学生应知信息知识

〔语言简明教程（中）

狄登峰 主编

目　　录

第四章　数组及其应用

迄今为止，我们使用的都是属于基本类型（整形、字符型、实型）的数据，C 语言还提供了构造类型的数据，它们有：数组类型、结构体类型、共用体类型。构造类型数据是由基本类型数据按照一定的规则组成的，因此有的书又称它们为“导出类型”。

本章将只介绍数组。数组是具有相同数据类型且按一定次序排列的一组变量的集合体，构成一个数组的这些变量称为数组元素。数组有一个统一的名字叫数组名，每个数组元素并没有另外的名字，它可通过数组名及其在数组中的位置（叫下标）来确定。即数组元素是用数组名后跟用方括号（[]）括住的下标来表示。例如 name［15]、list［5］［10］等。数组按下标个数分类，有一维、二维和三维数组等，二维以上数组统称为多维数组。

4.1　一维数组

4.1.1　一维数组的定义

一维数组是数组名后只有一对方括号的数组，其定义方式为：

数据类型　　数组名［元素个数］；

例如：

char str［50］

此语句定义了一个由 50 个元素组成的一维数组，数组名为 str，这 50 个元素分别为 str［0］、str［1］、str［2］、str［3］…str［49］，每个元素都是字符型变量。关于数组的定义，应注意如下几点：

（1）数组名后用方括号括住数组元素的个数，不能使用圆括号。

（2）元素个数可以是整型常量，也可以是整型常量表达式，但决不能含有变量，因为此表达式的值是在编译时计算出来的，而编译时系统并不能确定变量的取值。

（3）数组元素个数必须大于或等于 1。

（4）数组元素的下标是从 0 开始编号的，因此，对于定义：float a［8］，其第一个元素是 a［0］而不是 a［1］，最后一个元素是 a［7］，而不是 a［8］。

根据上述要点可以判断，下面四个定义是非法的：

```
int size1，size2；
folat height［size1］；                    （使用变量做下标是错误的）
float width［size1+size2+1］；             （用含变量的表达式做下标是错误的）
int number［-8］                           （使用负数做下标是错误的）
int m（8）                                 （数组名后面用（）是错误的）
```

而下面的数组定义是合法的：

```
# define STRSIZE 50
char string［STRSIZE］；
int m［15* STRSIZE］；
float x［3*32+1］；
```

4.1.2 一维数组的存储形式

一维数组在内存中存储时，按下标递增的次序连接存放。对于 int a［15］数组名 a 或 a［0］是数组存储区域的首地址，即数组第一个元素存放的地址。其中＆是地址运算符，它表示取 a［0］的地址。因此，数组名是一个地址常量，不能对其进行赋值和进行＆运算。

4.1.3 一维数组的引用

与变量类似，任何一个数组都应先定义或说明，然后再引用。在 C 语言中不能对数组整体进行操作，例如不能对整个数组进行赋值或其他各种运算。只能对数组元素进行操作。数组的引用形式为：

数组名［下标］

其中下标既可以是整型常量表达式，也可以是含有变量的整型表达式。例如，在例中的 x［k+2］。

```
# define SIZE 4
# include < stdio.h >
main（）
{
float x[SIZE],sum= =0.0;
int k =0;
scanf（" % d % d % d",x）；                    / * 对数组进行整体输
入是错误的 * /
scanf（" % d",& x[3]）；                       / * 正确的输入方式 *
/
```

```
    while（k <SIZE）{
    sum + = x[k];                                    / * 正确的引用方式
* /
    k + +;
    }
    k=0
    printf（" % d % d % d % d \ n",x）;              / * 对数组进行整体输
出是错误的 * /
    printf（" % d % d \ n",x[k+1],x[k+2]）;          / * 正确的引用方式 *
/
    printf（" % d \ n",x[SIZE]）;                    / * 下标越界 * /
    }
```

在 C 语言中，编译和执行程序时，系统并不自动检查数组下标是否越界，因此可能访问到数组所占的内存空间以外的存储单元，而这种访问往往是十分危险的，因此程序员要特点注意下标越界的问题。

