第5章 数 组

在解决实际问题的过程中，常会遇到对大量类型相同的数据进行统一处理的情况。例如：对班级全体学生的考试成绩进行排序处理；对若干个统计数据做均值、方差计算等。在解决这类问题的时候，如果每个学生的成绩都用简单变量来存储，就需要很多变量，程序中引用这些变量比较烦琐，给程序设计带来极大的不便。

本章将介绍一种重要的数据类型——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，这些变量在内存中的存储位置是连续的，称为数组元素。数组有统一的名字叫作数组名，用数组名和下标来唯一确定数组中的元素。按照下标数量，数组分为一维数组、二维数组和多维数组，本章只介绍一维数组和二维数组。

本章要点

- 一维数组、二维数组及字符数组的定义、初始化及元素引用方法。
- 求最大值、最小值及排序、查找等常用算法。
- 字符数组的输入与输出、字符串处理函数及对字符串处理的算法。

5.1 一维数组的定义和引用

数组是数据类型相同的一组数，和变量一样，数组也必须先定义后使用。本节主要介绍数组的定义、初始化及引用方法。

5.1.1 一维数组的定义

一维数组的定义格式为：

```
数据类型 数组名[常量表达式];
```

例如：

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

它表示定义了一个名称为 a 的数组，该数组有 10 个元素，即数组的长度为 10。这 10 个元素分别为 a[0]，a[1]，a[2]，a[3]，…，a[9]，其存放的数据类型为整型。

例如：

```
float b[15];
```

所定义的数组 b 有 15 个元素，元素分别为 b[0]，b[1]，b[2]，…，b[14]，相当于同时定

义了 15 个单精度浮点型变量。

【说明】

(1) 数据类型用来说明数组元素的类型：int、float、double 或 char。

(2) 数组的命名应遵守标识符的命名规则。

(3) 数组名后用方括号括起常量表达式。常量表达式表示的是数组元素的个数，即数组的长度。常量表达式中可以包括常量和符号常量，但是不能包含变量，因为 C 语言规定数组不能动态定义。例如，下面都是正确的数组定义：

```
#define N 10
int a[4+11],b[3*10],c[N],d[N*2];
```

而下列数组定义是非法的：

```
int n=5;
int a[n],b[n+2];
```

(4) 数组元素的下标从 0 开始。例如，在前面定义的数组 a，该数组有 10 个元素，第一个元素是 a[0]，而不是 a[1]，最后一个元素是 a[9]，而不是 a[10]。

(5) 数组元素在内存中是连续存储的。例如，在前面定义的数组 a，其在内存中的存储形式如下：

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

数组所占用的内存空间的字节数为：

$$\text{数组长度} \times \text{sizeof(数据类型)}$$

例如，在前面定义的数组 a 在内存中占的空间为：

$$10 \times \text{sizeof(int)} = 10 \times 4\text{B} = 40\text{B}$$

(6) 数组名是地址常量，代表着整个数组中的所有元素在内存中存储的起始地址，称为数组的首地址。

5.1.2 一维数组的初始化

数组的初始化是指在定义数组的同时给数组元素赋初值。当系统为所定义的数组在内存中开辟连续存储单元时，这些存储单元中并没有确定的数值，欲使数组元素有一个确定的值，可以在程序中用赋值语句或输入函数来实现，也可以在定义数组时给数组元素指定初始值。

一维数组初始化的格式为：

```
数据类型  数组名[常量表达式]={初值列表};
```

例如：

```
int a[5]={11,12,13,14,15};
```

其作用是在定义数组的同时将常量 11、12、13、14、15 分别存入数组元素 a[0]、a[1]、a[2]、a[3]、a[4]中。

【说明】

(1) 初值必须依次放在花括号{}内，各值之间用半角逗号隔开。

（2）可以只给部分数组元素赋初值。例如：

```
int a[10] = {1,3,5,7,9};
```

数组 a 有 10 个元素，但只提供 5 个初值分别赋给前 5 个数组元素，后 5 个数组元素由系统自动赋值 0。

（3）在进行数组初始化时，{}中值的个数不能超过数组长度。例如，下面是一种错误的数组初始化方式：

```
int a[5] = {10,20,30,40,50,60,70};
```

（4）在给所有数组元素赋初值时，可以不指定数组长度，由系统根据初值的个数来确定数组的长度。例如：

```
int a[ ] = {2,4,6,8,10};
```

系统会自动定义数组 a 的长度为 5。

（5）若定义数组时不进行初始化，则该数组元素的值是不确定的。如果想将所有数组元素的初值置为 0，可以采用如下方式：

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

5.1.3 一维数组元素的引用

C 语言规定数组不能以整体形式参与数据处理，只能逐个引用数组元素。一维数组元素的引用方式为：

```
数组名[下标表达式]
```

其中，“下标表达式”可以是整型常量、整型变量或整型表达式，其值表示数组中某一元素的顺序号。假如有定义：

```
int a[10] = {11,13,15,17,19},i = 2;
a[0] = 12;
a[2 * 4] = 8;
a[i + 5] = 20;
a[9] = a[1] + a[i + 1] + a[i * 3];
```

以上都是正确的表达式。

如果想将 1,2,3,...,10 依次存入数组 a 中，可通过下面的循环来实现：

```
for(i = 0;i < 10;i++)
    a[i] = i + 1;
```

也可以通过下面的循环来实现对数组 a 中元素的输入：

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

通过上述关于数组元素的引用，可以发现对多个数据类型相同的数据进行操作时，使用数组是比较方便的。

【说明】

（1）只能逐个引用数组元素，而不能一次引用整个数组。

（2）可以像使用普通变量一样使用数组元素，数组元素可以出现在表达式的任何位置。

在使用各数组元素时，可以利用循环控制下标表达式的值来操作数组。例如：

```
int a[5] = {10,20,30,40,50},i,s = 0;
for(i = 0;i < 5;i++)
    s = s + a[i];
```

以此实现对数组 a 中所有数组元素求和。

(3) 数组的长度和下标是两个完全不同的概念。如，int a[10]；表示数组 a 有 10 个数组元素，只能有效引用 a[0]，a[1]，a[2]，…，a[9]这 10 个数组元素。如果程序中出现对 a[10]的引用，这时 C 语言的编译系统不会出错，但 a[10]的值是不确定的，也不是用户所需要的。这一点对初学者来说在应用中经常出错，要格外注意。

(4) 数组中的数组元素相当于一个变量，同变量一样可以作为函数的实参。用数组元素做实参，是将数组元素的值传递给对应的形参。

(5) 数组名不同于数组元素。数组名代表整个数组的所有元素在内存中存储的起始地址，称为数组的首地址，是地址常量。数组名可以作为函数的实参，函数调用时是把数组的首地址传给被调函数，此时要求实参数组与对应的形参数组数据类型必须相同，而形参数组可以不指定大小。

5.1.4 一维数组应用举例

【例 5-1】 编写用户自定义函数实现素数判断，求已知数组 a[8]={2,4,8,9,11,17,19,35}中的所有素数之和。

算法设计如下。

定义一个函数实现素数的判定，如果是素数，返回值为 1；如果不是素数，返回值为 0。在主函数 main()中依次使用数组 a 的元素作为实参进行函数调用，即可实现对数组 a 中的素数求和。

程序代码：

```
#include <stdio.h>
int fun(int x)                          /* 判定 x 是否为素数,如是返回 1,否则返回 0 */
{
    int k;
    for(k = 2;k <= x/2;k++)
        if(x % k == 0) return 0;
    return 1;
}
void main()
{
    int a[8] = { 2,4,8,9,11,17,19,35},i,s = 0;
    for(i = 0;i < 8;i++)
        if(fun(a[i]) == 1) s += a[i];    /* 数组元素做函数实参 */
    printf("s = %d\n",s);
}
```

