第5章

数　　组

在很多应用问题中，需要对大批相同类型的数据进行同样功能的编程。例如，从高到低输出全班学生程序课程的考试成绩。显然，我们首先需要将全班学生的成绩存储起来。那么，如何定义变量来存储一批类型相同的数据呢？

到目前为止，我们所学的变量都只能保存单一数据的标量，即基本数据类型的变量。实际上，在C语言中，除了提供整型、实型、字符型等基本数据类型外，还提供了组合数据类型，它们有数组、结构体、联合体等，这些数据类型是由基本类型组合而成的，其对应的变量可以用来存储数值的集合。

本章介绍数组的定义和使用，重点讨论一维数组、字符数组、数组作函数参数、数组指针等内容。

5.1　数组的基本语法

5.1 数组的基本语法

数组是由相同类型的一组数据按照一定的顺序组合而成的，其中每个数据都是该数组中的一个数组元素。数组元素被一个共同的名字（即数组名）和各自的顺序号（即下标）来唯一地标识。用数组来标识大量同类型的数据比用基本类型的变量标识要简单，并且处理效率高，程序的可读性好。

数组可以是一维的，也可以是多维的。若每个数据在数组中的顺序只须用一个下标表示，则该数组是一维数组，例如数学中的向量、数据的有限序列等一维线性结构的数据就可用一维数组来描述；若每个数据在数组中的顺序必须用两个或多个下标才能表示，则该数组是多维数组。例如，数值计算中经常用到的矩阵，它有两个维，不是线性结构，即矩阵中的元素必须通过两个下标指定，可以用二维数组来描述。二维数组是最简单的多维数组。C语言支持定义二维及更高维的数组，并且把二维数组看作数组类型为一维数组的数组，即数组中的所有数组元素又都是同类型同长度的一维数组。把三维数组看作是数组类型为二维数组的数组。以此类推，n 维数组即为数组类型为 $n-1$ 维数组的数组。本节只介绍一维数组和二维数组，多维数组可由二维数组类推得到。

5.1.1　数组的定义

同其他基本数据类型的变量一样，在C语言中使用数组变量也遵循“先定

义、后使用”的原则。

n 维数组的定义方式为

```
类型说明符 数组名[常量表达式 1][常量表达式 2]…[常量表达式 n]
```

其中：

(1) 类型说明符是任一种基本数据类型或构造数据类型的说明符。数组的类型实际上是指各个数组元素的数据类型。对于同一个数组，其所有元素的数据类型都是相同的。

(2) 数组名是用户定义的数组标识符，其书写规则应符合标识符的书写规定，且不能与其他变量同名。

(3) 方括号中的常量表达式 i 表示第 i 维数组元素的个数，也称为数组第 i 维的长度。它同时规定了数组中数组元素第 i 维下标的取值范围。

例如，int a[10]表示数组 a 是一维数组，且包含 10 个数组元素，其下标从 0 开始，因此 10 个数组元素分别为 a[0]、a[1]、a[2]、a[3]、a[4]、a[5]、a[6]、a[7]、a[8]、a[9]。又如，float b[3][4]说明了一个三行四列的二维数组，数组名为 b，其数组元素的类型为单精度浮点型，该数组的数组元素共有 3×4 个，即：

b[0][0],b[0][1],b[0][2],b[0][3]
b[1][0],b[1][1],b[1][2],b[1][3]
b[2][0],b[2][1],b[2][2],b[2][3]

众所周知，执行到变量定义语句时系统会在内存为变量分配相应的存储空间。例如，为整型变量分配 2 字节，为字符型变量分配 1 字节。那么，系统是怎样为数组变量分配存储空间的呢？实际上，系统会分配一段连续的存储空间依次存储数组中的各个数组元素，如图 5-1 所示。为了能够访问数组中的各个元素，系统将这段空间的第一个内存单元地址记入数组名，即数组名中存放的是数组的首地址。假设首地址为 10000H，则数组名中存放的地址值即为 10000H。因此，数组名与变量名有着本质的区别：程序中直接使用变量名代表的是该变量中存放的数据值，而直接使用数组名则代表的是数组的首地址，它是一个地址值，不是数据值。变量可以被赋值，而数组名是常量，不能被赋值。

