第5章 数 组

数组是一个由若干同类型变量组成的集合，引用这些变量时可用同一名字。数组均由连续的存储单元组成，最低地址对应于数组的第一个元素，最高地址对应于最后一个元素，数组可以是一维的，也可以是多维的。

5.1 一维数组

一维数组的一般说明形式如下：

```
type-specifier var_name [size];
```

在C语言中，数组必须显示地说明，以便编译程序为它们分配内存空间。在上式中，类型说明符指明数组的类型，也就是数组中每一个元素个数，一维数组的总字节数可按下式计算：

sizeof(类型)* 数组长度 = 总字节数

[例5-1] 将数字0到9装入一个整型数组。

```
main( )
{
 int x[10];  /* 定义包含10个整型数的数组，引用为x[0]，x[1]...x[9]*/
 int t ;
 for (t=0; t<10;++t)  x[t]=t;
}
```

C语言并不检验数组边界，因此，数组的两端都有可能越界而使其它变量的数组甚至程序代码被破坏。在需要的时候，数组的边界检验便是程序员的职责。例如，当使用 gets( )接收字符输入时，必须确认字符数组的长度足以存放最长的字符串。

一维数组在本质上是由同类数据构成的表，例如，对下列数组 a:

```
char a[7]
```

图5-1说明了数组a在内存中的情形，假定起始地址为 1000。

元素	0	1	2	3	4	5	6
地址	1000	1001	1002	1003	1004	1005	1006

图5-1 起始地址为1000的 7 元素字符数组

5.1.1 向函数传递一维数组

将一维数组传递给函数时，把数组名作为参数直接调用函数即可，无需任何下标。这样，数组的第一个元素的地址将传递给该函数。 C语言并不是将整个数组作为实参来传递，而是用指针来代替它。例如，下面的程序将数组 i的第一个元素的地址传递给函数 func1( )。

```
main( )
```

```
{
 int i[10];
 func1(i);     /*函数调用，实参是数组名*/
    .
    .
    .
 }
```

函数若要接收一维数组的传递，则可以用下面的二种方法之一来说明形式参数； 1) 有界数组；2) 无界数组。例如，函数func1 ( )要接收数组i可如下说明:

```
func1(str)
char str[10];  /*有界数组，数组的下标只能小于或等于传递数组的大小。*/
{
  .
  .
  .
}
```

也可说明为:

```
func1(str)
char str[ ]; / *无界数组 * /
{
  .
  .
  .
}
```

这二种说明方法的效果是等价的，它们都通知编译程序建立一个字符指针。第一种说明使用的是标准的数组说明；后一种说明使用了改进型的数组说明，它只是说明函数将要接收一个具有一定长度的整型数组。细想就会发现，就函数而言，数组究竟有多长并无关紧要，因为C语言并不进行数组的边界检验。事实上，就编译程序而言，下面的说明也是可行的。

```
func1 (str);
int str[32];
{
 .
 .
 .
}
```

因为编译程序只是产生代码使函数func1( )接收一个指针，并非真正产生一个包含32个元素的数组。

5.1.2 字符串使用的一维数组

显然，一维数组的最普通的用法是作为字符串。在C语言中，字符串被定义为一个以空字符终结的字符数组。空字符以‘ \0 ’来标识，它通常是不显示的。因此，在说明字符数组时，必须比它要存放的最长字符串多一个字符。例如，假如要定义一个存放长度为 10的字符串的数组s，可以写成:

```
char s[11];
```

这样就给字符串末尾的空字符保留了空间。

尽管C语言并不把字符串定义为一种数据类型，但却允许使用字符串常量。字符串常量是由双引号括起来的字符表。例如，下面两个短语均为字符串常量：

```
"hello there"
"this is a test"
```

不必向字符串的末尾加空字符，C编译程序会自动完成这一工作。

C语言支持多串操作函数，最常用的有：

名字	功能
strcpy(s1 s2)	将s2拷贝到s1
strcat(s1 s2)	将s2连接到s1的末尾
strlen(s1)	返回s1的长度
strcmp(s1，s2)	若s1与s2相等，返回值为0
	若s1<s2，返回值小于0
	若s1>s2，返回值大于0

例5-2说明了这些函数的用法。

[例5-2]

```
# include <stdio.h>
main ( )
{
 char s1[80], s2[80];    /*定义字符数组*/
 gets (s1);     /*输入字符串*/
 gets (s2);
 printf ("lengthsf: %d %d \n", strlen(s1)  , strlen(s2));
 if (!strcmp(s1, s2))
     printf ("the strings are equal \n");
 strcat(s1  , s2);
 printf ("%s\n", s1);
}
```