运行结果：

```
s = 49
```

如同本例，用数组元素做函数实参，一般是通过多次调用函数将数组元素分别传递到被

调函数，进行某种分析处理，并返回所要的结果。

【例 5-2】 计算 n 个数 $x_1, x_2 \cdots x_n$ 的方差 S^2。方差公式如下：

$$S^2 = ((x_1 - x_0)^2 + (x_2 - x_0)^2 + \ldots + (x_n - x_0)^2)/n$$

其中 $x_0 = (x_1 + x_2 + \ldots x_n)/n$

方差是各个数据分别与其平均数之差的平方和的平均数，是一组数据离散程度的度量。在许多实际问题中，研究方差即偏离程度有着重要意义，方差越小，代表这组数据越稳定，方差越大，代表这组数据越不稳定。

算法设计如下。

(1) 在主函数 main()中定义包含 n 个数组元素的一维数组 a，输入 n 个数存入其中。

(2) 在函数 fun()中先利用公式求 $x_0 = (x_1 + x_2 + \cdots x_n)/n$，再利用公式求 $S = ((x_1 - x_0)^2 + (x_2 - x_0)^2 + \cdots + (x_n - x_0)^2)/n$，最后将 S 返回主函数 main()。

(3) 在主函数 main()中输出 S 的值。

这里假设求 10(n=10)个实数的方差。

程序代码：

```
#include <stdio.h>
float fun(float x[],int n)
{
    float x0 = 0,S = 0;                    /* 用变量 S 代表方差 S2 */
    int k;
    for(k = 0;k < 10;k++)
        x0 = x0 + x[k];
    x0 = x0/n;
    for(k = 0;k < 10;k++)
        S = S + (x[k] - x0) * (x[k] - x0);
    S = S/n;
    return(S);
}
void main()
{
    float a[10],FC = 0;                    /* 用变量 FC 代表方差 S2 */
    int i;
    printf("input 10 numbers:\n");
    for(i = 0;i < 10;i++)
        scanf("%f",&a[i]);
    FC = fun(a,10);
    printf("FC = %f\n",FC);
}
```

运行结果：

```
input 10 numbers:
76.0 87.5 90.0 77.5 92.5 91.0 81.0 80.5 88.0 91.0 ↙
FC = 33.950000
```

【例 5-3】 通过键盘输入 10 个整数存入数组中，找出其中的最大值。

算法设计如下。

(1) 在主函数 main()中定义含 10 个数组元素的一维数组 a，输入 10 个整数存入其中。

(2) 在函数 fun()中实现求最大值：先假设第一个数组元素 x[0]为最大值元素存入

max 中，然后将数组余下元素 x[k](k=1,2,3,...,9)依次与 max 比较，如 x[k]的值比 max 的值大就进行赋值 max=x[k]，最终 max 的值即为最大值，并返回主函数 main()。

(3) 在主函数 main()中输出 max 的值。

程序代码：

```
#include <stdio.h>
int fun(int x[],int n)                    /* 形参数组 x 不需指定大小 */
{
    int k,max;
    max = x[0];
    for(k = 1;k < n;k++)
        if(x[k]> max) max = x[k];
    return(max);
}
void main()
{
    int a[10],max,i;
    printf("input 10 numbers:\n");
    for(i = 0;i < 10;i++)
        scanf("%d",&a[i]);
    max = fun(a,10);                      /* 数组名 a 作为函数实参 */
    printf("the max number:\nmax = %d\n",max);
}
```

运行结果：

```
input 10 numbers:
23 11 67 9 8 14 65 9 21 17 ↙
the max number:
max = 67
```

求若干个数的最大值问题，好比是若干个人打擂台，先一人上台(暂时为擂主，max=x[0])，余下的人依次攻擂，每次的胜者暂时为擂主(if(x[k]>max) max=x[k]；)，最后留在台上的人为真正的擂主(最终保留在 max 中的值为这些数的最大值)。

求若干个数的最小值的算法类似，请读者自己完成。

【例 5-4】 数组 a 中保存由大到小排好顺序的 10 个整数。输入一个整数 x，用二分法查找 x 是数组 a 中第几个元素的值。如果 x 不在数组 a 中，则输出“无此数”。

算法设计如下。

(1) 在主函数 main()中定义含 10 个数组元素的一维数组 a，输入 10 个整数存入其中。

(2) 在函数 fun()中实现求最大值：先假设第一个数组元素 x[0]为最大值元素存入 max 中，然后将数组余下元素 x[k](k=1,2,3,...,9)依次与 max 比较，如 x[k]的值比 max 的值大就进行赋值 max=x[k]，最终 max 的值即为最大值，并返回主函数 main()。

(3) 在主函数 main()中输出 max 的值。

程序代码：

```
#include <stdio.h>
int bsearch(int a[],int n,int x)
{
    int low = 0,high = n - 1,mid = 0,count = 0;
    while(low <= high)
```

```
    {
        count++;
        mid = (low + high)/2;
        if(x < a[mid])
            low = mid + 1;
        else if(x > a[mid])
            high = mid - 1;
        else if(x == a[mid])
            return mid;
    }
    return -1;
}
void main()
{
    int x,a[10] = {10,9,8,7,6,5,4,3,2,1},t;
    printf("Input x:");
    scanf("%d",&x);
    t = bsearch(a,10,x);
    if(t == -1) printf("无此数!\n");
    else printf("%d是数组中的第%d个数\n",x,t + 1);
}
```

【例 5-5】 输出 Fibonacci 数列：1,1,2,3,5,8,13…前 30 项。

Fibonacci 数列的规律是：前两个数据项都是 1，从第 3 项开始，当前项是它前面相邻两个数据项之和。解决该问题的思想是：根据 Fibonacci 数列的规律，依次生成各数据项存入数组中，然后输出数组中的元素。

算法设计如下。

(1) 定义包含 30 个数组元素的一维数组 a，并对 a[0]、a[1]赋初值 1。

(2) 利用递推关系 a[k]=a[k−1]+a[k−2] (k>=2)生成数组的其他元素。

(3) 按每行 5 个数输出数组中的元素。

程序代码：

```
#include <stdio.h>
void Fib(long x[],int n)
{
    int k;
    x[0] = x[1] = 1;                    /*前两项都是1*/
    for(k = 2;k < n;k++)
        x[k] = x[k - 2] + x[k - 1];
}
void main()
{
    int i;
    long f[30];
    Fib(f,30);
    for(i = 0;i < 30;i++)
    {
        printf("%10ld", f[i]);
        if(i % 5 == 4) printf("\n");    /*控制每行输出5个数*/
    }
}
```

运行结果：

```
     1       1       2       3       5
     8      13      21      34      55
    89     144     233     377     610
   987    1597    2584    4181    6765
 10946   17711   28657   46368   75025
121393  196418  317811  514229  832040
```

循环输出30个数组元素值的过程中，用语句if(i%5==4) printf ("\n")；控制每行输出5个数，这是常用的做法。如想改变每行输出数的个数，只做相应的变动即可。如用语句：

```
if(i%8==7) printf("\n");
控制每行输出数8个数.
```

【例 5-6】 通过键盘输入n(0<n≤10)个整数存入数组，编写程序使前面的数顺序向后移m(0<m<10)个位置，最后m个数变成最前面m个数，如图5-1所示。

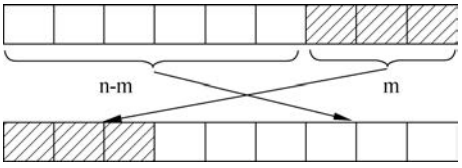


图5-1 移位

算法设计如下。

(1) 在主函数main()中输入n值确定数组实际长度，输入m值确定移动个数，输入n个整数保存在数组中。

(2) 在自定义函数fun()中，将a[1]至a[n－1]共n－1个数组元素依次前移，然后将a[0]存入a[n－1]中；将上述方法重复m次，实现题目要求。

(3) 在主函数main()中输出移动后的数组。

程序代码：

```
#include<stdio.h>
void fun(int a[],int n,int m)
{
    int i,j,t;
    for(i=1;i<=m;i++)
    {
        t=a[0];
        for(j=1;j<n;j++)
            a[j-1]=a[j];
        a[n-1]=t;
    }
}
void main()
{
    int i,n,m,a[10];
    printf("Input n(0<n<=10):");
    scanf("%d",&n);
```

```
    for(i = 0;i < n;i++)
        scanf(" % d",&a[i]);
    printf("Input m(0 < m < 10):");
    scanf(" % d",&m);
    fun(a,n,m);
    printf("The result number:");
    for(i = 0;i < n;i++)
        printf(" % d\t",a[i]);
    printf("\n");
}
```