由于各数组元素是同种类型，且是顺序存放的，所以可以很容易通过数组的首地址计算出数组中第 i 个元素的存放地址。因此，数组是一种可直接存取的线性结构。

系统为数组分配的存储空间大小由数组类型和数组长度共同决定，即数组所占用的存储空间等于所有数组元素占用空间之和。例如，对于数组定义语句“int a[10];”由于每个数组元素都是 int 类型，均需要占用 4 字节，所以系统会分配 40 字节用来存储该数组。

二维数组在逻辑上是二维的，即其下标在两个方向上变化，数组元素在数组中的位置也处于一个平面之中，而不是像一维数组只是一个向量。但是，实际的硬件存储器却是连续编址的，是一个线性结构。在 C 语言中，二维数组是按“行优先”的原则进行存储的。例如，对于数组定义语句 float b[3][4]，先存放 b[0]行，再存放 b[1]行，最后存放 b[2]行。每行中的四个元素也是依次存放。由于数组 b 说明为 float 类型，该类型占 4 字节的内存空间，所以每个元素均占 4 字节，总的存储空间为 3×4×4 字节，如图 5-2 所示。

思考：如何根据数组名、数组类型、各下标值计算多维数组元素的存放地址？

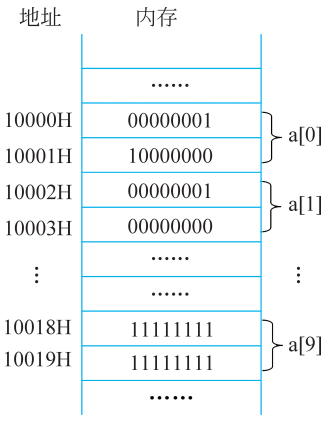

图 5-1　一维数组的存储

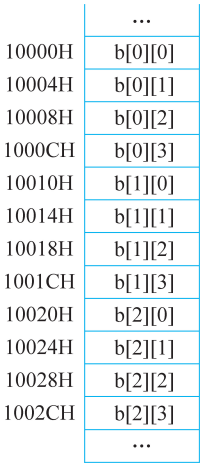

图 5-2　二维数组 b 的存放形式

定义数组还应特别注意：

（1）不能在方括号中用变量来表示元素的个数，但是可以是符号常量或常量表达式。考虑到在程序的后续维护过程中可能需要修改数组的长度，因此较好的做法是用宏来定义数组的长度。

例如：

```
#define FD 5
    int main()
    {
      int a[3+2],b[7+FD];
      ⋮
      return 0;
}
```

是合法的。

但是下述说明方式是错误的。

```
int main()
{
    int n=5;
    int a[n];
    ⋮
    return 0;
}
```

（2）允许在同一个类型说明中，说明多个数组和多个变量。例如：

```
int a,b,c,d,k1[10],k2[20];
```

(3) 在数组定义前加 const 可将数组变为“常量”。例如:

```
const  int  month[ ]={31,28,31,30,31,30,31,31,30,31,30,31};
```

C 语言规定,不允许修改常量数组中各元素的值。将程序执行过程中不希望改变其值的数组声明为常量数组,有利于编译器发现错误,避免不必要的错误发生。

5.1.2 数组的引用

虽然所有的数组元素是用一条语句同时定义的,但人们可能并不期望同时操作所有的数组元素,而只对其中的某个或某几个数组元素进行操作。因此,需要能够对数组元素进行单独操作,也就是单独引用数组元素。

数组元素是组成数组的基本单元,其标识方法为数组名后跟若干个下标,下标表示该数组元素在数组中的顺序号,是几维数组就跟几个下标。这样系统就可以根据数组名、数组类型及数组元素的下标值计算出该数组元素的存放地址,进而对该数组元素进行读写操作。

数组元素也称为单下标变量,其表示形式为

数组名[下标表达式 1][下标表达式 2]…[下标表达式 n]

其中,下标表达式 i 只能为整型常量、整型变量或整型表达式。若为小数,在编译时将自动取整。

这里的下标表达式 i 和数组定义中的常量表达式 i 在形式上有些相似,但这两者具有完全不同的含义。数组定义时方括号中给出的是数组第 i 维的长度,即第 i 维下标的有效范围值;而数组元素中的下标是该元素在数组第 i 维中的位置标识。前者只能是常量,后者可以是常量、变量或表达式。