切记，当两个串相等时，函数 strcmp( )将返回False，因而当测试串的等价性时，要像前例中的那样，必须用逻辑运算符！将测试条件取反。

当程序运行并以“hello”和“hello”这两个串作为输入时，其输出为：

```
RUN
hello
hello
lengths:5  5
The strings are equal
hellohello
```

5.2 二维数组

5.2.1 二维数组的一般形式

C语言允许使用多维数组，最简单的多维数组是二维数组。实际上，二维数组是以一维数

组为元素构成的数组，要将d说明成大小为（10，20）的二维整型数组，可以写成：

```
int d[10][20]
```

请留心上面的说明语句，C不像其它大多数计算机语言那样使用逗号区分下标，而是用方括号将各维下标括起，并且，数组的二维下标均从0计算。

与此相似，要存取数组d中下标为（3，5）的元素可以写成：

```
d[3][5]
```

在例5-3中，整数1到12被装入一个二维数组。

[例5-3]

```
main ( )
{
 int t, i, num[3][4]
 for (t=0; t<3; ++t)
   for (i=0;i<4;++i)
num[t][i]=(t*4)+i+1;
 }
```

在此例中，num[0][0]的值为1，num[0][2]的值为3，.....，num[2][3]的值为12。可以将该数组想象为如下表格：

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

二维数组以行—列矩阵的形式存储。第一个下标代表行，第二个下标代表列，这意味着按照在内存中的实际存储顺序访问数组元素时，右边的下标比左边的下标的变化快一些。图5-2是一个二维数组在内存中的情形，实际上，第一下标可以认为是行的指针。

记住，一旦数组被证明，所有的数组元素都将分配相应的存储空间。对于二维数组可用下列公式计算所需的内存字节数：

行数×列数×类型字节数＝总字节数

因而，假定为双字节整型，大小为（10，5）的整型数组将需要：10×5×2=100 字节

当二维数组用作函数的参数时，实际上传递的是第一个元素（如[0][0]）的指针。不过该函数至少得定义第二维的长度，这是因为C编译程序若要使得对数组的检索正确无误，就需要知道每一行的长度。例如，将要接收大小为（10，10）的二维数组的函数，可以说明如下：

```
func1 (x)
int x[ ][10]
{
  .
  .
  .
}
```