【例 5-7】 用交换法对 10 个整数由小到大进行排序。

交换法排序的基本思想是：将当前数依次和后面位置的数进行比较，如果后面的数比当前第 1 个位置上的数小就交换，否则不交换。

用交换法对 n 个整数由小到大进行排序的过程如下。

(1) 将第 1 个位置上的数依次和后面位置上的数进行比较，如果后面的数比第 1 个位置上的数小就交换，否则不交换——这样 1 轮的操作就会把 n 个数中的最小者交换到最前面(第 1 个位置)。

(2) 按照(1)的做法，对后 n－1 个位置上的数进行第 2 轮操作，就会把后 n－1 个位置上的数中的最小者交换到第 2 个位置。

(3) 重复上述操作，共经过 n－1 轮的操作结束排序。

例如，对 5 个整数 4、3、2、5、1 用交换法由小到大进行排序，排序过程如下。

原顺序：4　3　2　5　1

第 1 轮：1　4　3　5　2

第 2 轮：1　2　4　5　3

第 3 轮：1　2　3　5　4

第 4 轮：1　2　3　4　5

对 10 个整数用交换法由小到大进行排序的程序代码：

```
#include < stdio.h >
void fun(int x[],int n)
{
    int i,j,t;
    for(i = 0;i < n - 1;i++)
        for(j = i + 1;j < n;j++)
            if(x[j]< x[i]) {t = x[i];x[i] = x[j];x[j] = t;}
}
void main()
{
    int i,a[10];
    printf("Input 10 numbers:\n");
    for(i = 0;i < 10;i++)
        scanf (" % d", &a[i]);
    fun(a,10);
    printf("The sorted numbers;\n");
    for(i = 0;i < 10;i++)
        printf(" % 4d",a[i]);
    printf("\n");
}
```

运行结果：

```
Input 10 numbers:
3 6 2 12 34 11 17 25 9 33↙
the sorted numbers:
    2 3 6 9 11 12 17 25 33 34
```

对若干个数进行排序是非常典型、重要的问题，人们为此进行了多种算法设计。除了本例介绍的交换法外，还有选择法、冒泡法等。下面将介绍冒泡法、选择法的排序算法及程序设计。

【例 5-8】 用冒泡法对 10 个整数由小到大进行排序。

冒泡法排序的基本思想是：对每相邻的两个数进行比较，如果前面的数比后面的数大就交换两个数的位置，否则不交换。

用冒泡法对 n 个整数由小到大进行排序的过程如下。

(1) 对 n 个数从头到尾进行相邻两个数的比较，如果前面的数比后面的数大就交换两个数的位置——第 1 轮的操作就会把 n 个数中的最大者移到最后(第 n 个位置)。

(2) 对前 n－1 个数从头到尾进行第 2 轮操作，就会把前 n－1 个数中的最大者移到第 n－1 个位置上。

(3) 重复上述操作，共经过 n－1 轮次的操作结束排序。

例如，对 5 个整数 6、3、8、5、1 用冒泡法由小到大进行排序，排序过程如下。

原顺序：6　3　8　5　1

第 1 轮：3　6　5　1　8

第 2 轮：3　5　1　6　8

第 3 轮：3　1　5　6　8

第 4 轮：1　3　5　6　8

对 10 个整数用冒泡法由小到大进行排序的程序代码：

```
#include <stdio.h>
void fun(int x[],int n)
{
    int i,j,t;
    for(i=0;i<n-1;i++)
        for(j=0;j<n-1-i;j++)
            if(x[j]>x[j+1]) {t=x[j]; x[j]=x[j+1]; x[j+1]=t;}
}
void main()
{
    int k,a[10];
    printf("Input 10 numbers:\n");
    for(k=0;k<10; k++)
        scanf("%d", &a[k]);
    fun(a,10);
    printf("the sorted numbers:\n");
    for(k=0;k<10;k++)
        printf("%4d", a[k]);
    printf("\n");
}
```

运行结果：

```
Input 10 numbers:
11 56 3 78 24 32 75 81 9 18↙
the sorted numbers:
    3 9 11 18 24 32 56 75 78 81
```

【例 5-9】 用选择法对 10 个整数由小到大进行排序。

选择法排序的基本思想是：先找到数组中元素的最小值，将其与第一个元素互换，以此类推。

用选择法对 n 个整数由小到大进行排序的过程如下。

(1) 首先从 n 个数中找到最小的数，与第 1 个位置上的数交换——这样 1 轮的操作就会把 n 个数中的最小者交换到最前面(第 1 个位置)。

(2) 按照(1)的做法，对后 n－1 个位置上的数进行第 2 轮的操作，就会把后 n－1 个位置上的数中的最小者交换到第 2 个位置。

(3) 重复上述操作，共经过 n－1 轮次的操作结束排序。

例如，对 5 个整数 2、6、7、4、1 用选择法由小到大进行排序，排序过程如下。

原顺序：2　6　7　4　1

第 1 轮：1　6　7　4　2

第 2 轮：1　2　7　4　6

第 3 轮：1　2　4　7　6

第 4 轮：1　2　4　6　7

对 10 个整数用选择法由小到大进行排序的程序代码：

```
#include <stdio.h>
void fun(int a[],int n)
{
    int i,j,temp,max;
    for(i=0;i<n-1;i++)
        {
            max=i;
            for(j=i+1;j<n;j++)
                if(a[max]>a[j]) max=j;
            if(max!=i){temp=a[i];a[i]=a[max];a[max]=temp;}
        }
}
void main()
{
    int i,a[10];
    printf("Input 10 numbers:\n");
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    fun(a,10);
    printf("the sorted numbers:\n");
    for(i=0;i<10;i++)
        printf("%4d",a[i]);
    printf("\n");
}
```

运行结果：

```
Input 10 numbers:
65 34 21 16 87 43 22 59 71 15 ↙
the sorted numbers:
    15 16 21 22 34 43 59 65 71 87
```

在以上三个例子的程序代码中，要弄清实现排序的循环嵌套以及外循环控制排序的轮次，如果 n 个数参加排序，则外循环要被执行 n－1 次。内循环实现数的交换，而且随着外循环的进程，内循环被执行的次数越来越少。

在以上一维数组的应用举例中，涉及了数组名做函数参数的问题，这里进行以下几点说明。

(1) 用数组名做函数参数，应该在主调函数和被调用函数中分别定义数组，不能只在一方定义。

(2) 实参数组与形参数组类型应该一致，如不一致，结果将出错。

(3) 实参数组和形参数组大小可以一致也可以不一致，C 编译对形参数组大小不做检查，只是将实参数组的首地址传给形参数组。

(4) 形参数组可以不指定大小。定义形参数组时，在数组名后面跟一个空的方括号。为了在被调用函数中处理数组元素，可以另设一个参数，传递数组元素的个数。

(5) 数组名做函数参数时，进行的不是“值传送”，而是把实参数组的首地址传递给形参数组，即“地址传递”。两个数组共占同一段内存单元，实际上形参数组就是实参数组，因此在被调函数中表面上看是形参数组元素的值发生改变，实际上改变的是对应实参数组元素的值，具体见例 5-2 至例 5-9。

5.2 二维数组的定义与引用

在实际应用中，经常处理由多行、多列数据组成的矩形数据表(矩阵)。在 C 语言中，通常用二维数组来存储这种形式的数据。与一维数组相同，二维数组也必须先定义，后使用。

