```
    int a[3][4]={{1,2,3,4},{9,8,7,6},{-10,10,-5,2}};  //定义数组并赋初值
    max=a[0][0];                                      //先认为 a[0][0]最大
    for (i=0;i<=2;i++)
      for (j=0;j<=3;j++)
        if (a[i][j]>max)                              //如果某元素大于 max,就取代 max 的原值
          {max=a[i][j];
           row=i;                                     //记下此元素的行号
           colum=j;                                   //记下此元素的列号
          }
    printf("max=%d\nrow=%d\ncolum=%d\n",max,row,colum);
    return 0;
}
```

运行结果：

```
max=10
row=2
colum=1
```

最大值为 10，此元素为 a[2][1]。

6.3　字符数组

前已介绍：字符型数据是以字符的 ASCII 代码存储在存储单元中的，一般占一个字节。由于 ASCII 代码也属于整数形式，因此在 C 99 标准中，把字符类型归纳为整型类型中的一种。

由于字符数据的应用较广泛，尤其是作为字符串形式使用，有其自己的特点，因此，在本书中专门加以讨论，希望读者熟练掌握。

C 语言中没有字符串类型，也没有字符串变量，字符串是存放在字符型数组中的。

6.3.1　怎样定义字符数组

用来存放字符数据的数组是字符数组。在字符数组中的一个元素内存放一个字符。

定义字符数组的方法与定义数值型数组的方法类似。例如：

```
char c[10];
c[0]='I'; c[1]=' '; c[2]='a'; c[3]='m'; c[4]=' ';c[5]='h'; c[6]='a'; c[7]='p'; c[8]='p';
c[9]='y';
```

以上定义了 c 为字符数组，包含 10 个元素。赋值以后数组的状态如图 6.11 所示。

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]
I	␣	a	m	␣	h	a	p	p	y

图　6.11

由于字符型数据是以整数形式(ASCII 代码)存放的，因此也可以用整型数组来存放字符数据，例如：

```
int c[10];
c[0]='a';                                    //合法,但浪费存储空间
```

6.3.2 字符数组的初始化

对字符数组初始化,最容易理解的方式是用“初始化列表”,把各个字符依次赋给数组中各元素。例如:

```
chnr c[10]={'I',' ','a','m',' ','h','a','p','p','y'};
```

把10个字符依次赋给c[0]~c[9]这10个元素。

如果在定义字符数组时不进行初始化,则数组中各元素的值是不可预料的。如果花括号中提供的初值个数(即字符个数)大于数组长度,则出现语法错误。如果初值个数小于数组长度,则只将这些字符赋给数组中前面那些元素,其余的元素自动定为空字符(即'\0')。例如:

```
char c[10]={'c',' ','p','r','o','g','r','a','m'};
```

数组状态如图6.12所示。

如果提供的初值个数与预定的数组长度相同,在定义时可以省略数组长度,系统会自动根据初值个数确定数组长度。例如:

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

数组c的长度自动定为10。用这种方式可以不必人工去数字符的个数,尤其在赋初值的字符个数较多时,比较方便。

也可以定义和初始化一个二维字符数组,例如:

```
char diamond[5][5]={{' ',' ','*'},{' ','*',' ','*'},
                    {'*',' ',' ',' ','*'},{' ','*',' ','*'},{' ',' ','*'}};
```

用它代表一个菱形的平面图形,见图6.13。完整的程序见例6.7。

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

图 6.12

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

图 6.13

6.3.3 怎样引用字符数组中的元素

可以引用字符数组中的一个元素,得到一个字符。

【例6.6】 输出一个已知的字符串。

解题思路:先定义一个字符数组,并用“初始化列表”对其赋以初值。然后用循环逐个输出此字符数组中的字符。

编写程序:

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

```
int main()
  {char c[15]={'I',' ','a','m',' ','a',' ','s','t','u','d','e','n','t','.'};
    int i;
    for(i=0;i<15;i++)
      printf("%c",c[i]);
    printf("\n");
    return 0;
  }
```

运行结果：

```
I am a student.
```

【例 6.7】 输出一个菱形图。

解题思路： 先画出一个如图 6.13 所示的平面菱形图案，每行包括 5 个字符，其中有的是空白字符，有的是'*'字符，记下在每行中'*'字符出现的位置。定义一个字符型的二维数组，用“初始化列表”进行初始化。初始化列表中的字符顺序就是图 6.12 中各行中的字符顺序。这样字符数组中已存放了一个菱形的图案。然后用嵌套的 for 循环输出字符数组中的所有元素。

编写程序：

```
#include <stdio.h>
int main()
  {char diamond[][5]={{' ',' ','*'},{' ','*',' ','*'},{'*',' ',' ',' ','*'},
                      {' ','*',' ','*'},{' ',' ','*'}};
    int i,j;
    for (i=0;i<5;i++)
      {for (j=0;j<5;j++)
          printf("%c",diamond[i][j]);
       printf("\n");
      }
    return 0;
  }
```

