和 puts()函数，这两个函数专门为字符串的输入和输出设计。

1) 使用 gets()函数输入字符串

gets()函数的功能是从标准输入设备输入一个字符串，并存储在指定数组中。其一般用法如下：

```
gets(字符数组名)
```

例如：

```
char str[12];
gets(str);
```

执行 gets(str)函数后，从键盘输入一个字符串存储到 str 数组中。

gets()函数以 Enter 键作为输入结束符，而在字符数组中对应存储一个字符串结束标记'\0'。

2) 使用 puts()函数输出字符串

puts()函数的功能是输出存储在字符数组中的字符串。其一般用法如下：

```
puts(字符数组名)
```

例如：

```
char c[6] = "China";
puts(c);
```

该 puts(c)函数被执行后,立即输出存储在字符数组 c 中的字符串。结果如下:

```
China
```

【例 5-4】 用 gets()函数输入一个字符串,将其存储到 str 数组中,然后使用 puts()函数输出 str 中的字符串。

程序如下:

```
#include <stdio.h>
#define N 100
int main()
{
    char str[N];                          /* 定义字符数组 str,用于存储字符串 */
    printf("String: ");
    gets(str);                            /* 输入字符串,存储到 str 数组中 */
    printf("Result: ");
    puts(str);                            /* 输出 str 数组中的字符串 */
    return 0;
}
```

程序执行结果:

```
String: This is an example.↵
Result: This is an example.
```

2. 使用 scanf()和 printf()输入、输出字符串

在格式化输入输出函数 scanf()和 printf()中设置了"%s"格式符,专门用于字符串的输入和输出。

【例 5-5】 用 scanf()函数输入一个字符串,将其存储到 str 数组中,然后使用 printf()函数输出 str 中的字符串。

程序如下:

```
#include <stdio.h>
#define N 100
int main()
{
    char str[N];                          /* 定义字符数组 str,用于存储字符串 */
    printf("String: ");
    scanf("%s",str);                      /* 输入字符串,存储到 str 数组中 */
    printf("Result: ");
    printf("%s\n",str);                   /* 输出 str 数组中的字符串 */
    return 0;
}
```

程序执行结果:

```
String: example.↵
Result: example.
```

再次执行：

```
String: This is an example.↵
Result: This
```

在使用 scanf()函数和%s 格式符输入字符串时需要注意以下几点：

(1) 在 C 语言中，数组名代表数组的起始地址，因此使用字符数组接收字符串时在 scanf()函数中直接使用该数组名。例如，上述程序中的 scanf("%s",str)函数直接使用字符数组名 str。

(2) 在输入的字符串中，只有第 1 个空格(字符串前端空格除外)之前的字符串被读入字符数组中。上面给出了程序两次执行的结果，在第 2 次执行程序时，输入字符串中有空格符，结果表明空格之后的字符串并未输入字符数组 str 中。

(3) 可以一次输入多个字符串，输入的各字符串之间要以“空格”分隔。

例如：

```
char str1[5],str2[5],str3[5];
scanf("%s%s%s",str1,str2,str3);
输入数据：How␣are␣you?↵
```

则字符数组 str1、str2、str3 分别获得字符串"How"、"are"、"you?"，其存储情况如图 5-6 所示。

str1:	H	o	w	\0	
str2:	a	r	e	\0	
str3:	y	o	u	?	\0

图 5-6　数组 str1、str2 及 str3 的存储情况

问题思考

(1) 在例 5-4 的程序中，字符串的输出操作是由“puts(str);”语句实现的，试将其改为由 printf()函数实现，查看程序的执行结果有无变化。

(2) 在例 5-5 的程序中，字符串的输出操作是由“printf("%s\n",str);”语句实现的，试将其改为由 puts()函数实现，查看程序的执行结果有无变化。

5.2.2　多字符串操作函数

多字符串操作函数具有较复杂的函数原型，其函数类型、参数类型必须使用指针类型进行描述，相关知识迄今尚未介绍。本节仅就已学知识简单介绍字符串操作函数的基本用法，函数原型请参阅附录 B。

1. 字符串连接函数 strcat()

使用格式：

```
strcat(s1,s2)
```

函数功能：将字符串 s2 连接到字符串 s1 的后面。

说明：

(1) s1 是字符数组名或字符数组的开始地址，s2 既可以是字符数组名，也可以是字

符串。

(2) 函数在执行之后,s1是连接之后的字符串,s2保持不变。在定义s1数组时,其数组长度应不小于两个字符串的长度之和。

【例5-6】 将两个字符串连接为一个新字符串,并输出该字符串。

程序如下:

```
#include<stdio.h>
#include<string.h>
int main()
{
    char c1[20]="China",c2[10]= "man"; /*定义字符型数组并初始化*/
    strcat(c1,c2);                     /*将c2存储的字符串连接到c1存储的字符串之后*/
    printf("String c1: ");
    puts(c1);                          /*输出字符串c1*/
    printf("String c2: ");
    puts(c2);                          /*输出字符串c2*/
    return 0;
}
```

执行结果:

```
String c1: Chinaman
String c2: man
```

问题思考

在上面的程序中,连接使用的两个字符串是在程序内部定义的,因此该程序并没有通用性。若要求通过键盘输入两个字符串,然后把它们连接起来,应该怎样修改程序?

2. 字符串复制函数 strcpy()

使用格式:

```
strcpy(s1,s2)
```

函数功能:把字符串s2复制到字符数组s1中。

说明:

(1) s1是字符数组名或字符数组的开始地址;s2可以是数组名或字符数组的开始地址,也可以是一个字符串。s1不能是字符串。

(2) s1数组的长度应不小于s2的长度,以保证能够存储s2,否则会出现意想不到的错误结果。

【例5-7】 字符串复制示例。

程序如下:

```
#include<stdio.h>
#include<string.h>
int main()
{
    char c1[20]="program",c2[10]="example";
    strcpy(c1,c2);                 /*把c2中的字符串复制到c1中,c1的原串被覆盖*/
    printf("String c1: ");
```

```
    puts(c1);                          /* 输出 c1 中的字符串 */
    printf("String c2: ");
    puts(c2);                          /* 输出 c2 中的字符串 */
    return 0;
}
```

执行结果：

```
String c1: example
String c2: example
```

3. 字符串比较函数 strcmp()

使用格式：

```
strcmp(s1,s2)
```

函数功能：比较字符串 s1 和字符串 s2 的大小。

说明：

(1) s1、s2 可以是字符数组名或字符数组的开始地址，也可以是字符串。

(2) 字符串比较就是比较字符串中字符的编码值(例如 ASCII 码值)，编码值大的字符串大。比较的方法是对两个字符串自左至右逐个字符比较，直到遇到不同字符或字符串结束标记'\0'时比较过程结束，此时编码值大的字符所在的字符串大。