5.2.1 二维数组的定义

二维数组的定义格式为：

```
数据类型 数组名[常量表达式 1][常量表达式 2];
```

例如：

```
int x[3][4];
```

定义 x 为 3 行 4 列的整型二维数组，该数组包含 12 个数组元素，相当于定义了 12 个整型变量，它们分别为：x[0][0]、x[0][1]、x[0][2]、x[0][3]、x[1][0]、x[1][1]、x[1][2]、x[1][3]、x[2][0]、x[2][1]、x[2][2]、x[2][3]。这 12 个数组元素在逻辑上是按 3 行 4 列的位置排列的，即：

```
x[0][0]  x[0][1]  x[0][2]  x[0][3]
x[1][0]  x[1][1]  x[1][2]  x[1][3]
x[2][0]  x[2][1]  x[2][2]  x[2][3]
```

但实际在内存中，12 个数组元素是按行的顺序逐行连续存储的。

【说明】

（1）数据类型、数组名、常量表达式的意义与一维数组相同。

（2）注意定义二维数组时的格式，必须分别用方括号把表示行、列的常量表达式括起来。而不能写成：

```
int x[3,4];
```

或

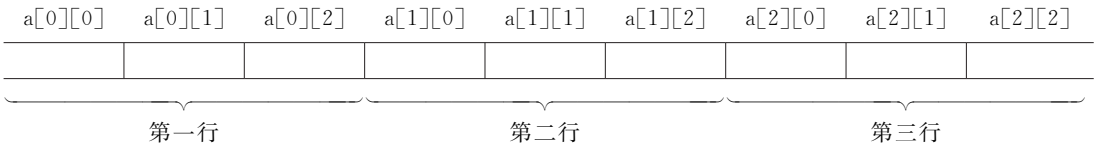
```
int x(3,4);
```

（3）二维数组的数组元素在内存中按行的顺序存放，即在内存中先顺序存放第一行数组元素，再存放第二行数组元素，以此类推。

例如：

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

数组 a 的元素在内存中的存储形式为：



（4）二维数组可以看成是特殊的一维数组，例如：

```
float t[4][5];
```

可以把数组 t 看成是包含 4 个元素的一维数组，即由 t[0]、t[1]、t[2]、t[3]组成的一维数组。但这里的 4 个元素 t[0]、t[1]、t[2]、t[3]又都是包含 5 个元素的一维数组的名字。

t[0] → t[0][0] t[0][1] t[0][2] t[0][3] t[0][4]
t[1] → t[1][0] t[1][1] t[1][2] t[1][3] t[1][4]
t[2] → t[2][0] t[2][1] t[2][2] t[2][3] t[2][4]
t[3] → t[3][0] t[3][1] t[3][2] t[3][3] t[3][4]

了解了一维数组、二维数组的定义，就可以类推出多维数组的定义。例如：

```
int a[4][3][2];                /* 定义了三维数组 a */
float s[3][3][2][3];           /* 定义了四维数组 s */
```

在进行多维数组的定义时，第一个[]称为第一维，第二个[]称为第二维，数组的维从左到右以此类推。

5.2.2 二维数组的初始化

与一维数组相同，二维数组也可以在定义时对数组元素赋初值。可以用以下方法对二维数组进行初始化。

（1）按行赋初值。每一行的初始化值用“{}”括起来，用“,”将不同行的初始化值分开，总体再加一对“{}”括起来。例如：

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

(2) 二维数组是按行连续存储的,因此可以用一维数组的初始化方式来初始化二维数组。例如:

```
int b[2][3] = {11,12,13,14,15,16};
```

通过定义初始化,11、12、13 分别赋给第 1 行的数组元素,14、15、16 分别赋给第 2 行的数组元素。

(3) 可以只对一部分数组元素赋初值。例如:

```
int c[3][4] = {{1,2,3},{0},{8,9}};
```

通过定义使 c[0][0]=1、c[0][1]=2、c[0][2]=3、c[2][0]=8、c[2][1]=9,数组的其余元素值都为 0。例如:

```
int d[2][3] = {9,8,7,6};
```

通过定义使 d[0][0]=9、d[0][1]=8、d[0][2]=7、d[1][0]=6,数组的其余元素值都为 0。

(4) 如果对数组的全部元素都赋初值,则定义数组时可以不指定数组的第一维长度,但第二维长度不可以省略。例如:

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

可以写成:

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

int x[][3]={1,2,3,4,5,6,7,8}; 也是正确的二维数组定义。二维数组的存储是按行自左至右的顺序存储的。由于每行有 3 个数组元素,要 3 行才能存储下 8 个整数,系统定义数组 x 为 3 行。但是,int x[3][]={1,2,3,4,5,6,7,8}; 是错误的,因为系统无法确定二维数组 x 每行有多少个数组元素。

5.2.3 二维数组元素的引用

无论是一维数组还是二维数组,都不能对其进行整体引用,只能对具体数组元素进行引用。与一维数组元素的引用类似,二维数组元素的引用方式为:

```
数组名[下标 1][下标 2]
```

其中下标可以是整型常量、整型变量或整型表达式。习惯上,下标 1 称为行标,下标 2 称为列标。无论行标还是列标,下标均从 0 开始变化,其值分别小于数组定义中的[常量表达式 1]与[常量表达式 2]。

在对二维数组进行引用时,一般通过双重循环来实现。例如:

```
int a[4][4],i,j,s = 0;
for(i = 0;i < 4;i++)                    /* 双循环实现对二维数组 a 的各元素进行赋值 */
    for(j = 0;j < 4;j++)
        a[i][j] = i + j + 1;
```

赋值之后二维数组 a 中各元素的值为:

```
1 2 3 4
2 3 4 5
```

```
3 4 5 6
4 5 6 7
```

外循环 for(i=0；i<4；i++)中的控制变量 i 的取值决定对哪一行元素进行操作，内循环 for(j=0；j<4；j++)中的控制变量 j 的取值决定对某行中的各列元素进行操作。通过双循环实现了对二维数组 a 的各行各列所有元素进行赋值。

当只对二维数组中的一部分元素进行引用时，应注意对所引用元素的下标的正确表示。例如，对主对角线上的数组元素进行求和：

```
for(s = 0,j = 0;j < 4;j++)
    s = s + a[j][j];                    /* 主对角线上数组元素下标的特点是行标与列标相等 */
```

对主对角线及下方数组元素进行求和：

```
for(s = 0,i = 0;i < 4;i++)
    for(j = 0;j <= i;j++)           /* 主对角线及下方数组元素下标的特点是列标<= 行标 */
        s = s + a[i][j];
```

还有对周边元素进行求和等。

5.2.4 二维数组应用举例

【例 5-10】 求一个 3 行 3 列的整数数组中的最大值。

求二维数组中的最大值与求一维数组中的最大值的方法相同，只是要逐行逐列进行比较。

程序代码：

```
#include <stdio.h>
int fun(int x[][3])                     /* 形参为二维数组,可以省略第一维下标 */
{
    int i,j,t;
    t = x[0][0];
    for(i = 0;i < 3;i++)
        for(j = 0;j < 3;j++)
            if(x[i][j]> t) t = x[i][j];
    return(t);
}
void main()
{
    int a[3][3] = {{5,7,8},{19,9,2},{7,37,11}},max;
    max = fun(a);                       /* 二维数组名做参数 */
    printf("max value is %d\n", max);
}
```

运行结果：

```
max value is 37
```

【例 5-11】 某班级有 10 名学生，本学期每名学生均有 5 门课的考试成绩，分别求出每门课的平均分数。

算法设计如下。

(1) 定义数组 score[10][5]及数组 ave[5]，并对 ave[5]赋初值。

(2) 输入 10 名学生的 5 门课成绩存入数组 score 中。

(3) 按列对数组 score 求平均值存入数组 ave 中。

(4) 输出数组 ave 中的元素值。

程序代码：

```
#include <stdio.h>
void fun(float x[][5],float y[],int n)
{
    int i,j;
    for(i=0;i<5;i++)
    {
        for(j=0;j<n;j++)
            y[i]+=x[j][i];
        y[i]/=n;
    }
}
void main()
{
    float score[10][5],ave[5]={0};
    int m,n;
    for(m=0;m<10;m++)
        for(n=0;n<5;n++)
            scanf("%f",&score[m][n]);
    fun(score,ave,10);
    printf("average of scores:\n");
    for(m=0;m<5;m++)
        printf("%d:%6.2f\n",m+1,ave[m]);
}
```