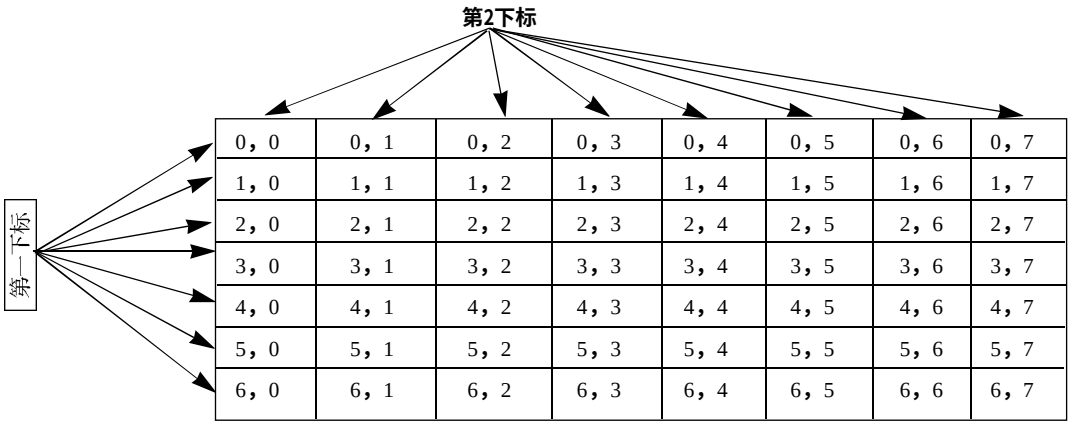

图5-2 内存中的二维数组

第一维的长度也可指明，但没有必要。

C编译程序对函数中的如下语句：

```
X[2][4]
```

处理时，需要知道二维的长度。若行长度没定义，那么它就不可能知道第三行从哪儿开始。

[例5-4] 用一个二维数组存放某一教师任教的各班学生的分数。假定教师有三个班，每班最多有三十名学生。注意各函数存取数组的方法。

```
#define classes 3
#define grades 30
#include <stdio.h>
main( )
{
  void enter_grades( );
  void disp_grades( );
  int get_grade( );
  int a[classes] [grades];定义二维数组，每行存放一个班学生成绩 */
  char ch;
  for( ; ;)
{
   do {  /*菜单显示*/
      printf("(E)nter grades\n");
      printf("(R)eport grades\n");
      printf("(Q)uit\n");
     ch=toupper(getchar());  将键盘输入字符转换为大写*/
     } while(ch!='E' && ch!='R' && ch!='Q');
   switch(ch)
{
     case 'E':
      enter_grades( );
      break;
     case 'R':
```

```
        disp_grades(grade);
        break;
      case 'Q':
        exit(0);
       }
     }
  }

void enter_grades(a)
int a[][grades];
{
  int t, i;
  for (t=0;t<classes;t++)
     {
       printf (" class #%d:\n",t+1);
       for (i=0; i<grades; i++)
          a[t][i]=get_grade(i);
       }
}
 int get_grades(num)
 int num;
 {
   char s[80];
   printf("enter grade for student # %d:\n",num+1);
   gets(s) ;/*输入成绩*/
   return(atoi(s));
  }
 void disp_grades(g)  /*显示学生成绩*/
 int g[ ][grades];
 {
  int t, i;
  for(t=0; t<classes; ++t) {
   printf("class # %d:\n",t+1);
   for(i=0;i<grades;++i)
     printf("grade for student #%d is %d\n",i+1 , g[t][i]);
  }
 }
```

我们将实际问题简化为共有2个班，每班两个学生，即将程序中的常量定义修改如下：

```
#define  classes 2
#define grades 2
```

运行程序：

```
RUN
(E)nter grades
(R)eport grades
(Q)uit: e
class #1:
enter grade for student #1:78
```

```
enter grade for student #289
class #2
enter grade for student #198
enter grade for student #290
(E)nter grades
(R)eport grades
(Q)uit: r
class #1
     grade for student #1 is 78
     grade for student #2 is 89
class #2
     grade for student #1 is 98
     grade for student #2 is 90
(E)nter grades
(R)eport grades
(Q)uit :q
```

运行程序，我们首先看到一个菜单，选择“ e ”输入成绩，选择“ r ”显示成绩，选择“ q ”退出。atoi()函数用于将实参字符串转换为整型。

5.2.2 字符串数组

程序设计中经常要用到字符串数组。例如，数据库的输入处理程序就要将用户输入的命令与存在字符串数组中的有效命令相比较，检验其有效性。可用二维字符数组的形式建立字符串数组，左下标决定字符串的个数，右下标说明串的最大长度。例如，下面的语句定义了一个字符串数组，它可存放30个字符串，串的最大长度为80个字符：

```
char str_array[30][80];
```

要访问单独的字符串是很容易的，只需标明左下标就可以了。例如，下面的语句以数组str_array中的第三个字符串为参数调用函数 gets( )。

```
gets(str_array[2]);
```

该语句在功能上等价于：

```
gets(&str_array[2][0]);
```

但第一种形式在专业程序员编制的C语言程序中更为常见。

为帮助理解字符串数组的用法，研究例5-5。它以一个字符串数组为基础做简单的文本编辑。

[例5-5]

```
#include <stdio.h>
#define MAX 100
#define LEN 80
char text [MAX][LEN]
 /* 一个非常简单的文本编辑器 */
main( )
{
  register int t, i, j;
  for(t=0;t<MAX; t++)     /逐行输入字符串*/
{
```

```
    printf("%d:"  , t);
    gets(text[t]);
    if(! text[t][0])
         break;            /*空行退出 */
  }
  for(i=0;i<t  , i++)              /按行，逐个字符输出字符串*/
  {
    for(j=0; text [i][j];j++)
        putchar(text [i][j]);
    putchar( '\n');
  }
  }
```

该程序输入文本行直至遇到一个空行为止，而后每次一个字符地重新显示各行。

5.3 多维数组

C语言允许有大于二维的数组，维数的限制（如果有的话）是由具体编译程序决定的。多维数组的一般说明形式为：

```
Type-specifier name [a][b][c]...[z];
```

由于大量占有内存的关系，二维或更多维数组较少使用。如前所述，当数组定义之后，所有的数组元素都将分配到地址空间。例如，大小为（10，6，9，4）的四维字符数组需要10×6×9×4即2160字节。