(3) strcmp()函数返回一个数值。当 s1 与 s2 相同时，strcmp(s1,s2)的值为 0；当 s1 大于 s2 时，strcmp(s1,s2)的值为一个正数；当 s1 小于 s2 时，strcmp(s1,s2)的值为一个负数。

注意：字符串只能用 strcmp()函数比较，不能用关系运算符“==”比较。例如，对于字符串 s1、s2，若其相同时输出"yes"，应使用如下语句：

```
if(strcmp(s1,s2) == 0) printf("yes");
```

而下面的用法是错误的：

```
if (s1 == s2) printf("yes");
```

【例 5-8】 使用 strcmp()函数设计一个密码验证程序。

程序如下：

```
#include <stdio.h>
#include <string.h>
#define N 3
int main()
{
    int count = 1;
    char word[12];
    while(count++<=N)
    {
        printf("Password: ");
        gets(word);                            /* 输入密码 */
        if(strcmp(word,"beijing2008") == 0)
            break;                             /* 比较成功则终止循环 */
```

```
    }
    if(count > N + 1)                          /* 对结束循环的情况进行判断 */
        printf("Sorry!\n");                    /* 连续3次输入错误 */
    else
        printf("Continue,please!\n");          /* 输入了正确密码 */
    return 0;
}
```

程序内设的密码是字符串"beijing2008"。当要求用户输入口令时,如果用户能正确地输入该字符串,则视为合法用户,可以继续运行程序;若用户连续3次都不能正确地输入该字符串,则视为非法用户,不能继续使用程序。

问题思考

(1) 在执行上述程序时,若密码在第3次输入时才正确,结束while循环后count的值是多少?

(2) 若连续3次都不能正确地输入密码,结束while循环后count的值是多少?

4. 其他字符串操作函数

除上面介绍的字符串操作函数以外,字母的大小写转换函数、求字符串长度函数等也是常用的字符串操作函数,表5-1是关于这几个函数的基本描述。

表5-1 其他常用的字符串操作函数

函数及用法	函数功能	说 明
strlwr(s)	将字符串s中的大写字母转换为小写字母	s可以是字符数组名(字符串首地址),也可以是字符串常量
strupr(s)	将字符串s中的小写字母转换为大写字母	
strlen(s)	求字符串s的长度	

5.3 二维数组程序设计

数组分为一维数组、二维数组和多维数组,不同维数的数组既具有共同的性质,又具有各自不同的特点。本节对二维数组程序设计的基本知识进行介绍。

5.3.1 二维数组的定义及元素引用

1. 二维数组的定义

二维数组的数据排列通常具有如下形式:

26	38	19	55
21	17	66	18
29	65	16	29

横向的每一组数称为数组的一行,纵向的每一组数称为数组的一列。如果要定义二维数组,除了要说明它的元素的数据类型、数组名以外,还需说明数组的行数和列数。

二维数组的一般定义格式如下:

```
数据类型 数组名[表达式1][表达式2];
```

例如：

```
int a[3][4];
```

该语句定义了数组名为 a 的 int 型二维数组，该数组有 3 行 4 列，共 12 个数组元素，每个数组元素均要用两个下标进行标识。如下所示：

a[0][0]　a[0][1]　a[0][2]　a[0][3]
a[1][0]　a[1][1]　a[1][2]　a[1][3]
a[2][0]　a[2][1]　a[2][2]　a[2][3]

说明：

(1) 二维数组的数据类型的说明与一维数组相同，都是指数组中每个元素的数据类型。

(2) “表达式 1”用来定义二维数组的行数，“表达式 2”用来定义二维数组的列数，“表达式 1”和“表达式 2”可以是整数型常量，也可以是由常量构成的整数型表达式，但不允许是变量表达式。例如，下面对数组的定义方法是错误的：

```
int m = 5,n = 10;
float b[m][n];
```

2. 二维数组元素的引用

二维数组元素的引用形式如下：

```
数组名[下标 1][下标 2]
```

其中，“下标 1”和“下标 2”允许是任何形式的整数型表达式，分别表示数组元素所在的行号和列号。C 语言规定，二维数组的行下标和列下标都从 0 开始编号。对于上述 a 数组，行下标的取值范围为 0～2，列下标的取值范围为 0～3。

通常使用 a[i][j]表示二维数组 a 的任意元素，当指定 i、j 为特定值时，a[i][j]则为数组的一个特定元素。例如，当 i＝0、j＝0 时，a[i][j]即为 a[0][0]；当 i＝0、j＝1 时，a[i][j]即为 a[0][1]；以此类推。

用户也可以用一维数组的观点看待一个二维数组。对于 M 行 N 列的二维数组，可将其视为有 M 个元素的一维数组，其中每个数组元素又是一个具有 N 个元素的一维数组。例如一个 2 行 3 列的二维数组 a，可以将其视为包含 a[0]、a[1]两个元素的一维数组，而 a[0]、a[1]又是各包含 3 个元素的一维数组，其中 a[0]的 3 个元素为 a[0][0]、a[0][1]、a[0][2]，a[1]的 3 个元素为 a[1][0]、a[1][1]、a[1][2]，此时可以将 a[0]、a[1]视为数组名。

5.3.2　二维数组的输入和输出

二维数组的输入和输出是通过每个二维数组元素的输入和输出实现的。当数组元素是简单变量时，与简单变量的输入和输出方法相同。

例如，对于上述 a 数组，输入 a[1][2]的值时使用如下语句：

```
scanf("%d",&a[1][2]);
```

输出 a[1][2]的值时使用如下语句：

```
printf("%d", a[1][2]);
```

对于M行N列的二维数组,如果要访问它的每个元素,一般使用双重循环实现。

【例5-9】 有一个3行4列的二维数组,从键盘输入其前两行各元素的数据,并将这两行的数组元素按列求和的结果对应存储在第3行的各元素中。

例如,设以下两行数据为该二维数组前两行的数据:

10 20 30 40
15 25 35 45

按题目要求,首先将这两行数据对应输入到二维数组前两行的各元素中,然后按列求和,填充第3行的数据。以下是填充数据之后数组的最终结果:

10 20 30 40
15 25 35 45
25 45 65 85

程序如下:

```
#include<stdio.h>
int main()
{
    int a[3][4],i,j;
    printf("Input data:\n");
    for(i=0;i<2;i++)                    /*输入前两行的数据*/
        for(j=0;j<4;j++)
            scanf("%d",&a[i][j]);
    for(j=0;j<4;j++)
        a[2][j]=a[0][j]+a[1][j];        /*填充第3行的元素*/
    printf("Result:\n");
    for(i=0;i<3;i++)                    /*输出a数组的全部元素*/
    {
        for(j=0;j<4;j++)
            printf("%d ",a[i][j]);
        printf("\n");                   /*每行输出结束后换行*/
    }
    return 0;
}
```

程序执行结果:

```
Input data:
10 20 30 40 15 25 35 45↵
Result:
10  20  30  40
15  25  35  45
25  45  65  85
```