例如,a[5]、a[i]、a[i+j]都是合法的数组元素。数组元素通常也称为下标变量。

注意:对于“int a[10];”,数组元素 a[10]是不合法的引用,因为这里的下标表达式 10 超出了该数组的有效下标范围。

必须先定义数组,才能使用下标变量。一个下标变量(即数组元素)在本质上相当于一个同类型(数组类型)的普通变量。例如,若有定义 int a[5],则 a[0]、a[1]、a[2]、a[3]、a[4]这 5 个下标变量在程序中的作用相当于 5 个普通的整型变量。在 C 语言中只能逐个地使用下标变量,而不能一次引用整个数组。

例如,输出有 10 个元素的数组必须使用循环语句逐个输出各下标变量:

```
for(i=0; i<10; i++)
    printf("%d",a[i]);
```

而不能用一个语句输出整个数组。

对数组 a,下面的写法是错误的:

```
printf("%d",a);
```

数组下标可以是任何整型表达式，如下面的例 5.1 中的两个程序是完全等价的。但若将程序 2 中的语句“a[i++]=i;”修改为“a[++i]=i;”，两个程序就不再等价了。因此，当下标中包含自增操作时一定要特别注意。

例 5.1　以下两个程序等价。

程　序　1	程　序　2

main()
{
  int i,a[10];
  for(i=0;i<=9;i++)
  a[i]=i;
  for(i=9;i>=0;i--)
  printf("%d ",a[i]);
}
	main()
	{
	  int i,a[10];
	  for(i=0;i<10;)
	  a[i++]=i;
	  for(i=9;i>=0;i--)
	  printf("%d",a[i]);
	}

因为必须对数组中的各个元素进行分别处理，所以对数组的编程往往都离不开循环。通常是对数组元素的一趟或几趟遍历。

例 5.2　录入一串数，然后反向输出这串数。

程序如下：

```
#include<stdio.h>
#define N 10                          /*定义常量 N*/
int main()
{
  int a[N],i;                         /*定义长度为 N 的数组 a*/
  printf("Enter %d numbers:",N);
  for(i=0;i<N;i++)
    scanf("%d",&a[i]);                /*从键盘输入 10 个数,初始化数组 a*/
  printf("In reverse order:");
  for(i=N-1;i>=0;i--)
    printf("%d ",a[i]);               /*逆序输出数组 a 中的各个元素*/
  printf("\n");
  return 0;
}
```

从例 5.2 中不难看到，仅仅通过变换数组下标就可以方便地访问数组中的各个元素。下标的变换要通过循环变量的变化来控制，所以找到它们之间的关系就成为程序设计中运用数组时的关键。另外，这个程序很好地体现了用宏定义数组长度的好处：不但提高了程序的可读性，而且日后想要修改数组的长度时，只需要改一个地方就能做到“一改全改”，方便又快捷。

数组元素的下标总是从 0 开始，所以长度为 n 的数组，其有效的数组下标范围为 0～$n-1$。因此，只有当下标表达式的值在该范围内时数组引用才是有效的。但实际上 C 语

言并不要求对下标范围进行检查,即编译器不认为数组下标越界是语法错误。这就造成存在数组下标越界问题的程序在编译时能通过,执行时却得不到预期的正确结果,甚至运行时会产生错误。

对数组的越界访问会造成不可预计的后果。越界取得的数据显然没有意义,使用这种数据可能导致程序给出莫名其妙的结果。越界赋值则更危险,这种操作的后果是非常可怕的。越界赋值会破坏被赋值位置的原有数据,其后果难以预料,因为根本无法知道被这个操作破坏的到底是什么,可能是其他程序的变量值,也可能是重要的内部控制信息。实际上恶意攻击者或恶意程序最常用的一种技术就是设法造成程序执行中出现数组越界访问,并借机取得被攻击计算机系统的控制权。总之,保证对数组元素的访问不超出合法范围是非常重要的。

同一维数组一样,二维数组元素每一维的下标都必须小于该维的长度规定,否则也会出现数组越界,造成不可预知的后果。

就像前面用单重循环处理一维数组一样,双重循环是处理二维数组的理想选择。

例 5.3 一个学习小组有 5 个人,每个人有 3 门课程的考试成绩,如表 5-1 所示。求各门课程的平均成绩和所有课程的总平均成绩。

表 5-1 某学习小组的考试成绩

课程	张	王	李	赵	周
Math	80	61	59	85	76
C language	75	65	63	87	77
FoxPro	92	71	70	90	85

问题分析:可设一个二维数组 a[5][3]存放 5 个人 3 门课程的成绩,再设一个一维数组 v[3]存放所求得的各门课程的平均成绩,设变量 average 为所有课程的总平均成绩。