如果上面的数组是两字节整型的，则需要4320字节，若该数组是双字型的（假定每个双字为8字节）则需要34560字节，存储量随着维数的增加呈指数增长。

关于多维数组，需要注意一点：计算机要花大量时间计算数组下标，这意味着存取多维数组中的元素要比存取一维数组的元素花更多的时间。由于这些和其它原因，大量的多维数组一般采用C语言动态分配函数及指针的方法，每次对数组的一部分动态地分配存储空间。

多维数组传递给函数时，除第一维外，其它各维都必须说明。例如，将数组m定义成：

```
int m[4][3][6][5];
```

那么接收m的函数应写成：

```
func1 (d)
int d[][3][6][5];
```

当然，如果愿意，也可加上第一维的说明。

5.4 数组的初始化

5.4.1 数组初始化

C语言允许在说明时对全局数组和静态局部数组初始化，但不能对非静态局部数组初始化。与其它变量相似，数组初始化的一般形式如下：

```
type-specifier array_name[size1]...[sizen]={value-list};
```

数值表是一个由逗号分隔的常量表。这些常量的类型与类型说明相容，第一个常量存入

数组的第一个单元，第二个常量存入第二个单元，等等，注意在括号“ }”后要加上分号。

下列中一个10元素整型数组被初始化装入数字1到10：

```
int i[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

这意味着i[0]的值为1，而i[9]的值为10。

存放字符串的字符数组的初始化可采用如下简化的形式：

```
char array_name[size] = "string";
```

例如，以下代码段将str 初始化为”hello”。

```
char str[6] = "hello";
```

上面代码产生和下面代码相同的结果：

```
char str[6]={'h', 'e', 'l', 'l', 'o', '\o'};
```

因为C语言中的字符串都以空（NULL）字符为终结，故要确认定义的数组足够长以存放空字符。这就是为什么hello只有5个字符，而str 要有6个字符长的原因。使用字符串常量时，编译程序自动地在末尾加上空字符。

多维数组初始化的方法与一维数组相同，例如，下式将 sqrs初始化为从1到10及它们各自的平方数。

```
int sqrs[10][2]={
                    1, 1,
                    2, 4,
                    3, 9,
                    4, 16,
                    5, 25,
                    6, 36,
                    7, 49,
                    8, 64,
                    9, 81,
                    10, 100,
                                };
```

5.4.2 变长数组的初始化

设想用数组初始化的方法建立一个如下错误信息表：

```
char e1[12] = "read error\n";
char e2[13] = "write error\n";
char e3[18] = "cannot open file\n";
```

可以想象，如果用手工去计算每一条信息的字符数以确定数组的长度是何等的麻烦。利用变长数组初始化的方法可以使C自动地计算数组的长度。变长数组初始化就是使 C编译程序自动建立一个不指明长度的足够大的数组以存放初始化数据。使用这种方法，以上信息表变为：

```
char e1[] = "read error\n";
char e2[] = "write error\n";
char e3[] = "cannot open file\n";
```

给定上面的初始化，下面的语句

```
printf("%s has length %d\n",e2, sizeof(e2));
```

将打印出：

```
write error
has length 13
```

除了减少麻烦外，应用变长数组初始化使程序员可以修改任何信息，而不必担心随时可能发生的计算错误。

变长数组初始化的方法不仅仅限于一维数组。但在对多维数组初始化时，必须指明除了第一维以外其它各维的长度，以使编译程序能够正确地检索数组。其方法与数组形式参数的说明类似。这样就可以建立变长表，而编译程序自动地为它们分配存储空间。例如，下面用变长数组初始化的方法定义数组 sqrs:

```
int sqrs[ ][2]={
                  1, 1,
                  2, 4,
                  3, 9,
                  4, 16,
                  5, 25,
                  6, 36,
                  7, 49,
                  8, 64,
                  9, 81,
                  10, 100
                 };
```

相对定长数组的初始化而言，这种说明的优点在于可以在不改变数组各维长度的情况下，随时增加或缩短表的长度。

5.5 应用程序举例

[例5-6] 为比赛选手评分。

计算方法：从10名评委的评分中扣除一个最高分，扣除一个最低分，然后统计总分，并除以8，最后得到这个选手的最后得分(打分采用百分制)。

```
#include<stdio.h>
main()
{
        int score[10];       /*10 个评委的成绩*/
        float mark;           最后得分*/
        int i;
        int max = -1;      /*最高分*/
        int min = 101;      /最低分*/
        int sum = 0;        /*1个评委的总和*/

        for(i=0;i<10;i++)
        {
              printf("Please Enter the Score of No. %d:",i+1);
              scanf("%d\n"  , &score[i]);
```