该程序有两个双重循环,分别实现a数组的输入和输出。第1个双重循环的内循环体是“scanf("%d",&a[i][j]);”语句,该语句共执行8次,依次为数组元素a[0][0]、a[0][1]、a[0][2]、a[0][3]、a[1][0]、a[1][1]、a[1][2]、a[1][3]输入数据。在输入数据时需按照这一顺序对应输入,即按行逐列输入。数组a的输出是由后一个双重循环实现的,输出结果分行显示。程序中间的一个for语句计算第3行每个元素的值,其循环体语句“a[2][j]=

a[0][j]+a[1][j];”共执行 4 次，依次为元素 a[2][0]、a[2][1]、a[2][2]、a[2][3]赋值，即填充。

5.3.3 二维数组的初始化

二维数组的初始化是在定义二维数组时为数组元素赋初值，既可对数组的全部元素初始化，也可对数组的部分元素初始化。

1. 按行初始化

按行初始化的思想是把二维数组的一行当作一个一维数组对待，每行提供一个独立的数据集合。例如：

```
int a[2][3] = {{1,2,3},{4,5,6}};
```

初值部分的{1,2,3}对应数组 a[0]行的 3 个元素，{4,5,6}对应数组 a[1]行的 3 个元素，每行元素的初始化方式与一维数组相同。初始化后 a 数组的各元素值如下：

```
a[0][0] = 1、a[0][1] = 2、a[0][2] = 3、a[1][0] = 4、a[1][1] = 5、a[1][2] = 6
```

按行初始化时，也可以只对二维数组的部分元素初始化。例如：

```
int a[2][3] = {{1,2},{4}};
```

数组的 a[0]行只有两个初值，按顺序分别赋给 a[0][0]和 a[0][1]；数组的 a[1]行只有一个初值 4，赋给 a[1][0]。

2. 按行逐列初始化

按行逐列初始化是二维数组常用的初始化方式，它把提供的初始化数据按照逐行逐列的顺序依次赋给对应的数组元素。例如：

```
int b[3][2] = {10,20,30,40,50,60};
```

大括号“{ }”中的 6 个数据依次赋给 b 数组的元素 b[0][0]、b[0][1]、b[1][0]、b[1][1]、b[2][0]、b[2][1]。

按行逐列初始化也可以只对部分数组元素初始化。例如：

```
int b[3][2] = { 10,20,30};
```

大括号“{}”内只有 3 个初值，对应赋给元素 b[0][0]、b[0][1]、b[1][0]。

3. 初始化时二维数组的行数定义部分允许省略

例如：

```
int a[][4] = {{1,2},{1,2,3}};
int b[][3] = {1,2,3,4,5,6,7,8,9};
```

数组 a 的初始化数据有两组，系统自动确定数组行数为 2；数组 b 的初始化数据共有 9 个，列数值为 3，即每行 3 个数，所以数组 b 的行数是 9/3=3，即数组 b 为 3 行 3 列。当数据总数不能被列数值整除时，数组行数值为商值加 1。例如：

```
int c[][3] = {1,4,5,6,8};
```

在该数组定义中,所给出的初始化数值的个数为5,不能被数组的列数值3整除,系统按商值加1的原则,将数组c的行数定义为2,即数组c为2行3列的二维数组。

读者务必注意,不管用哪种方式对二维数组初始化,数组列数的定义都是不能省略的,即第2个维数不能省略。

【例5-10】 一个4×4数组a具有如下性质:数组元素a[i][j]和a[j][i]对应相等。已知a的下三角区域的元素值如下,编写程序填充数组的其他元素。

$$\begin{bmatrix} 1 & & & \\ 6 & 1 & & \\ 8 & 7 & 1 & \\ 9 & 5 & 3 & 1 \end{bmatrix}$$

程序如下:

```
#include<stdio.h>
int main()
{
    int a[4][4]={{1},{6,1},{8,7,1},{9,5,3,1}}; /*部分元素初始化*/
    int i,j;
    for(i=0;i<3;i++)
        for(j=i+1;j<4;j++)
            a[i][j]=a[j][i];                    /*利用下三角元素填充上三角元素*/
    for(i=0;i<4;i++)                            /*输出填充后的二维数组a*/
    {
        for(j=0;j<4;j++)
            printf("%4d",a[i][j]);
        printf("\n");
    }
    return 0;
}
```

程序执行结果:

```
1  6  8  9
6  1  7  5
8  7  1  3
9  5  3  1
```

请读者对照结果分析程序的执行过程。

5.3.4 二维数组的存储

二维数组在计算机中存储时,计算机按照二维数组的大小分配一段连续的内存空间,逐一存储二维数组的各元素,各元素的存储采用按行逐列的顺序。例如,对于m×n的二维数组a,各元素的存储次序如下:

a[0][0]、a[0][1]…a[0][n-1]、a[1][0]、a[1][1]…a[1][n-1]…a[m-1][0]、a[m-1][1]…a[m-1][n-1]

系统为其分配的存储单元数为 m×n×每个元素占用的存储单元数。

其中，每个元素占用的存储单元数取决于数组的数据类型和使用的 C 语言系统。

例如，2×2 数组 example 的存储情况如图 5-7 所示。

数组所占存储空间的首单元地址称为数组的首地址，该地址可直接使用数组名表示。数组名的这一性质在使用指针处理数组时被广泛应用。

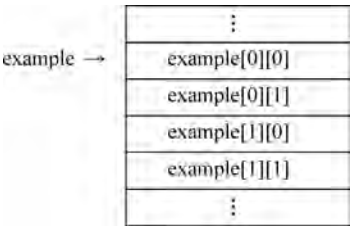

图 5-7　二维数组存储举例

5.4　数组应用程序实例

数组在批量数据处理中有重要的应用，本节介绍的数组程序设计均是一维数组和二维数组应用的典型实例。

【例 5-11】 对 N 个整数进行升序排序，并输出排序结果。

1）问题分析与算法设计

排序是将一组数据按照一定的顺序排列起来，由小到大排列时称为升序排序，反之则为降序排序。在使用一维数组进行排序时，通常的过程是先将待排序的一组数据存储在数组中，然后使用一定的排序方法进行具体的排序操作。

排序的方法有多种，这里使用冒泡排序算法。实际上，例 5-2 中后移最大数的操作已经体现了冒泡排序的思想。冒泡排序的过程如下：对于给定的待排序数据，从头开始依次对相邻的两个数据进行比较，当前者大时两数交换位置，直到比较完最后一个数据，此时这些数据的最大值处于最末位置，这称为一趟比较。然后对其余数据重复这种比较过程，直到排序结束。对于 N 个数据的数列需进行 N－1 趟排序操作。

以下是对 5 个数据排序时每趟排序结束之后的情况，[]内的数据是按序定位的数据，不再参加下一趟的排序操作。

待排序数列：　6　28　21　－19　5

第 1 趟结束：　6　21　－19　5　[28]