执行程序时，依次输入每名学生的 5 门成绩，各科成绩数据间用空格分开。请读者自拟成绩数据运行该程序。

以上两个例子中涉及二维数组作为函数参数的问题。实参数组与形参数组的数据类型应该相同，形参数组的第一维下标可以省略。实际上，形参数组的结构与大小都可以与实参数组的结构不同，只要形参数组的元素个数不超过实参数组的元素个数即可。下面将例 5-10 改成如下代码：

```
#include <stdio.h>
int fun(int x[2][2]) /* 注意:形参数组 x 与实参数组 a 的行、列都不同 */
{
    int i,j,t;
    t=x[0][0];
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            if(x[i][j]>t) t=x[i][j];
    return(t);
}
void main()
{
    int a[3][3]={{5,7,8},{19,9,2},{7,37,11}},max;
    max=fun(a);                    /* 二维数组名做参数 */
    printf("max value is %d\n", max);
}
```

运行结果：

```
max value is 37
```

在这段代码中，形参数组 x 与实参数组 a 的结构与大小不同，实参数组 a 含有 9 个数组元素，而形参数组 x 只含有 4 个数组元素。由于二维数组的元素在内存中是按行连续存储的，函数调用时，实参数组 a 的首地址传递给形参数组 x，形参数组 x 只取实参数组 a 的前 4 个数组元素：a[0][0]、a[0][1]、a[0][2]、a[1][0]，其余部分不起作用。

5.3 字符数组与字符串

字符数组是数据类型为字符型的数组，是用于存储多个字符数据或字符串的。字符数组的每一个数组元素用于存放一个字符型数据，在内存中占一个字节。一般情况下，一维字符数组用于存储一个字符串，而二维字符数组用于存储多个字符串。

5.3.1 字符数组的定义与引用

1. 字符数组的定义

字符数组的定义格式：

```
char 数组名[常量表达式];
char 数组名[常量表达式 1][常量表达式 2];
```

例如：

```
char s[5];
char t[4][5];
```

s 为一维字符数组，该数组包含 5 个数组元素，最多可以存放 5 个字符型数据。t 为二维字符数组，该数组有 4 行，每行 5 列，最多可以存放 20 个字符型数据。

2. 字符数组的初始化

字符数组的初始化方式与其他类型数组的初始化方式类似。

（1）逐个数组元素赋初值。例如：

```
char a[5] = {'C','h','i','n','a'};
```

（2）如果初值的个数大于数组长度，则按语法错误处理。例如：

```
char b[4] = {'C','h','i','n','a'};
```

C 编译系统将给出错误提示。

（3）如果初值的个数小于数组长度，则 C 编译系统自动将未赋初值的元素定为空字符（即 ASCII 码值为 0 的字符——'\0'）。例如：

```
char c[10] = {'C','h','i','n','a'};
```

则数组 c 中元素的初值为：

C	h	i	n	a	\0	\0	\0	\0	\0

（4）如果省略数组的长度，C 编译系统会自动根据初值个数来确定数组长度。例如：

```
char d[] = {'H','o','w',' ','a','r','e',' ','y','o','u','?'};
```

通过字符数组定义初始化，数组 d 的长度自动设定为 12。

(5) 二维字符数组也可以进行初始化。见例 5-13，定义并赋初值，二维字符数组 c[5][5]中元素的初值为：

□	□	*	\0	\0
□	*	*	*	\0
*	*	*	*	*
□	*	*	*	\0
□	□	*	\0	\0

其中“□”表示空格。

3. 字符数组的引用

字符数组元素中存储的是字符型数据，引用一个数组元素将得到一个字符。其引用方法与前面介绍的一维、二维数组元素引用方法相同。具体引用情况看下面的例子。

【例 5-12】 输出一个字符串。

算法设计如下。

(1) 定义一个字符数组并把字符串中的每个字符通过赋初值存入字符数组中。

(2) 逐个输出数组中的元素。

程序代码：

```
#include <stdio.h>
void main()
{
    char c[10] = {'I',' ','a','m',' ','h','a','p','p','y'};
    int i;
    for(i = 0;i < 10;i++)
        printf("%c",c[i]);                    /* 逐个输出数组中的元素 */
    printf("\n");
}
```

运行结果：

```
I am happy
```

【例 5-13】 输出菱形图形。

算法设计如下。

该问题的处理方法同例 5-12，首先定义二维字符数组并赋初值，然后逐个输出数组中的元素。

程序代码：

```
#include <stdio.h>
void main()
{
    char c[5][5] = {{' ',' ','*'},{' ','*',' ','*'},{'*',' ','*',' ','*'},{' ','*',' ','*'},{' ',' ','*'}};
    int i,j;
    for(i = 0;i < 5;i++)
```

```
    {
        for(j = 0;j < 5;j++)
            printf(" % c",c[i][j]);
        printf ("\n");
    }
}
```

运行结果：

```
  *
 ***
*****
 ***
  *
```

上面两个例子介绍了用字符数组处理字符串的简单应用，通过对字符数组元素的逐个引用，每引用一个元素得到一个字符数据。通过上述例子发现，对二维字符数组元素进行引用时比较烦琐。C 语言系统提供了专门用于字符串处理的函数，用于解决字符串的输入输出及运算问题。

5.3.2 字符串与字符数组

1. 字符串的定义

字符串是由若干个有效字符组成且以字符'\0'为结束标志的字符序列。字符串常量是用双引号括起来的若干个字符。例如："a ** b8d"、"a"、"123"、"8"都是字符串常量。

C 编译系统在处理字符串时，在其末尾自动添加一个'\0'作为结束符，'\0'称为字符串结束标志。'\0'代表 ASCII 码为 0 的字符，是不可以显示的字符，是一个“空操作字符”，即它不引起任何控制动作。

在 C 语言中，对字符串进行处理时遇到'\0'，则表示字符串结束，即'\0'前面的字符组成的字符序列才是有效的，是系统处理的对象。

2. 字符串与字符数组

C 语言中没有字符串数据类型，所以不能使用字符串变量。字符串是通过字符数组来存储的。一维字符数组可存放一个字符串，二维字符数组可存放多个字符串。

在存储字符串时，字符串结束标志'\0'占用一个字节的存储单元，但是它不计入字符串的实际长度。

前面曾用下面的方法对字符数组赋值：

```
char s[15] = {'I',' ','a','m',' ','h','a','p','p','y'};
```

这样处理虽然也实现了将字符串保存到数组 s 中，但比较麻烦。在实际应用中，通常将字符串看成一个整体来处理，即用字符串常量来直接初始化字符数组，例如：

```
char s[15] = {"I am happy"};
```

系统将字符串中的字符依次赋给字符数组中的各个元素，并自动在末尾补上字符串结束标志'\0'，一起存到字符数组中。数组 s 中有 15 个元素，余下的数组元素系统自动补上字符串结束标志'\0'。其存储形式为：

I		a	m		h	a	p	p	y	\0	\0	\0	\0	\0

对字符数组初始化可以省略大括号而直接用字符串常量来进行。例如：

```
char c[] = "I am happy";
```

这与上面的定义完全等价。

也可以将多个字符串存入二维字符数组中，例如：

```
char t[3][15] = {"ABCDEF","123456789","xyz789"}
```

其存储形式为：

A	B	C	D	E	F	\0	\0	\0	\0	\0	\0	\0	\0	\0
1	2	3	4	5	6	7	8	9	\0	\0	\0	\0	\0	\0
x	y	z	7	8	9	\0	\0	\0	\0	\0	\0	\0	\0	\0

每个字符串在内存中都占用连续的存储空间，而且这段连续的存储空间有唯一确定的首地址。如果它只是一个字符串常量，那么这个字符串常量本身代表的就是该字符串在内存中所占连续存储空间的首地址，是一个地址常量；如果将字符串赋值给一个一维字符数组，那么这个一维字符数组的名字就代表这个首地址。