```
            sum=sum+score[i];
        }
        for(i=0;i<10;i++)
        {
            if(score[i]>max)
                max=score[i];
        }
        for(i=0;i<10;i++)
        {
            if(score[i]<min)
                min=score[i];
        }
        mark=(sum-min-max)/8.0;
    printf("The mark of the player is %.1f\n",mark);
    }
```

[例5-7] **数列排序，采用选择法实现对有5个数的数列进行排序。**

选择法的算法思想是：(降序)

1. **将待排序的**n**个数放入数组**num**中，即**num[0]、num[1]、...num[n-1]。

2. **让**num[0]**与后续**num[1]...num[n-1]**依次比较，保证大数在前、小数在后。此次比较，**num[0]**是数组中最大。**

3. **余下**n-1**个元素**

4. num[1]**与**num[2]...num[n-1]**依次比较，大数在前、小数在后，此次**num[1]**是全部元素的最大。**

num[n-2]**与**num[n-1]**比较，**num[n-2]**存大数。**

num[n-1]**存小数，比较结束，整理有序。**

例：待排序5**个数为：** 44 76 82 63 71

一趟排序：1**次比较：**76 44 82 63 71

2**次比较：**82 44 76 63 71

3**次比较：**82 44 76 63 71

4**次比较：**82 44 76 63 71

最大

```
#include <stdio.h>
main()
{
    int num[5];
    int i, j;
    int temp;

    num[0]=94; num[1]=76; num[2]=82; num[3]=63;  num[4]=71;
    for(i=0; i<4; i++)
        for(j=i+1; j<5; j++)
           {
```

```
                    if(num[i]>num[j])
                    {
                            temp=num[i];
                            num[i]=num[j];
                            num[j]=temp;
                    }
                }
    for(i=0; i<5; i++)
            printf("%4d"  , num[i]);
    printf("ok\n");
    }
```

这是一个非常简单的排序程序，我们只需稍加扩展就可以编制出很多功能强大的管理程序，如学生统计总分、平均排列年级名次等。

[例5-8] 简易学生成绩查询系统。

图5-3为学生成绩登记表，下例程序完成如下功能：

1) 根据输入的学生学号，给出各次考试成绩及平均成绩；

2) 根据输入考试的次数，打印出该次考试中每个学生的成绩，并给出平均分；

3) 根据学号查出学生某次考试成绩；

4) 录入考试成绩。

考试 / 学号 成绩	1	2	3	4	5	6
1	80	60	70	80	50	90
2	80	70	82	50	90	60
3	75	86	74	81	92	61
4	55	61	70	72	74	81

图5-3 学生成绩表

```
    #include <stdio.h>
    mian()
    {
            int select;
            int i, j;
            int score[5][7];
            int average=0;
            int sum=0;
       do{
            printf(" 本程序有4项功能\n");
            printf("1 、根据学号查询学生成绩\n");
            printf("2 、根据考试号统计成绩\n");
            printf("3 、根据考试号和学号查询成绩\n");
            printf("4 、成绩录入\n");
            printf("0 、退出\n");
            printf(" 请输入选择（0-4）:");
            scanf("%d\n"  , &select);
```

```
switch(select)
{
        case 0:
                printf("OK\n");
                exit(0)
                break;
        case 1:
                printf(" 输入学号：");
                scanf("%d\n"  , &i);
                for(j=1; j<7; j++)
                        {
                                printf(" 第%d科成绩是%d\n" , j, score[i][j]);
                                 sum += score[i][j];
                        }
                average=sum/6;
                printf(" 学生的平均成绩是%d\n" , average);
                break;
        case 2:
                printf(" 输入考试号：");
                scanf("%d\n"  , &j);
                for(i=1; i<5; i++)
                    {
                printf(" 第%d号学生本科成绩是%d\n" , i, score[i][j]);
                                sum += score[i][j];
                    }
                average=sum/4;
                printf(" 本科平均成绩是%d\n" , average);
                break;
        case 3:
                printf(" 输入学号和考试号 ：");
                scanf("%d %d\n", &i, &j);
                printf(" 第%d号学生的第 %d科考试成绩是 %d\n" , i, j,
                score[i][j]);
                break;
        case 4:
                printf(" 请输入成绩\n");
                for(i=1; i<5; i++)
                        for(j=1; j<7; j++)
                                scanf("%d\n"  , &score[i][j]);
                break;
        default:
         break;
         }while(1);
}
```

从本例中可以看出，当涉及到二维数组时，通常用两重 for循环来存取元素。