4.1.4 一维数组的初始化

初始化就是给数组元素赋初始值，有如下两种初始化方法：

1．用赋值语句初始化

用赋值语句初始化是在程序执行时实现的。

2．在数组定义时初始化

这种初始化是在编译时进行的。其一般形式如下：

数据类型 数组名［数组元素个数］＝｛值 1，值 2，…值 n｝；

对上述形式说明如下：

- **花括号中的值是初始值，用逗号分开。例如：**

```
int m［3］= {0，1，2} ；
```

那么各数组元素的初始值为：m［0］＝0，m［1］＝1，m［2］＝2。

- **如果花括号中值的个数少于数组元素的个数，则多余的数组元素初始化为** 0。**例如：**

```
int m［3］= {0，1} ；
```

则各数组元素的初始值为：m［0］＝0，m［1］＝1，m［2］＝0。

- **可对数组中的部分元素赋值，这时对不赋值的数组元素可在括号中缺少相应的值，但逗号不能省略，而缺省值视为** 0。**例如：**

```
int m［3］= {0，，1} ；
```

此时各数组元素的初始值为：m［0］＝1，m［1］＝0，m［2］＝1。

- **在数组定义中，可缺省方括号（[]）中的元素个数，而用花括号中初始值的个数来决定数组元素个数。例如：**

```
int m［］= {0，1，2} ；
```

相当于：

```
int m［3］= {0，1，2} ；
```

- **对于静态或全局类型的数组，如果不在定义显示初始化，则多数编译系统都将其初始化为** 0。

4.1.5 一维数组的应用举例

【例 4-1**】有一递推数列，满足：**f（0）＝0，f（1）＝1，f（2）＝2，f（n+1）=f（n）+2f（n-1）f（n-2）（n>=2）。**使用数组编写程序，顺序打印出** f（0）**到** f（10）**的值。**

分析：可以定义一个整型数组，用于存放 f（0）**到** f（10）

这 11 个整数。先将 f（0）、f（1）、f（2）的值直接赋给数组的前三个元素，然后建立一个循环，每次取出数组中的三个元素（最后一个元素必须是上一次进行循环体时刚计算出来的），通过递推公式求出下一项的值并存入数组中，直到 11 项都被计算出来为止。

程序如下：

```
# include < stdio.h >
main（）
{
int f[11],k;
f[0]=0,f[1]=1,f[2]=2;                    / * 前三项赋初值 * /
for[k=3;k <11;k + +）
f[k]=f[k-1]+ 2 * f[k-2] * f[k-3];        / * 递推求解每一项的值 *
/
for（k=0;k<11;k+ +）
printf（" % d,",f[k]）;
}
```

从上例中可以看出，使用数组的方便之处之一是通过下标存取数组元素正好同循环控制变量及其表达式相匹配，因此，数组+循环就可以简单地实现顺序地、逆序地或跳跃地对大量数据进行连续处理，这是仅使用基本数据类型无法办到的。

【例 4-2】请用户输入一个含有 12 个浮点数的一维数组，请分别计算出数组中所有的正数的和以及所有的负数的和。

分析：显然应该建立一个循环，从头到尾地将整个数组搜索一遍，同时将其中所有的正数和负数分别累加起来以求得结果。程序如下：

```
# include < stdio.h >
main（）
{float data[12];                          / * 存放浮点数的一维数组 * /
float result1=0.0,result2=0.0;            / * 将要用于分别存放正数和，
负数和 * /
int i;
printf（"please input 12 float numbers: \ n"）;
for（i=0;i<12;i++）
scanf（" % f",& data[i]）;                / * 逐一输入数据 * /
for（i=0;i<12;i+ +）
{ if（data[i]>0.0）
result 1 + =data[i];                      / * 累加正数 * /
else
result2 + =data[i];                       / * 累加负数 * /
}
printf（" the sum of all the positive numbers is % .3f \ n",result1）;
printf（" the sum of all the positive numbers is % .3f \ n",result2）;
}
```