运行结果：

6.3.4 字符串和字符串结束标志

在 C 语言中，是将字符串作为字符数组来处理的。例 6.6 就是用一个一维的字符数组来存放字符串"I am a student."的，字符串中的字符是逐个存放到数组元素中的。在该例中，字符串的实际长度与数组长度相等。

在实际工作中，人们关心的往往是字符串的有效长度而不是字符数组的长度。例如，定义一个字符数组长度为 100，而实际有效字符只有 40 个。为了测定字符串的实际长度，

C语言规定了一个“字符串结束标志”，以字符'\0'作为结束标志。如果字符数组中存有若干字符，前面9个字符都不是空字符('\0')，而第10个字符是'\0'，则认为数组中有一个字符串，其有效字符为9个。也就是说，在遇到字符'\0'时，表示字符串结束，把它前面的字符组成一个字符串。

注意：C系统在用字符数组存储字符串常量时会自动加一个'\0'作为结束符。例如"C program"共有9个字符。字符串是存放在一维数组中的，在数组中它占10个字节，最后一个字节'\0'是由系统自动加上的。

有了结束标志'\0'后，字符数组的长度就显得不那么重要了。在程序中往往依靠检测'\0'的位置来判定字符串是否结束，而不是根据数组的长度来决定字符串长度。当然，在定义字符数组时应估计实际字符串长度，保证数组长度始终大于字符串实际长度。如果在一个字符数组中先后存放多个不同长度的字符串，则应使数组长度大于最长的字符串的长度。

说明：'\0'代表ASCII码为0的字符，从ASCII码表中可以查到，ASCII码为0的字符不是一个可以显示的字符，而是一个“空操作符”，即它什么也不做。用它来作为字符串结束标志不会产生附加的操作或增加有效字符，只起一个供辨别的标志。

前面曾用过以下语句输出一个字符串。

```
printf("How do you do? \n");
```

在执行此语句时系统怎么知道应该输出到哪里为止呢？实际上，在向内存中存储时，系统自动在最后一个字符'\n'的后面加了一个'\0'，作为字符串结束标志。在执行printf函数时，每输出一个字符检查一次，看下一个字符是否为'\0'，遇'\0'就停止输出。

对C语言处理字符串的方法有以上的了解后，再对字符数组初始化的方法补充一种方法，即用字符串常量来使字符数组初始化。例如：

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

也可以省略花括号，直接写成

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

这里不像例6.6那样用单个字符作为字符数组的初值，而是用一个字符串（注意字符串的两端是用双撇号而不是单撇号括起来的）作为初值。显然，这种方法直观、方便、符合人们的习惯。请注意，此时数组c的长度不是10，而是11。因为字符串常量的最后由系统加上一个'\0'。上面的初始化与下面的初始化等价。

```
char c[] ={'I', ' ', 'a','m', ' ','h','a','p','p','y','\0'};
```

而不与下面的等价：

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

前者的长度为11，后者的长度为10。如果有：

```
char c[10]={"China"};
```

数组c的前5个元素为：'C','h','i','n','a'，第6个元素为'\0'，后4个元素也自动设定为空

字符，见图 6.14。

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

图　6.14

说明：字符数组并不要求它的最后一个字符为'\0'，甚至可以不包含'\0'。像以下这样写完全是合法的：

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

是否需要加'\0'，完全根据需要决定。由于系统在处理字符串常量存储时会自动加一个'\0'，因此，为了使处理方法一致，便于测定字符串的实际长度，以及在程序中作相应的处理，在字符数组中也常常人为地加上一个'\0'。例如：

```
char c[6]={'C','h','i','n','a','\0'};
```

这样做，便于引用字符数组中的字符串。

如定义了以下的字符数组：

```
char c[]={"C program."};
```

由于系统自动在字符串常量的最后一个字符后面加了一个'\0'，因此 c 数组的存储情况如下：

C		p	r	o	g	r	a	m	.	\0

若想用一个新的字符串代替原有的字符串"C program."，如从键盘输入"Hello"分别赋给 c 数组中前面 5 个元素。如果不加'\0'的话，字符数组中的字符如下：

H	e	l	l	o	g	r	a	m	.	\0

新字符串和老字符串连成一片，无法区分开。如果想输出字符数组中的字符串，则会连续输出：

```
Hellogram.
```

如果在"Hello"后面加一个'\0'，它取代了第 6 个字符'g'。在数组中的存储情况为

H	e	l	l	o	\0	r	a	m	.	\0

'\0'是字符串结束标志，如果用以下语句输出数组 c 中的字符串：

```
printf("%s\n",c);                    //输出数组 c 中的字符串
```

在输出字符数组中的字符串时，遇'\0'就停止输出，因此只输出了字符串"Hello"。而不会输出"Hellogram."。

从这里可以看到在字符串末尾加'\0'的作用。

6.3.5 字符数组的输入输出

字符数组的输入输出可以有两种方法。

(1) 逐个字符输入输出。用格式符“%c”输入或输出一个字符,如例6.6。

(2) 将整个字符串一次输入或输出。用“%s”格式符,意思是对字符串(string)的输入输出。例如:

```
char c[]={"China"};
printf("%s\n",c);
```