第 2 趟结束：　6　－19　5　[21　28]

第 3 趟结束：　－19　5　[6　21　28]

第 4 趟结束：[－19　5　6　21　28]

从上述分析可知，冒泡排序需要用双重循环实现：外循环进行趟数控制，内循环将当前待排序数据中的最大数移到末尾。若趟数控制变量用 i 表示，数组的任一元素用 a[j]表示，在对 5 个数据进行排序时，每趟的待排序元素及其比较情况如图 5-8 所示，加阴影的元素是当前已经按序定位的元素，不再参加比较操作。

对照图 5-8 不难看出，若待排序元素数为 N，则趟数控制变量 i 的取值范围为 1～N－1。进行第 i 趟排序时，元素下标变量 j 的取值范围为 0～(N－i)－1。

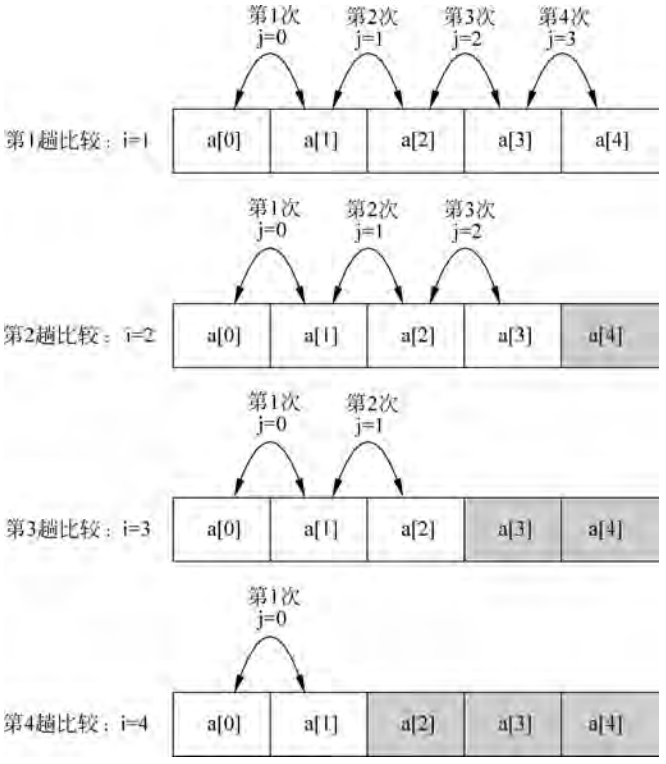

图 5-8　每趟待排序元素及其比较操作示意图

2）实现程序

程序如下：

```
#include<stdio.h>
#define N 10
int main()
{
    int a[N],i,j,temp;
    printf("Input data: ");
    for(i=0;i<N;i++)                    /*从键盘输入 N 个整数,存储在数组 a 中*/
        scanf("%d",&a[i]);
    for(i=1;i<N;i++)                    /*进行趟数控制,共比较 N-1 趟*/
        for(j=0;j<N-i;j++)              /*在每一趟中控制相邻元素两两比较*/
            if(a[j]>a[j+1])             /*相邻元素前者大、后者小时两元素值交换*/
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
    printf("Result: ");
    for(i=0;i<N;i++)                    /*输出排序结果*/
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

程序中第 2、第 3 个 for 语句是一个双重循环结构，用于实现数组元素的排序。其中的外循环进行趟数控制，内循环控制每一趟参加比较的数据区间，并在该区间内自始至终对相邻的元素进行两两比较，将较大数后移。每次内循环结束后，该数据区间中的最大值被移至该区间的最后一个元素位置。例如进行第 3 趟比较时，参加比较的数据区间为 a[0]至 a[2]，该趟比较结束时，该区间中的最大值被移至 a[2]元素位置，即 a[2]为该区间的最大值。

程序执行结果：

```
Input data: 10 20 19 21 9 6 15 8 12 16 ↵(输入数据)
Result: 6  8  9  10  12  15  16  19  20  21(排序后的数列)
```

拓展知识

类似上述程序，在调试时经常需要输入批量数据，低效而烦琐。此时可以利用 C 语言提供的随机数生成函数 rand()自动生成一批数据，用于程序调试。rand()函数的说明在头文件 stdlib.h 中，使用时要在程序开头添加宏命令#include<stdlib.h>。

下面是对 10 个随机数排序的完整程序。

```
#include<stdio.h>
#include<stdlib.h>
#define N 10
int main()
{
    int a[N],i,j,temp;
    printf("Data: ");
    for(i=0;i<N;i++)                    /*产生 N 个 100 以内的随机数,存储在数组 a 中*/
    {
        a[i]=rand()%100;
        printf("%4d",a[i]);             /*输出随机数*/
    }
    for(i=1;i<N;i++)
        for(j=0;j<N-i;j++)
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
    printf("\nResult:");
    for(i=0;i<N;i++)                    /*输出排序后的数组*/
        printf("%4d",a[i]);
    printf("\n");
    return 0;
}
```

程序执行结果：

```
Data:  41  67  34  0  69  24  78  58  62  64
Result:  0  24  34  41  58  62  64  67  69  78
```

【例 5-12】 在利用一维数组存储的一个升序数列中,使用折半查找法查找指定数值。

1) 问题分析与算法设计

查找是在一组数据中找指定的值,不同的数据对象有不同的查找方法。数据存储的结构不同,查找的方法也不同。为便于理解题意,先对查找的方法进行简要说明。

对于给定的一组数,查找 x 的方法有多种,最简单的方法是顺序查找,就是从头开始依次与 x 进行比较,当找到 x 时查找成功,查找过程结束;全部数据查找一遍,若未找到 x,则查找失败,查找过程结束。顺序查找方法简单,但效率低。折半查找是一种高效的查找方法,它要在一个有序的数列中进行,具体方法如下。

设有 N 个数据的数列已升序排序,存储在长度为 N 的一维数组 a 中,对查找范围内的数据设置 3 个特殊点,首位置为 top,末位置为 bot,中间位置为 mid=(top+bot)/2。查找初始,top 为数组首元素的下标值 0,bot 为数组末元素的下标值 N−1,mid 为中间位置元素的下标值(N−1)/2。以下为查找过程:

(1) 比较 x 与 a[mid],若 x 与 a[mid]相等,则查找成功,查找过程结束;否则,若 x>a[mid],则 top=mid+1,若 x<a[mid],则 bot=mid−1。

(2) 若 top>bot,则查找失败,查找过程结束;否则,mid=(bot+top)/2,转(1)。

图 5-9 所示为在数列{3,6,7,8,21,23,36,38,67,69,80,82,85,88,96}中查找 69 时 top、bot、mid 的变化情况。该查找过程经过 3 次之后结束,查找成功。图 5-9 中的①、②、③是对 3 次查找所做的标识,表 5-2 列出了查找过程中各项数据的动态变化情况。

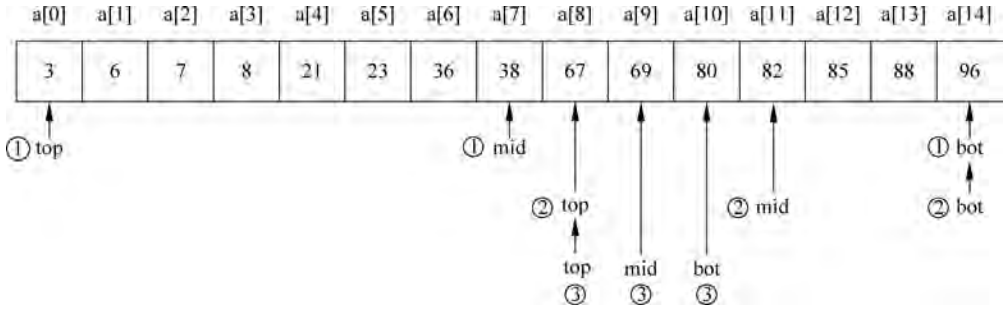

图 5-9 折半查找举例

表 5-2 折半查找过程动态信息表

查找过程	本次查找范围及中间位置			本次查找结果	下一次查找范围
	top	bot	mid		
第 1 次	0	14	7	a[mid]<x	在 mid 之后,top=mid+1
第 2 次	8	14	11	a[mid]>x	在 mid 之前,bot=mid−1
第 3 次	8	10	9	a[mid]==x	查找结束

上述查找过程的算法 N-S 图如图 5-10 所示。

2) 实现程序

程序如下:

```
#include<stdio.h>
```

```
#define N 15
int main()
{
    int a[N] = {3,6,7,8,21,23,36,38,67,69,80,82,85,88,96};
    int x,top,bot,mid;
    printf("Input x: ");
    scanf(" %d",&x);                    /* 输入要查找的数值 */
    top = 0;                            /* 设置初始查找边界 */
    bot = N - 1;
    do
    {
        mid = (top + bot)/2;            /* 确定中间位置 */
        if(a[mid] == x)
            break;                      /* 找到 x 后终止循环 */
        else if(a[mid]< x)
            top = mid + 1;              /* 确定后半段为下一次查找的范围 */
        else
            bot = mid - 1;              /* 确定前半段为下一次查找的范围 */
    }while(top < = bot);
    if(bot < top)                       /* 只有 x 不存在时 bot < top 才会成立 */
        printf(" %d: no found.\n",x);
    else
        printf("Success! a[ %d] is %d.\n",mid,x);
    return 0;
}
```

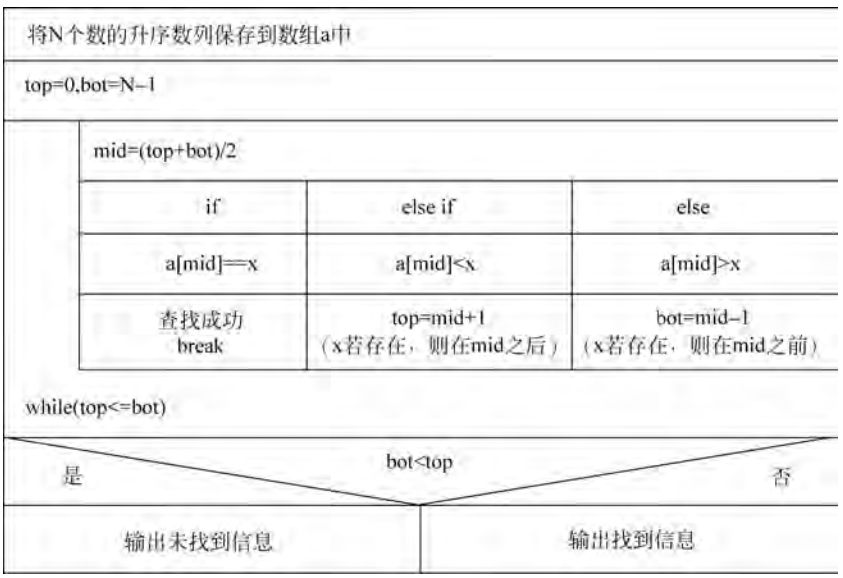

图 5-10　折半法查找 x 的算法 N-S 图

程序执行结果：

```
Input x: 69 ↵(第 1 次执行,在数列中查找 69)
Success!  a[9] is 69.(查找成功,元素 a[9]是 69)
```

```
Input x: 33 ↵(第 2 次执行,在数列中查找 33)
33: no found.  (在数列中未找到其值是 33 的数据)
```

在数列中查找 33 时,经过 4 次查找之后,循环条件 top<=bot 不再成立,查找过程结束。

【例 5-13】 输入一个字符串,统计其中单词的个数。

1) 问题分析与算法设计

在一个字符串中,把由空格分隔的一组连续字符视为一个单词,例如字符串"␣␣This ␣is ␣␣a ␣c ␣␣␣program."包含 5 个单词。分辨单词是对文本进行识别的基本操作。"单词统计"问题是字符型数组的一种典型应用。

如果要统计单词的个数,首先需要把单词找出来,这是进行单词统计最关键的环节。设长度是 n 的字符串含有的单词个数为 word,该字符串存储在字符数组 text 中,各字符元素分别为 text[0]、text[1]、text[2]、…、text[n-1]。

对于 text,有以下 3 种情况。

第 1 种情况:text 为空串,即没有输入任何字符,text[0]= '\0',word 的统计结果为 0。

第 2 种情况:text 的开始字符为空格符,即 text[0]= '␣',开始时没有出现单词,word 的初值置为 0。

第 3 种情况:text 的开始字符为非空格符,一开始就出现了单词,word 的初值置为 1。

对于非空字符串,设置 word 初值之后按以下步骤检测下一个单词:

(1) 若 text[i](i 初值为 1)不是字符串结束标记'\0',当满足下列条件时必然出现新单词:

```
text[i-1] == '␣'&&text[i]!= '␣'
```

(2) 执行"i++;",跳转到步骤(1),直到 text[i]为'\0'时结束。

2) 实现程序

程序如下:

```
#include<stdio.h>
#define N 100
int main()
{
    char text[N];                          /*设输入的字符串长度小于 N*/
    int word=0,i;
    printf("String: ");
    gets(text);
    if(text[0]!='\0')                      /*text 为非空字符串时进行单词统计*/
    {
        if(text[0]!=' ')word=1;            /*text 首字符是非空格符,置 word 初值为 1*/
        for(i=1;text[i]!='\0';i++)
            if(text[i-1]==' '&&text[i]!=' ')  /*满足条件时即遇到单词*/
                word++;
    }
    printf("Result: %d\n",word);
    return 0;
}
```

程序执行结果：