【例 4-3】使用直接插入法对 12 个整数进行排序（按从小到大的顺序排列）。

直接插入排序的算法描述如下：

比较待排序数组中前两个数的大小，按要求的顺序排列好；将第三个数加入到由前两个数组成的有序子序列中，保证此三个数的排列依然保持所要求的顺序；以此类推，当前 N 个数已按从小到大的顺序排好后，将第 N+1 个数依次同前面的各数相比较，直到找到一个合适的位置将其插入，使前 N+1 个数保持从小到大的顺序排列：重复上述过程，直到整个数组中的所有元素都被处

理过为止。

程序如下：

```
# include < stdio.h >
main（）
{
int array[12],i,j;
printf（"input 12 integers: \ n"）;
for（i =0;i<12;i+ +）
scanf（" % d",& array[i]）;          / * 逐一输入数据 * /
for（i=0;i<12;i+ +）                  / * 从第二个数开始，逐一将当前数据插入到此数之
                                        前的数据序列中，所以，当前数以前的数据序列总是有序 * /
{ int temp =array[i];
for（j =i-1;j > =0;j - -）             / * 逐一搜索当前数以前的有序数据序列 * /
{if（array[j]>temp）                   / * 未达到插入点，则将原序列中的数据依次后移 * /
{if（array[j+1]=array[j];
else
{array[j+1]=temp;
break;
}
}
if（j= = -1）                          / * 前序列中的第一个数都比此数小，则插入到第一的位置 * /
array[0]=temp;
}
```

```
for（i=0;i<12;i+ +）
printf（"% d,",array[i]）;
}
```

【例 4-4**】用数组来处理求** Fibonacci **数列问题。**

程序如下：

```
main（）
{
int i;
static int f[20]={1,1};
for（i=2;i<20;i++）
f[i]=f[i-2]+f[i-1];
for（i=0;i<20;i++）
{
if（i%5= =0）printf（"\n"）;
printf（"%12d",f[i]）;
}
}
```

运行结果如下：

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

if **语句用来控制换行，每行输出** 5 **个数据。**

4.2 多维数组

数组名后有两对方括号的数组叫二维数组，有三对

方括号的数组叫三维数组，方括号对数大于或等于二的数组统称多维数组。

4.2.1 多维数组的定义

多维数组定义的一般形式如下：

数据类型标识符 数组名［常量表达式 e1］［常量表达式 e2］…；

多维数组定义的数组元素个数为：e1*e2…

同一维数组一样，多维数组每一维元素的下标也都从 0 开始。例如：

```
char c［2］［2］；
```

此二维数组共有四个元素，分别为：c［0］［0］，c［0］［1］，c［1］［0］，c［1］［1］。这四个数组元素的类型均为字符型。又如：

```
int n［2］［3］［2］；
```

此三维数组共有 12 个元素，分别为：

n［0］［0］［0］ n［0］［0］［1］ n［0］［1］［0］ n［0］［1］［1］ n［0］［2］［0］ n［0］［2］［1］

n［1］［0］［0］ n［1］［0］［1］ n［1］［1］［0］ n［1］［1］［1］ n［1］［2］［0］ n［1］［2］［1］

这些元素均为整型变量。一维数组定义需注意的问题对多维数组也适用，此处不再一一例举。

4.2.2 多维数组的存储形式

多维数组在内存中按下标顺序依次存储在内存的连续空间中。

对于二维数组，是按先行后列的顺序存放，如：char

c［2］［2］；先存放第一行，且顺序为 c［0］［0］，c［0］［1］，然后再存放第二行，且顺序为 c［1］［0］，c［1］［1］，数组 c［2］［2］在内存中的连续存放顺序如图 4-1a 所示。

C 语言允许使用多维数组，下面介绍多维数组是如何存储的。

以三维数组为例，如 int n［2］［3］［2］；则先存放 n［0］［0］［0］，n［0］［0］［1］，然后再存放 n［0］［1］［0］，n［0］［1］［1］，…，最后存放 n［1］［2］［0］，n［1］［2］［1］，如图 4-1b 所示。