关于字符串有以下几点说明。

(1) 字符串结束标志'\0'用于判断字符串是否结束，输出字符串时不输出'\0'。

(2) 字符串结束标志'\0'占用一个字节内存空间，但不计入字符串的有效字符。例如：

```
char a[] = "china";
```

数组 a 的大小为 6，而不是 5。

(3) 用字符串来初始化字符数组时，有效的字符个数至少比数组长度少 1，否则系统提示错误。例如：

```
char a[5] = "china";
```

是错误的。而如下定义：

```
char b[5] = {'c','h','i','n','a'};
```

是正确的，请注意前后的不同。但不能将 b 当作字符串使用，原因是数组 b 中的字符序列尾部没有'\0'。

5.3.3 字符数组的输入与输出

1. 用格式符“%c”逐个输入与输出字符数组元素

用格式符“%c”逐个输入、输出字符数组元素，例如：

```
char s[10];
int i;
for(i = 0;i < 10;i++)
{
    scanf("%c",&s[i]);                    /* 输入一个字符存入数组元素 s[i] */
    printf("%c",s[i]);                    /* 输出数组元素 s[i]中的字符 */
}
```

2. 用格式符“%s”对字符数组进行输入或输出

用格式符“%s”，可将输入的字符串存入字符数组中，也可对存储在字符数组中的字符串进行整体输出。例如：

```
char str[100];
scanf("%s",str);                    /* 输入一个字符串存入数组 str */
printf("%s",str);                   /* 输出存储在数组 str 中的字符串 */
```

【例 5-14】 从键盘读入一串字符，将其中的大写字母转换成小写字母后输出该字符串。

算法设计：大写字母比其对应的小写字母的 ASCII 码值小 32，在 C 语言中允许字符数据与整数直接进行算术运算，即'A'+32 就可得到小写字母数据'a'。

程序代码：

```
#include <stdio.h>
void fun(char x[])
{
    int i;
    for(i=0;x[i]!='\0';i++)
        if(x[i]>='A'&&x[i]<='Z') x[i]+=32;
}
void main()
{
    char str[100];
    scanf("%s",str);                /* 输入一个字符串存入数组 str */
    fun(str);
    printf("%s\n",str);             /* 输出存储在数组 str 中的字符串 */
}
```

运行该程序两次。

第一次输入：

```
ProGram↙
```

运行结果：

```
program
```

第二次输入：

```
HOW DO YOU DO?↙
```

运行结果：

```
how do you do?
```

在实际应用中，经常会遇到对字符数组中的字符串进行逐个字符处理的问题。应理解并掌握相应的循环控制，即循环 for(i=0; x[i]!='\0'; i++)中的判定表达式 x[i]!='\0'，其功能是从字符数组中的第一个字符开始，逐个验证其是否为字符结束标志'\0'，是就结束循环。实际上，循环 for(i=0; x[i]!='\0'; i++)可以写成 for(i=0; x[i]; i++)，原因是当前字符为字符结束标志'\0'时，x[i]的值为数 0，循环结束。

【注意】

(1) 用“%s”格式符输入字符串时，scanf 函数中的地址项是数组名，不要在数组名前加

地址运算符“&”，因为数组名本身就是地址。

(2) 用“%s”格式符输出字符串时，printf 函数中的输出项是字符数组名，而不是数组元素。如果写成下面的格式是错误的：

```
printf ("%s", str[0]);
```

(3) 以 scanf ("%s",数组名)；形式读入字符串时，遇空格或回车都表示字符串结束，系统只是将第一个空格或回车前的字符置于数组中。例如：

```
char s[20];
scanf ("%s",s);
```

若输入为：

```
How are you?↙
```

则系统只是将字符串 How 存入数组 s 中。因此，若想将含有空格的字符串输入一维字符数组中，用“%s”格式符是无法实现的。实际应用时，常用下面介绍的字符串输入函数 gets()来完成。

3. 用库函数 gets()与 puts()实现字符数组的输入和输出

(1) 字符串输入函数：gets()。

一般格式：

```
gets(字符数组名);
```

功能：从终端输入一个字符串到字符数组。使用该函数可以输入空格，遇回车结束输入。

例如：

```
char str[20];
gets(str);
puts(str);
```

输入：

```
I am happy↙
```

运行结果：

```
I am happy
```

(2) 字符串输出函数：puts()。

一般格式：

```
puts(字符数组名);
```

功能：将一个字符串输出到终端，字符串中可以包含转义字符。

例如：

```
char str[] = "China\nBeijing";
puts(str);
```

运行结果：

```
China
Beijing
```

【注意】

puts 函数输出字符串后，自动产生换行。

5.3.4 字符串处理函数

C 系统的库函数中提供了一些字符串函数，使用它们可以方便地实现对字符串的处理，如字符串的连接、拷贝、比较、测试长度等。使用这些函数时要包含头文件“string.h”，即应在程序的开始做如下包含处理：

```
#include <string.h>
```

1. 字符串连接函数：strcat()

格式：

```
strcat (字符数组 1,字符数组 2);
```

功能：将字符数组 2 中的字符串连接到字符数组 1 中的字符串的后面，结果放在字符数组 1 中。

例如：

```
char str1[14] = "China",str2[ ] = "Beijing";
strcat(str1, str2);
printf("%s", str1);
```

输出结果为：

```
ChinaBeijing
```

说明：使用 strcat 函数时，字符数组 1 应足够大，以便能容纳连接后的新字符串。

2. 字符串拷贝（复制）函数：strcpy()和 strncpy()

格式：

```
strcpy(字符数组 1,字符数组 2);
```

功能：将字符数组 2 中的字符串拷贝到字符数组 1 中。

例如：

```
char str1[10],str2[ ] = "China";
strcpy(str1,str2);
puts(str1);
```

运行结果：

```
China
```

【说明】

（1）字符数组 1 的长度应大于或等于字符数组 2 的长度，以便容纳被复制的字符串。

（2）字符数组 1 必须写成数组名的形式（如上例中的 str1），字符数组 2 可以是一个字符串常量。例如：

```
char str1[10];
strcpy(str1,"China");
```

其结果与 strcpy(s1,s2)相同。

(3) 执行 strcpy 函数后,字符数组 1 中原来的内容将被字符数组 2 的内容(或字符串)所代替。

(4) 不能用赋值语句将一个字符串常量或字符数组直接赋给另一个字符数组。下面的用法是错误的:

```
str1 = str2;
```

在进行字符串的整体赋值时,必须使用 strcpy 函数。

(5) 函数 strncpy 的功能是将字符串 2 中的前 n 个字符复制到字符数组 1 中去。例如:

```
char str1[20] = "ABCDEFGH",str2[10] = "12345";
strncpy(str1,str2,3);
```

执行代码将 str2 最前面的 3 个字符复制到字符数组 str1 中,取代 str1 最前面的 3 个字符。

【注意】

复制的字符个数 n 应该小于 str1 的数组长度。

执行后输出字符数组 str1:

```
puts(str1);
```

运行结果:

```
123DEFGH
```

3. 字符串比较函数:strcmp()

格式:

```
strcmp(字符串 1,字符串 2)
```

功能:比较两个字符串的大小,比较的结果由函数值带回。其规则如下。

(1) 如果字符串 1 等于字符串 2,函数值为 0。

(2) 如果字符串 1 大于字符串 2,函数值为一个正整数。

(3) 如果字符串 1 小于字符串 2,函数值为一个负整数。

例如:

```
char str1[10] = "Beijing",str2[10] = "Shanghai";
int t;
t = strcmp(str1,str2);
printf("%d\n",t);
```

运行结果:

```
-1                    /* 函数值为一个负整数, 说明字符串 s1 小于字符串 s2 */
```

例如:

```
t = strcmp("China","Beijing");
printf("%d\n",t);
```

函数值为一个正整数,说明字符串"China"大于字符串"Beijing"。

两个字符串比较时,自左至右逐个字符相比较(按 ASCII 码值大小比较),直到出现不同的字符或遇到'\0'为止。如果全部字符相同,则认为两个字符串相等;若出现不相同字