```
String: ␣␣This ␣is ␣␣a ␣c ␣␣␣program. ↵(字符串首字符为空格符)
Result:5
String: This ␣is ␣␣↵(字符串首字符不是空格符)
Result:2
String: ↵(直接按 Enter 键,输入空串)
Result:0
```

【例 5-14】 一个班级有 N 个学生，每个学生选修两门课程，实行百分制考核，要求分别统计各等级的人数，并将分等级统计的结果保存到一维数组中。分等级标准与第 4 章的例 4-9 相同。

1）问题分析与算法设计

第 4 章的例 4-9 讨论了“学生成绩分等级统计”问题，实现了一个班级学生成绩的分等级统计，各等级的统计结果分别用简单变量 r0、r1、r2、r3、r4 存储。若改用一维数组存储统计结果，程序会更简洁。

（1）学生成绩分为 5 个等级，因此可定义长度为 5 的 int 型数组 r，每个数组元素存储一个等级的统计结果。在例 4-9 的基础上改写程序时应将存储统计结果的各简单变量 r0、r1、r2、r3、r4 修改为相应的数组元素，其对应关系如表 5-3 所示。

表 5-3　统计变量对照表

对　应　项	存储各等级人数的数组元素	在例 4-9 程序中使用的变量
优秀人数	r[0]	r0
良好人数	r[1]	r1
中等人数	r[2]	r2
及格人数	r[3]	r3
不及格人数	r[4]	r4

（2）上述步骤完成后，各等级的统计结果存储到数组 r 中，将其各元素输出即可。

2）实现程序

程序如下：

```
#include<stdio.h>
#define N 6                                  /*班级人数*/
int main()
{
    int s1,s2,ave,i;
    static int r[5];
    for(i=0;i<N;i++)
    {
        printf("Score: ");
        scanf("%d,%d",&s1,&s2);              /*输入课程成绩 s1、s2*/
        ave=(s1+s2)/2;                       /*计算平均成绩*/
        if(ave>=90) r[0]++;                  /*统计优秀人数*/
        else if(ave>=80) r[1]++;             /*统计良好人数*/
        else if(ave>=70) r[2]++;             /*统计中等人数*/
        else if(ave>=60) r[3]++;             /*统计及格人数*/
```

```
            else r[4]++;                    /*统计不及格人数*/
        }
        for(i=0;i<5;i++)                    /*输出存储在数组r中的统计结果*/
            printf("%d",r[i]);
        printf("\n");
        return 0;
    }
```

该程序使用一维数组r存储统计结果,优秀、良好、中等、及格、不及格各等级人数分别存储在r[0]到r[4]中。

程序中定义保存统计结果的数组r时使用了“static int r[5];”语句,其中的static用于定义变量的存储类型,其作用是将r数组在使用前清零,即将其各元素初始化为0。有关变量存储类型的知识将在第6章进行介绍。

上述“学生成绩分等级统计”问题实现了一个班级学生成绩的分等级统计,以下是更进一步的问题——实现多个班级学生成绩的分等级统计。

【例5-15】 某年级共有3个班级,每班有N名学生,开设两门课程,要求分别对每个班级按照学习成绩进行分类统计,并将统计结果保存到一个二维数组中。

1) 问题分析与算法设计

该问题与上述“学生成绩分等级统计”问题的主要区别有两点:

(1) 对多个班级分别统计。

(2) 统计结果用二维数组保存。

对于第1点,可以通过在“学生成绩分等级统计”的基础上增加一个外重循环来实现。对于第2点,需要定义一个二维数组,存储数据的形式如表5-4所示。

表5-4 统计结果示意表

班级	优秀人数	良好人数	中等人数	及格人数	不及格人数
1班					
2班					
3班					

表5-4中空白处是要统计和存储的数据,因此可以定义一个3×5的二维数组r来保存统计结果,每个班级的统计结果存储在数组的一行上。例如,在对第1个班级进行统计时,优秀、良好、中等、及格和不及格这5个等级的统计结果应分别使用数组元素r[0][0]、r[0][1]、r[0][2]、r[0][3]和r[0][4]进行统计。

2) 实现程序

程序如下:

```
#include<stdio.h>
#define N 6                                 /*设每班有6名学生*/
int main()
{
    int s1,s2,ave,i,j;
    static int r[3][5];                     /*定义保存统计结果的二维数组*/
    for(i=0;i<3;i++)                        /*共有3个班级,用i控制*/
    {
```

```
        for(j = 1;j < = N;j++)                          /* 每个班级有 N 个学生,用 j 控制 */
        {
            printf("Class  %d score %d: ",i + 1,j);     /* 友好提示 */
            scanf(" %d, %d",&s1,&s2);                   /* 输入一个学生的两门课程成绩 */
            ave = (s1 + s2)/2;
            if(ave > = 90) r[i][0]++;                   /* 统计 i 班优秀人数 */
            else if(ave > = 80) r[i][1]++;              /* 统计 i 班良好人数 */
            else if(ave > = 70) r[i][2]++;              /* 统计 i 班中等人数 */
            else if(ave > = 60) r[i][3]++;              /* 统计 i 班及格人数 */
            else r[i][4]++;                             /* 统计 i 班不及格人数 */
        }
    }
    for(i = 0;i < 3;i++)                                /* 逐行输出各班级的统计结果 */
    {
        for(j = 0;j < 5;j++)                            /* 输出一个班级的各等级统计结果 */
            printf(" % 5d",r[i][j]);
        printf("\n");
    }
    return 0;
}
```

程序执行结果：

```
Class 1 score1: 87,91 ↵(输入 1 班的第 1 组数据)
Class 1 score2: 67,82 ↵(输入 1 班的第 2 组数据)
Class 1 score3: 56,72 ↵(输入 1 班的第 3 组数据)
Class 1 score4: 90,86 ↵(输入 1 班的第 4 组数据)
Class 1 score5: 92,89 ↵(输入 1 班的第 5 组数据)
Class 1 score6: 88,77 ↵(输入 1 班的第 6 组数据)
Class 2 score1: 67,87 ↵(输入 2 班的第 1 组数据)
Class 2 score2: 86,94 ↵(输入 2 班的第 2 组数据)
Class 2 score3: 51,61 ↵(输入 2 班的第 3 组数据)
Class 2 score4: 79,82 ↵(输入 2 班的第 4 组数据)
Class 2 score5: 66,60 ↵(输入 2 班的第 5 组数据)
Class 2 score6: 77,88 ↵(输入 2 班的第 6 组数据)
Class 3 score1: 71,81 ↵(输入 3 班的第 1 组数据)
Class 3 score2: 57,69 ↵(输入 3 班的第 2 组数据)
Class 3 score3: 87,81 ↵(输入 3 班的第 3 组数据)
Class 3 score4: 88,79 ↵(输入 3 班的第 4 组数据)
Class 3 score5: 67,69 ↵(输入 3 班的第 5 组数据)
Class 3 score6: 76,78 ↵(输入 3 班的第 6 组数据)
    1    3    1    1    0(1 班各等级统计结果)
    1    2    1    1    1(2 班各等级统计结果)
    0    2    2    2    0(3 班各等级统计结果)
```

程序的输出结果共有 3 行数据，分别是 3 个班级的学生成绩分等级统计情况，每列数据由前到后依次为优秀、良好、中等、及格和不及格的人数。

上述程序讨论的是各班级学生人数相同的情况，在通用性方面还有欠缺，可以进一步完善程序，使其适用于各班级人数不同的情况。例如定义一个一维数组 cla，用来存放各班级的人数，在输入一个班级的学生成绩时用 cla 的相应元素值作为成绩输入的循环控制量即能满足上述要求。

下面是实现程序（省略的代码是没有变动的程序段）：