程序如下:

```
#include<stdio.h>
int main()
{
  int i,j,s=0,average,v[3],a[5][3];
  printf("input score\n");
  for(i=0;i<3;i++){
      for(j=0;j<5;j++){
        scanf("%d",&a[j][i]);
        s=s+a[j][i];
      }
      v[i]=s/5;
      s=0;
  }
```

```
    average=(v[0]+v[1]+v[2])/3;
    printf("Math:%d\nC language:%d\nFoxPro:%d\n",v[0],v[1],v[2]);
    printf("total:%d\n", average);
    return 0;
}
```

程序中首先用了一个双重循环。在内循环中依次读入某一门课程的各个学生的成绩，并把这些成绩累加起来。退出内循环后再把该累加成绩除以 5 送入 v[i]之中，得到该门课程的平均成绩。外循环共循环 3 次，分别求出 3 门课各自的平均成绩并存放在 v 数组之中。退出外循环之后，把 v[0]、v[1]、v[2]相加除以 3 即得到总平均成绩。最后按题意输出成绩。

温馨提示：注意根据题意合理确定内外重循环的次序。例如在本例中，要求得到每门课的平均成绩，因此，外重循环表示课程、内重循环表示学生，这样处理逻辑清晰、可读性好。请读者将本例中的内外重循环次序颠倒，体会程序设计过程中数据处理上的异同。思考如果要求的是每名同学的平均成绩，又该如何设定内外重次序？

5.1.3　数组的初始化

给数组赋值的方法除了用赋值语句对数组元素逐个赋值外，还可以采用初始化赋值的方法。数组初始化赋值是指在数组定义时给数组元素赋予初值。数组初始化是在编译阶段进行的，这样将减少运行时间，提高效率。

初始化赋值的一般形式为

```
类型说明符 数组名[常量表达式]={值 1,值 2,…,值 n}
```

其中，在花括号中的各数据值即为各元素的初值，各值之间用逗号间隔。

例如："int a[10]={ 0,1,2,3,4,5,6,7,8,9 };"相当于"a[0]=0;a[1]=1,…,a[9]=9;"。

C 语言对数组的初始化赋值还有以下几点规定：

(1) 可以只给部分元素赋初值。当{ }中值的个数少于元素个数时，只给前面部分元素赋值，其余元素自动赋 0 值。

例如：

```
int a[10]={0,1,2,3,4};
```

表示只给 a[0]～a[4]这 5 个元素赋值，而后 5 个元素自动赋 0 值。因此，若想将数组 a 中的各元素均初始化为 0，可简写为："int a[10]={0};"。

(2) 只能给元素逐个赋值，不能给数组整体赋值。

例如，给 10 个元素全部赋 1 值，只能写为

```
int a[10]={1,1,1,1,1,1,1,1,1,1};
```

而不能写为

```
int a[10]=1;
```

(3) 如果给全部元素赋值,则在数组说明中可以不给出数组元素的个数,数组长度由花括号中值的个数决定。

例如,"int a[10]={1,2,3,4,5};"与"int a[]={1,2,3,4,5};"是不等价的。前一个数组长度为10,而后一个数组长度为5。

二维数组的初始化赋值可按行分段赋值,也可按行连续赋值。

例如,对数组a[5][3]按行分段赋值可写为

```
int a[5][3]={ {80,75,92},{61,65,71},{59,63,70},{85,87,90},{76,77,85} };
```

按行连续赋值可写为

```
int a[5][3]={ 80,75,92,61,65,71,59,63,70,85,87,90,76,77,85};
```

这两种赋初值的结果是完全相同的。

对于二维数组初始化赋值还有以下说明:

(1) 可以只对部分元素赋初值,未赋初值的元素自动取0值。

例如:

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

是对每一行的第一列元素赋值,未赋值的元素取0值。赋值后各元素的值为

```
1 0 0
2 0 0
3 0 0
```

又如:

```
int a [3][3]={{0,1},{0,0,2},{3}};
```

赋值后的元素值为

```
0 1 0
0 0 2
3 0 0
```

(2) 如对全部元素赋初值,则第一维的长度可以不给出。

例如:

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

可以写为

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

例 5.4　请以矩阵的形式输出矩阵 ***A*** 与矩阵 ***B*** 相加之和。

程序如下：

```
#include<stdio.h>
int main()
{
  int i,j;
  int a[5][3]={{80,75,92},{61,65,71},{59,63,70},{85,87,90},{76,77,85}};
  int b[5][3]={{1},{0,1},{0,0,1}};
  for(i=0;i<5;i++)
  {
    for(j=0;j<3;j++)
      printf("%4d",a[i][j]+b[i][j]);          /*对应位置的元素相加*/
    printf("\n");                             /*保证输出为矩阵形式*/
  }
  return 0;
}
```

5.1.4　多维数组的分解

数组是一种构造类型的数据结构。二维数组可以看作是由一维数组嵌套构成的。假设一维数组的每个元素又是一个数组，就组成了二维数组。当然，前提是各元素类型必须相同。根据这样的分析，一个二维数组也可以分解为多个一维数组。C 语言允许这种分解。

如二维数组 a[3][4]，可分解为三个长度为 4 的一维数组，其数组名分别为 a[0]、a[1]、a[2]。对这三个一维数组不需另外说明即可使用。这三个一维数组都有 4 个元素，例如，一维数组 a[0]的元素为 a[0][0]、a[0][1]、a[0][2]、a[0][3]，如图 5-3 所示。必须强调的是，a[0]、a[1]、a[2]不能当作数组元素使用，它们是数组名，不是一个单纯的数组元素。

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

图 5-3　a[3][4]分解示意图

多维数组的分解类似二维数组的分解。例如，四维数组可看作数组元素为三维数组的一维数组，而其中的每个数组元素（三维数组）又可看作数组元素为二维数组的一维数组，……以此类推。可见，数组定义本身就是一种递归定义的形式。

5.2 字符数组与字符串

5.2 字符数组与字符串

字符数组就是数组类型为字符类型的数组,用于保存一串字符。由于人们经常用 C 语言编写处理字符序列或各种文本的程序,因此 C 语言为处理字符数组提供了专门的支持。

字符串是典型的非数值对象,其存储模式是以字符数组作为存储空间,结尾加结束标志符"\0"。对字符串的基本操作主要通过标准库提供的函数来实现。

5.2.1 字符数组的基本语法

字符数组的定义形式与前面介绍的数值数组相同。例如,"char c[10];",由于字符型和整型通用,也可以定义为 int c[10],但这时每个数组元素占 4 字节的内存单元。

c数组	
C	c[0]
	c[1]
p	c[2]
r	c[3]
o	c[4]
g	c[5]
r	c[6]
a	c[7]
m	c[8]
\0	c[9]
…	

图 5-4 字符数组 c 的初始化结果

字符数组也可以是二维或多维数组。例如,"char c[5][10];"即为二维字符数组。

字符数组也允许在定义时作初始化赋值。例如,"char c[10]={'C',' ','p','r','o','g','r','a','m'};"赋值后各元素的值如图 5-4 所示。

其中,c[9]未赋值,系统自动赋予 0 值。

当对全体元素赋初值时也可以省去长度说明。例如,"char c[]={'C',' ','p','r','o','g','r','a','m'};"这时 c 数组的长度自动定为 9,即图 5-4 中最后一个 c[9]不存在。

在 C 语言中没有专门的字符串变量,通常用一个字符数组来存放一个字符串。前面介绍字符串常量时,已说明字符串总是以"\0"作为串的结束符。因此,当把一个字符串存入一个数组时,也把结束符"\0"存入数组,并以此作为该字符串是否结束的标志。字符串的长度计数不包括"\0"。

C 语言允许用字符串的方式对字符数组作初始化赋值。

例如,"char c[]={'C',' ','p','r','o','g','r','a','m'};"可写为"char c[]={"C program"};"或去掉花括号写为"char c[]="C program";"。

用字符串方式赋值比用字符逐个赋值要多占 1 字节,用于存放字符串结束标志"\0"。上面的数组 c 在内存中的实际存放情况为

C		p	r	o	g	r	a	m	\0

"\0"是由 C 编译系统自动加上的。由于采用了"\0"标志,所以在用字符串赋初值时一般无须指定数组的长度,而由系统自行处理。

思考:能否直接将一个字符串赋值给一个字符数组?为什么?