符，则以第一个不相同的字符的比较结果为准。

值得注意的是：比较两个字符串时，不能使用关系运算符（>、>=、<、<=、==、!=）进行比较。例如，判定两个字符串是否相等时，以下格式：

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

是错误的。应使用：

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

4. 字符串长度测试函数：strlen()

格式：

```
strlen(字符数组名)
```

功能：测试字符串的长度。函数返回值为字符数组中第一个'\0'前的字符的个数（不包括'\0'）。

例如：

```
char str[10] = "China";
printf(" % d\n",strlen(str));
```

运行结果：

```
5
```

5. 字符串小写函数：strlwr()

格式：

```
strlwr(字符串);
```

功能：将字符串中的大写字母转换成小写字母。

例如：

```
char str[30] = "Ab2c * 3 ** aBC56";
strlwr(str);
printf(" % s",str);
```

运行结果：

```
ab2c * 3 ** abc56
```

6. 字符串大写函数：strupr()

格式：

```
strupr(字符串);
```

功能：将字符串中的小写字母转换成大写字母。

例如：

```
char str[30] = "Ab2c * 3 ** aBC56";
strupr(str);
puts(str);
```

运行结果：

```
AB2C * 3 ** ABC56
```

5.3.5 字符数组应用举例

【例 5-15】 通过键盘输入两个字符串,编程实现两个字符串的连接(不使用 strcat 函数)。

算法设计如下。

(1) 定义两个字符数组 s1、s2,并输入两个字符串存入其中。

(2) 找到 s1 中字符串的结束位置(结束标志'\0'的下标),其目的是确定 s2 中的字符串接到 s1 中的起始位置。

(3) 将 s2 中的字符依次接到 s1 中的字符后面,并在 s1 中的有效字符后加上结束标志'\0'。

(4) 输出字符数组 s1。

程序代码:

```
#include <stdio.h>
#include <string.h>
void Lj(char str1[],char str2[])   /*形参数组类型须与实参数组类型相同*/
{
    int i,n;
    n = strlen(str1);   /*表达式 strlen(str1)的值是 str1 中字符串结束标志'\0'的下标*/
    for(i = 0;str2[i]!= '\0';i++)
        str1[n++] = str2[i];       /*将 str2 中的字符依次接到 str1 的字符串后面*/
    str1[n] = '\0';                /*在 str1 中的有效字符后加上结束标志'\0'*/
}
void main()
{
    char s1[100],s2[50];
    gets(s1);
    gets(s2);
    Lj(s1,s2);                     /*数组名 s1、s2 做实参,调用函数 Lj()实现字符串连接*/
    puts(s1);
}
```

输入:

```
China↙
2022↙
```

运行结果:

```
China2022
```

【注意】

程序中语句 str1[n]='\0'; 是不可缺少的。因为将 str2 中的字符串连接到 str1 中的字符串后面,必须加上结束标志'\0',str1 中的字符串才是确定的。

【例 5-16】 编程判断输入的字符串是否为“回文”,所谓“回文”是指顺读和倒读都是一样的字符串。例如:level

算法设计如下。

(1) 在主函数 main()中定义字符数组 str,输入字符串存入数组 str,调用函数 fun()实现判断。

(2) 在函数 fun()中对字符数组 s 进行如下操作：i=0 表示字符数组 s 首字符的下标，j=strlen(s)-1 表示字符数组尾字符的下标,若首尾字符相同则继续判断,直至循环条件数组首尾下标 i<j 不成立,循环结束后返回 1；若判断的过程中出现首尾字符不相同的情况，则返回 0,函数判断结束。

(3) 在主函数 main()中输出“回文”结论。

程序代码：

```
#include<stdio.h>
#include<string.h>
int fun(char s[])
{
    int i,j;
    for(i=0,j=strlen(s)-1;i<j;i++,j--)
        if(s[i]!=s[j])
            return 0;
    return 1;
}
void main()
{
    char str[100];
    printf("Input string:");
    gets(str);
    if(fun(str)) printf("%s是回文字符串!\n",str);
    else printf("%s不是回文字符串!\n",str);
}
```

【例 5-17】 通过键盘输入一个字符串存入 str1 数组中,找出字符串中下标为奇数,同时 ASCII 值也为奇数的字符,将其从字符串中删除,剩余的字符形成一个新字符串存入数组 str2 中。

算法设计如下。

(1) 在主函数 main()中定义两个字符数组 str1 和 str2,输入字符串存入数组 str1,调用函数 fun()实现符合条件字符的删除。

(2) 在函数 fun()中对字符数组 s1 中的所有元素做如下处理：下标和 ASCII 值同时为奇数的字符不写入数组 s2 中,其余字符写入数组 s2 中,实现相应字符的删除。

程序代码：

```
#include<stdio.h>
void fun(char s1[],char s2[])
{
    int i,j=0;
    for(i=0;s1[i];i++)
        if(i%2==1&&s1[i]%2==1) continue; /*下标和 ASCII 同为奇数的字符不写入数组 s2
中*/
        else s2[j++]=s1[i];
    s2[j]='\0';
```

```
}
void main()
{
    char str1[100],str2[100];
    scanf("%s",str1);
    fun(str1,str2);
    puts(str2);
}
```

输入：

```
AABBCCDD↙
```

运行结果：

```
ABBCDD
```

该程序是删除字符串中的某些字符，剩余字符存到另一字符数组中，原字符数组中的字符串没有被改变。若例题题干改为：将字符串中下标为奇数，同时 ASCII 值也为奇数的字符删除，则需要改变原字符数组中的字符串。常用的解决方法有以下两种。

方法一：如同例题实现，先将剩余字符存到另一个字符数组中，然后再覆盖原字符数组中的字符串。代码如下：

```
for(i=0;s1[i];i++)
    if(i%2==1&&s1[i]%2==1) continue;
    else s2[j++]=s1[i];
s2[j]='\0';
strcpy(s1,s2);
```

方法二：直接删除字符数组中符合条件的字符串，即把需要保留的字符重新组织存到原字符数组中，而要删除的字符不保存。代码如下：

```
for(i=0;s1[i];i++)
    if(i%2==1&&s1[i]%2==1) continue;
    else s1[j++]=s1[i];
s1[j]='\0';
```

【例 5-18】 通过键盘输入包含字母和 * 号的字符串，将字符串头部连续的 * 号全部移到字符串的尾部。

算法设计如下。

(1) 定义字符数组 str，输入符合规则的字符串存入 str 数组中。

(2) 统计保存在数组 str 中的字符串，其头部连续 * 号的个数，将统计结果存入变量 n 中。

(3) 删除保存在数组 str 中的字符串的头部连续 * 号。

(4) 在数组 str 中的字符串的尾部补充 n 个 * 号。

(5) 输出字符数组 str。

程序代码：

```
#include<stdio.h>
void fun(char str[])
{
```

```
    int i = 0,n = 0,j = 0;
    while(str[n] == '*')   /* 统计保存在数组 str 中的字符串头部连续 * 号的个数,存入 n 中 */
        n++;
    i = n;
    while(str[i]) /* 将 str 中的字符串前移,覆盖掉头部 n 个连续 * 号 */
    {
        str[j++] = str[i];
        i++;
    }
    while(n!= 0)            /* 在 str 中的字符串的后面补充 n 个 * 号 */
    {
        str[j] = '*';
        j++;
        n--;
    }
}
void main()
{
    char str[100];
    gets(str);
    fun(str);
    puts(str);
}
```

输入：

```
****a**b**cd**↙
```

运行结果：

```
a**b**cd******
```

程序说明如下。

(1) 循环语句 while(str[n]==='*') n++; 的功能是统计字符串头部的所有连续'*'号的个数 n,同时记录字符串从前数第 1 个不是'*'的字符下标。

(2) 循环语句 while(str[i])的{str[j++]=str[i]; i++;}的作用是将下标从 i 开始的字符向前移动,即将字符串头部所有连续'*'号覆盖掉。