```
#include<stdio.h>
int main()
{
    int s1,s2,ave,i,j;
    static int r[3][5];
    int cla[3];                                  /*数组各元素存储各班级的人数*/
    for(i=0;i<3;i++)                             /*输入各班级的人数*/
        scanf("%d",&cla[i]);
    for(i=0;i<3;i++)
    {
        for(j=1;j<=cla[i];j++)
        {
            …
        }
    }
    …
}
```

【例 5-16】 打印杨辉三角形的前 N 行数据。

1) 问题分析与算法设计

杨辉三角形是中国古代数学研究的一项重要发现,它是一个数值组合图形,借助杨辉三角形能求解二项式问题。杨辉三角形的结构如下:

```
1
1  1
1  2   1
1  3   3   1
1  4   6   4   1
1  5  10  10   5   1
```

若用 y(i,j)表示杨辉三角形中第 i 行、第 j 列的数值,则有:

$$y(i,j)=\begin{cases}1 & (j=1 \text{ 或 } j=i)\\ y(i-1,j-1)+y(i-1,j) & (j\neq 1 \text{ 且 } j\neq i)\end{cases}$$

根据杨辉三角形的数据排列特点,可以使用一个 N 行 N 列的二维数组 y 存储杨辉三角形,这样求解杨辉三角形的问题即转化为数组元素值的计算问题。

由于二维数组的行、列起始下标均从 0 开始,则对于第 i 行、第 j 列的元素 y[i][j]应按以下公式求值:

$$y[i][j]=\begin{cases}1 & (j=0 \text{ 或 } j=i)\\ y[i-1][j-1]+y[i-1][j] & (j\neq 0 \text{ 且 } j\neq i)\end{cases}$$

在编程实现时只使用 y 数组的下三角形元素,利用杨辉三角形每行首、尾元素值为 1 的特性对 y 数组赋初值,其他元素值经计算获得。

2) 实现程序

程序如下:

```
#include<stdio.h>
```

```
#define N 6
int main()
{
    int i,j,y[N][N];
    printf("\n");
    for(i=0;i<N;i++)                    /*生成杨辉三角形每行的首、尾元素值*/
    {
        y[i][0]=1;
        y[i][i]=1;
    }
    for(i=2;i<N;i++)                    /*生成杨辉三角形的其他元素值*/
        for(j=1;j<i;j++)
            y[i][j]=y[i-1][j-1]+y[i-1][j];
    for(i=0;i<N;i++)                    /*输出杨辉三角形*/
    {
        for(j=0;j<=i;j++)
            printf("%5d",y[i][j]);
        printf("\n");
    }
    return 0;
}
```

问题思考

杨辉三角形的呈现形式有多种，以下是一种较典型的杨辉三角形形式：

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
```

怎样修改上述程序可以使其输出该形式的杨辉三角形？

小　　结

本章介绍了关于数组的程序设计知识，主要有数组的特点、数组的定义及使用方法、数组在实际问题中的应用等，一维数组是本章的重点内容。

（1）数组是一种重要的数据结构，适合同类型批量数据的存储处理，因此在实际问题中有广泛的应用。

（2）数组与循环结构关系密切，数组的输入与输出、数组元素的运算等通常在循环体中进行，熟练掌握循环控制结构是进行数组程序设计的基础。

（3）数组分为一维数组、二维数组等，一维数组的定义格式为“数据类型　数组名[数组长度]”，二维数组的定义格式为“数据类型　数组名[数组行数][数组列数]”。无论数组的维数是多少，每一维的下标值都从0开始。

（4）数组初始化的方式有多种，可以对全部元素初始化，也可以对部分元素初始化。对于一维数组，若对全部元素初始化，数组长度说明可以省略；对于二维数组，只允许省略行

数的说明,在任何情况下都不能省略对列数的说明。

(5) 字符型数组用于存储字符串,它具有数组的一切性质,但又有其特点。字符型数组的操作对应大量的操作函数,例如字符串的复制、连接、比较等均有专门的函数予以实现。

(6) 数组的存储占用连续的内存空间,占用的存储单元数由数组长度和数组的数据类型确定。一维数组从首元素开始依次存储,二维数组的每一行是一个一维数组,在存储时从首行开始按行依次存储。数组名代表数组的首地址,该性质在后续程序设计中被广泛应用。

(7) 本章最后通过几个典型实例,系统介绍了数组应用程序的设计方法和过程。

习 题 五

一、选择题

1. 若有数组定义 int m[][2]={1,3,5,7,9},则以下叙述正确的是_____。

A. 该定义存在语法错误

B. 该定义等价于 int m[3][2]={1,3,5,7,9}

C. 该定义等价于 int m[][2]={{1,3,5},{7,9}}

D. 该定义等价于 int m[2][2]={1,3,5,7,9}

2. 对两个数组 a 和 b 进行如下初始化:

```
char a[] = {'a','b','c','d','e','f'};
char b[] = "abcdef";
```

则以下叙述正确的是_____。

A. a 数组与 b 数组完全相同

B. a 数组与 b 数组具有相同的长度

C. a 数组和 b 数组的最后一个字符都是字符串结束标识符'\0'

D. a 数组的长度比 b 数组的长度小

3. 有程序段如下:

```
int a[5],i;
for(i = 0;i < 5;i++)
   scanf(" %d",&a[i]);
```

要求通过键盘为数组 a 输入数据,在下面给出的数据输入形式中_____会使 a 数组获得无效数据。

A. 30 90 70 20 60 ↵

B. 30,90,70,20,60 ↵

C. 30 90 70 ↵
   20 60 ↵

D. 30 ↵90 ↵70 ↵20 ↵60 ↵

4. 有定义如下:

```
int a[10] = {3,5,7,9,8,4,21,10,6,15},t;
```

要求将数组 a 的首、尾元素交换,以下交换方式正确的是_____。

A. a[0]=a[9],a[9]=a[0];

B. t=a[1],a[1]=a[10],a[10]=t;

C. t=a[10],a[10]=a[1],a[1]=t;

D. t=a[0],a[0]=a[9],a[9]=t;

5. 以下数组定义正确的是_____。

A. int n=10,a[n];

B. int a[3][]={1,2,3,4,5,6,7};

C. char name[20]="liming";

D. int name[20]="liming";

6. 下列关于字符数组的描述中,错误的是_____。

A. 字符数组可以存放字符串

B. 字符数组中的字符串可以整体输入与输出

C. 在定义字符数组时可以使用一个字符串对其初始化

D. 可以用关系运算符对字符数组中的字符串进行比较

7. 下列程序执行后的输出结果是_____。

```
#include<stdio.h>
#include<string.h>
int main()
{
    char arr[2][4];
    strcpy(arr[0],"you");
    strcpy(arr[1],"me");
    arr[0][3]='&';
    printf("%s\n",arr[0]);
    return 0;
}
```