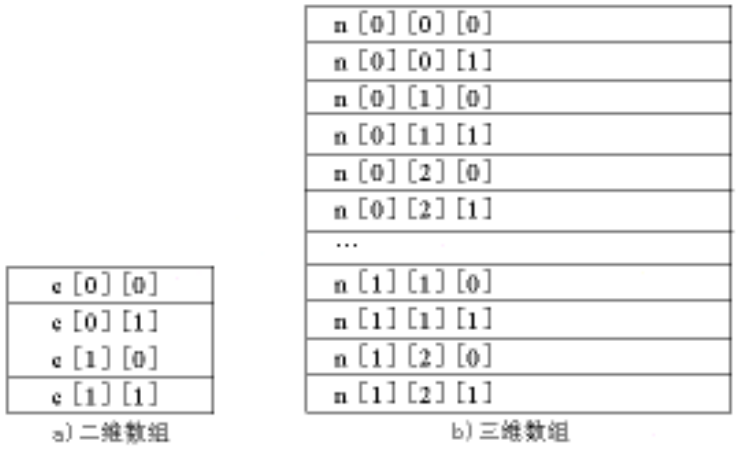

图 4-1 数组元素在内存中的连续存放示意图

4.2.3 多维数组的引用

不论是一维数组还是多维数组，都不能对其进行整体引用，只能对具体元素进行引用。引用格式与一维数组类似。

二维数组的引用形式：

数组名［e1］［e2］；

三维数组的引用形式：

数组名［e1］［e2］［e3］

其中 e1 e2 e3 是值大于或等于 0 的整型表达式，这些表达式可包含变量。例如，对于数组：

int n［10］［15］［12］；

以下的引用方式都是合法的：

n［0］［1］［2］，n［k+1］［0］［4］，n［3*k+2］［j+3］［11］

在数组引用中要特别注意下标越界。因为系统不检查下标越界问题，所以程序设计者就要特别注意。例如，对于数组：

int b［4］［5］；

则引用 b［4］［5］是错误的，因为该引用下标越界。

4.2.4 多维数组的初始化

同一维数组一样，对于全局类型和静态类型的数组可在定义的同时进行初始化。初始化的方式有如下两种：

1．把初始值括在一个花括号内

例如，对二维数组 c［2］［2］，可用如下方法初始化：

int c［2］［2］＝｛'a'，'b'，'c'，'d'｝

于是有：

c［0］［0］＝'a'，c［0］［1］＝'b',c［1］［1］＝'d'。

可以看出，实际上系统是按数组元素在内存中的存放顺序将初始化列表中的值依次赋给各数组元素的。从下面的例子中也可以得到同样的结论：

int n［2］［3］［2］＝｛0，1，2，3，4，5，6，7，8，9，10，11｝；

于是有：

n［0］［0］［0］=0 n［0］［0］［1］=1

n［0］［1］［0］=2 n［0］［1］［1］=3

n［0］［2］［0］=4 n［0］［2］［1］=5

n［1］［0］［0］=6 n［1］［0］［1］=7

n［1］［1］［0］=8 n［1］［1］［1］=9

n［1］［2］［0］=10 n［1］［2］［1］=11

2．把多维数组分解成多个一维数组

二维数组又可看作一维数组，该一维数组的每一个元素又是一个一维数组。例如：

int a［2］［3］；

可把它看成是具有两个元素的一维数组 a［0]、a［1]。而 a［0]、a［1］又都具有三个元素的一维数组，即：

a［0］：a［0］［0］，a［0］［1］，a［0］［2］

a［1］：a［1］［0］，a［1］［1］，a［1］［2］

因此，上例对二维数组的初始化又可分解成对多个一维数组的初始化：

char c［2］［2］={｛'a'，'b'｝，｛'c'，'d'｝}

其效果与不分解完全一样。又如对三维数组：

int n［2］［3］［2］；

也可按上述方法逐步分解,可分解成 2 个二维数组：

n［0］

n［1］

它们各有 3×2＝6 个元素。每个二维数组又可分解为 3 个一维数组，所以共分界出 6 个一维数组：

n［0］［0］

n［0］［1］

n［0］［2］