(3) 循环语句 while(n!=0){str[j]='*'; j++; n--;}的作用是在已被前移的字符后填补 n 个'*'号。

【例 5-19】 输入 5 个字符串,找出字符串中的最大值并输出。

算法设计如下。

找若干个字符串中的最大值或最小值,如同求若干个数中的最大值或最小值(见例 5-3),其算法相同,但要注意的是字符数组之间的赋值与比较必须用字符串函数进行。

程序代码：

```
#include <string.h>
#include <stdio.h>
void fun(char x[][50],char y[])          /* 形参数组 x 是二维数组的,可省略第一维下标 */
{
```

```
    int i;
    strcpy(y,x[0]);                    /* 将数组 x 中的第 1 个字符串存入字符数组 y 中 */
    for(i=1;i<5;i++)
        if(strcmp(x[i],y)>0) strcpy(y,x[i]);
}
void main()
{
    char str[5][50], maxstr[50];
    int i;
    for(i=0;i<5;i++)                   /* 输入 5 个字符串存入字符数组 str 中 */
        gets(str[i]);
    fun(str,maxstr);                   /* 字符数组名做参数 */
    printf("The largest string is: %s\n",maxstr);
}
```

输入：

```
CHINA↙
AMERICA↙
ENGLAND↙
AUSTRALIA↙
JAPAN↙
```

运行结果：

```
The largest string is:JAPAN
```

本章小结

本章主要学习了数组这种构造数据类型。数组是类型相同数据的集合，每个数组元素的数据类型都是相同的。通过本章的学习，掌握一维数组、二维数组及字符数组的定义、初始化及元素引用方法。通过应用实例介绍了一些常用算法，如排序、求最值等，还包括对字符串进行查询、统计、删除、插入等操作的实现算法。

对数组元素的引用是通过下标表达式进行的，操作时一是要注意数组元素的下标从 0 开始。另外一个要注意的问题是 C 编译系统对数组元素的下标不做边界检查。例如：

```
int a[5];
```

只能有效引用 a[0]、a[1]、a[2]、a[3]、a[4]这 5 个元素，如果程序中出现对 a[5]的引用，C 语言的编译系统不会报错，但 a[5]的值是不确定的，也不是用户所需要的。

在 C 语言中，字符串是使用字符数组进行存储的。一般情况下，一维字符数组用来存储一个字符串，二维字符数组用来存储多个字符串。在对字符串进行复制、比较、连接等操作时，不能直接使用关系运算符和赋值运算符，必须使用字符串处理函数来实现。

数组作为函数参数时，有两种形式：一种是数组元素做函数实参，用法与变量相同；另一种是数组名做实参和形参，传递的是数组的首地址。

本章章节知识脉络如图 5-2 所示。

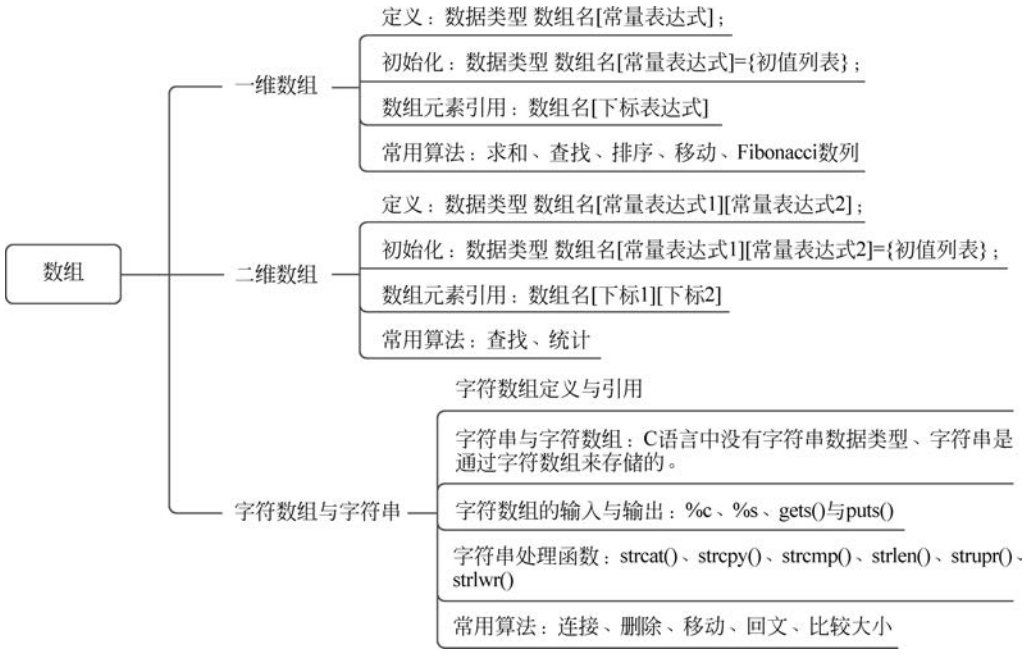


图 5-2　章节知识脉络

习题 5

1. 选择题

(1) 以下叙述中错误的是(　　)。

A. 对于 double 类型数组，不可以直接用数组名对数组进行整体输入或输出

B. 数组名代表的是数组所占存储区的首地址，其值不可改变

C. 当程序执行过程中，数组元素的下标超出所定义的下标范围时，系统将给出“下标越界”的出错信息

D. 可以通过赋初值的方式确定数组元素的个数

(2) 假设 int 类型变量占用两个字节，通过定义语句 int x[10]={0,2,4};，则数组 x 在内存中所占字节数是(　　)。

A. 3　　B. 6　　C. 10　　D. 20

(3) 执行下面的代码后，变量 k 的值是(　　)。

```
int k = 10,s[2];
s[0] = k;k = s[1] * 10;
```

A. 100　　B. 30　　C. 10　　D. 不定值

(4) 在定义 int a[5][4]; 之后，对 a 的引用正确的是(　　)。

A. a[2][4]　　B. a[1,3]　　C. a[4][3]　　D. a[5][0]

(5)若有定义语句 int m[ ][3]={1,2,3,4,5,6,7};，则与该语句等价的是(　　)。

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

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

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

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

(6) 若要求从键盘读入含有空格字符的字符串,应使用函数(　　)。

A. scanf()　　B. getc()　　C. getchar()　　D. gets()

(7) 以下叙述中正确的是(　　)。

A. 字符串常量"str1"的类型是字符串数据类型

B. char str1[]="str1"; 定义的数组包含 4 个元素

C. char str1[]={'g','o','o','d','\0'}; 用赋初值方式来定义字符串

D. 字符数组的每个元素中可存放一个字符,并且最后一个元素值必须是'\0'字符

(8) 以下叙述中正确的是(　　)。

A. 函数 strlen()会返回字符串实际占用内存的大小

B. C 语言本身没有提供对字符串进行整体操作的运算符

C. 两个字符串可以用关系运算符进行大小比较

D. 当两个字符串进行拼接时,结果字符串占用的内存空间是两个原串占用空间之和

(9) 以下叙述中正确的是(　　)。

A. 系统自动在字符串常量"hello!"最后加入'\0'字符,表示串结尾

B. 语句 char str[10] ="hello!"; 和 char str[10] = {"hello!"}; 并不等价

C. 对于一维字符数组,不能使用字符串常量来赋初值

D. 在语句 char str[] ="hello!"; 中,数组 str 的大小与字符串的长度相等

(10) 如下代码段的输出结果是(　　)。

```
char str[] = "Hello";
printf(" %d, %d",sizeof(str),strlen(str));
```

A. 5,5　　B. 5,6　　C. 6,5　　D. 6,6

2. 简答题

(1) 通过键盘输入 10 个整数存入一维数组,编程实现将数组的最大值和最小值交换并将输出新的数组。

(2) 生成并输出下面形式的 5 行 5 列二维数组。

1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9

(3) 从键盘输入一字符串存入数组 str 中,统计其中大写元音字母 A、E、I、O、U 出现的次数并输出。

(4) 输入一字符串存入数组中,逆序存放并输出。

(5) 设计一个函数,模拟字符串比较函数 strcmp(str1,str2)。