A. you　　B. me　　C. you&me　　D. me&you

8. 在下列程序段中不能输入字符串的是_____。

A.
```
char str[10];
puts(gets(str));
```

B.
```
char str[10];
scanf("%s",str);
```

C.
```
char str[10];
gets(str);
```

D.
```
char str[10];
getchar(str);
```

9. 以下是一个字符串输入与输出程序:

```
#include<stdio.h>
#include<string.h>
int main()
{
    char ch1[10],ch2[10];
    gets(ch1);
    gets(ch2);
    if(strcmp(ch1,ch2)>0)
        puts(ch1);
    else
        puts(ch2);
    return 0;
}
```

下列关于该程序功能的描述正确的是_____。

A. 输入两个字符串,然后按照输入顺序依次输出这两个字符串

B. 输入两个字符串,将其中的大字符串输出

C. 输入两个字符串,将其中的小字符串输出

D. 输入两个字符串,将其中最长的字符串输出

10. 以下是一个字符串输入与输出程序:

```
#include<stdio.h>
#include<string.h>
int main()
{
    char ch1[10],ch2[10];
    gets(ch1);
    gets(ch2);
    if(strlen(ch1)>strlen(ch2))
        puts(ch1);
    else
        puts(ch2);
    return 0;
}
```

下列关于该程序功能的描述正确的是_____。

A. 输入两个字符串,将其中的大字符串输出

B. 输入两个字符串,将其中的小字符串输出

C. 输入两个字符串,将其中的长字符串输出

D. 输入两个字符串,将其中的短字符串输出

二、程序分析题

1. 下面程序的功能是输出数组 s 中最大元素的下标。在横线处填上适当的内容,使它能输出正确的结果。

```
#include<stdio.h>
int main()
{
    int k,p,s[]={1,-9,7,2,-10,3};
    for(p=0,k=p;p<6;p++)
       if(s[p]>s[k])_____;
    printf("%d\n",k);
    return 0;
}
```

2. 下面程序的功能是将一个字符串中的小写英文字母全部改成大写形式,然后输出。在横线处填上适当的程序代码,使它能输出正确的结果。

```
#include<stdio.h>
int main()
{
    int i=0;
    char str[80];
    scanf("%s",str);
```

```
    while (    ①    )
    {
        if(    ②    ) str[i] = str[i] - 32;
        ____③____;
    }
    printf("%s\n", str);
    return 0;
}
```

3. 下列程序的输出结果是_____。

```
#include <stdio.h>
int main()
{
    int a[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i,j,s = 0;
    for(i = 1;i < 3;i++)
        for(j = 0;j < i;j++)
            s += a[i][j];
    printf("%d\n",s);
    return 0;
}
```

4. 下面的程序将二维数组 a 的行元素和列元素互换，然后存储到另一个二维数组 b 中。在横线处填上合适的程序代码。

```
#include <stdio.h>
int main()
{
    int a[2][3] = {{10,20,30},{40,50,60}};
    int b[3][2],i,j;
    for(i = 0;i <= 1;i++)
    {
        for(j = 0;j <= 2;j++)
        {
            printf("%5d",a[i][j]);
            _______;
        }
        printf("\n");
     }
     for(i = 0;i <= 2;i++)
     {
         for(j = 0;j <= 1;j++)
         printf("%5d",b[i][j]);
         printf("\n");
    }
    return 0;
}
```

三、编程题

1. 请回忆最近 5 次的网购情况，凡是“超值”的记为 1，“一般”的记为 0，如表 5-5 所示。编写程序，将超值记录数据按照网购顺序依次输入一维数组中，然后判断和输出网购评价结果。

表 5-5 超值网购评价表

网购顺序	超值记录
第1次	1
第2次	0
第3次	1
第4次	1
第5次	0

对应表 5-5 的网购数据将输出以下结果。

```
第1次,超值
第2次,一般
第3次,超值
第4次,超值
第5次,一般
```

2. 将 Fibonacci 数列前 20 项中的偶数值找出来,存储到一维数组中。

3. 某就业指导网站对 20 个行业进行了网络调查,并发布了每个行业的平均月薪。试编写程序,将这些数据输入一维数组中,然后将超过平均值的数据单独存储到一个高收入行业数组中。

4. 某研究小组共有 6 人,每个人的年龄按照从小到大的顺序存储到一维数组中。要求编写程序,计算他们的平均年龄,并确定最接近且不大于平均年龄的那个值在数组中的位置。

5. 某研究团队共有 N 个成员,将每个人的年龄随机存储到一维数组中。要求编写程序,计算他们的平均年龄,并确定最接近且不大于平均年龄的那个值。

6. 回文是顺读和倒读都一样的字符串,例如 ASDFDSA 是回文,而 ASDFDAS 不是回文。输入一个字符串,判断其是否为回文。若是回文,输出 Yes,否则输出 No。

7. 将一行电文译成密码,规律如下:

(1) 将 a、b、c、…、z 这 26 个小写字母分别译成 0、1、…、9、a、b、c、d、e、f、g、h、i、@、#、$、%、&、!、*;

(2) 将空格译成 j;

(3) 其他字符不变。

8. 打印点阵图案。先用纸笔绘制一个 N 行 N 列的点阵图案,并将图案信息输入一个二维数组中(例如,黑点用 1 表示,空白点用 0 表示),然后将图案打印出来(例如,对应 1 打印 *,对应 0 打印空格)。

9. 某人发表一篇英文短文,共有 5 段,每段有不超过 80 个字母,每个字符(不含空格)计稿费 0.5 元,编程计算共获得多少稿费。

10. 用一维数组作为存储结构,实现 Josephus 环报数游戏。以下是对 Josephus 环报数游戏的描述:n 个人围成一圈,从 1 开始顺序编号到 n,首先自 1 号开始顺时针从 1 报数,数到 m 者自动出列,然后自下一个人开始重新从 1 报数,数到 m 者仍然自动出列,直到最后一个人出列为止。编写程序,输出这 n 个人出列的顺序。

11. 鞍点问题。在二维数组中,若某一位置的元素值在该行中最大,而在该列中最小,则该元素即为该二维数组的一个鞍点。要求从键盘输入一个二维数组,当鞍点存在时把鞍点找出来。