在内存中数组c的存储情况为

C	h	i	n	a	\0

输出时,遇结束符'\0'就停止输出。输出结果为

```
China
```

说明:

(1) 输出的字符中不包括结束符'\0'。

(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,而不是数组元素名。写成下面这样是不对的:

```
printf("%s",c[0]);                                    //c[0]不是字符数组名,而是一个数组元素
```

(3) 如果数组长度大于字符串的实际长度,也只输出到遇'\0'结束。例如:

```
char c[10]={"China"};                                 //字符串长度为5,连'\0'共占6个字节
printf("%s",c);
```

只输出字符串的有效字符“China”,而不是输出10个字符。这就是用字符串结束标志的好处。

(4) 如果一个字符数组中包含一个以上'\0',则遇第一个'\0'时输出就结束。

(5) 可以用scanf函数输入一个字符串。例如:

```
scanf("%s",c);
```

scanf函数中的输入项c是已定义的字符数组名,输入的字符串应短于已定义的字符数组的长度。例如,已定义:

```
char c[6];
```

从键盘输入:

```
China↙
```

系统会自动在China后面加一个'\0'结束符。如果利用一个scanf函数输入多个字符串,则应在输入时以空格分隔。例如:

```
char str1[5],str2[5],str3[5];
```

```
scanf("%s%s%s",str1,str2,str3);
```

输入数据：

How are you? ↙

由于有空格字符分隔，作为 3 个字符串输入。在输入完后，str1，str2 和 str3 数组的状态如下：

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

数组中未被赋值的元素的值自动置'\0'。若改为

```
char str[13];
scanf("%s",str);
```

如果输入以下 12 个字符：

How are you? ↙

由于系统把空格字符作为输入的字符串之间的分隔符，因此只将空格前的字符"How"送到 str 中。把"How"作为一个字符串处理，故在其后加'\0'。str 数组的状态为

H	o	w	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

注意：scanf 函数中的输入项如果是字符数组名，不要再加地址符 &，因为在 C 语言中数组名代表该数组第一个元素的地址(或者说数组的起始地址)。下面写法不正确：

```
scanf("%s",&str);                    //str 前面不应加 &
```

分析图 6.15 所示的字符数组，若数组占 6 个字节。数组名 c 代表地址 2000。可以用下面的输出语句得到数组第一个元素的地址。

```
printf("%o",c);                      //用八进制形式输出数组 c 的起始地址
```

可以得到数组 c 的起始地址(例如 2000)。可知数组名 c 代表数组起始地址。

c 数组

2000	C
2001	h
2002	i
2003	n
2004	a
2005	\0

图　6.15

(6) 前面介绍的输出字符串的方法：

```
printf("%s",c);
```

实际上是这样执行的：按字符数组名 c 找到其数组第一个元素的地址，然后逐个输出其中的字符，直到遇'\0'为止。

6.3.6　使用字符串处理函数

在 C 函数库中提供了一些用来专门处理字符串的函数，使用方便。几乎所有版本的 C 语言编译系统都提供这些函数。下面介绍几种常用的函数。

1. puts 函数——输出字符串的函数

其一般形式为

puts(字符数组)

其作用是将一个字符串(以'\0'结束的字符序列)输出到终端。假如已定义 str 是一个字符数组名,且该数组已被初始化为"China"。则执行:

```
puts(str);
```

其结果是在终端上输出"China"。由于可以用 printf 函数输出字符串,因此 puts 函数用得不多。

用 puts 函数输出的字符串中可以包含转义字符。例如:

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

输出:

```
China
Beijing
```

在用 puts 输出时将字符串结束标志'\0'转换成'\n',即输出完字符串后换行。

2. gets 函数——输入字符串的函数

其一般形式为

gets(字符数组)

其作用是从终端输入一个字符串到字符数组,并且得到一个函数值。该函数值是字符数组的起始地址。如执行下面的函数:

```
gets(str);                              /str 是已定义的字符数组
```

如果从键盘输入:

```
Computer↙
```

将输入的字符串"Computer"送给字符数组 str(请注意,送给数组的共有 9 个字符,而不是 8 个字符),返回的函数值是字符数组 str 的第一个元素的地址。一般利用 gets 函数的目的是向字符数组输入一个字符串,而不大关心其函数值。

注意:用 puts 和 gets 函数只能输出或输入一个字符串,不能写成

```
puts(str1,str2);
```

或

```
gets(str1,str2);
```

3. strcat 函数——字符串连接函数

其一般形式为

strcat(字符数组1,字符数组2)

strcat是STRing CATenate(字符串连接)的缩写。其作用是把两个字符数组中的字符串连接起来,把字符串2接到字符串1的后面,结果放在字符数组1中,函数调用后得到一个函数值——字符数组1的地址。例如:

```
char str1[30]={"People's Republic of "};
char str2[]={"China"};
printf("%s", strcat(str1, str2));
```

输出:

```
People's Republic of China
```

连接前后的状况见图6.16所示。

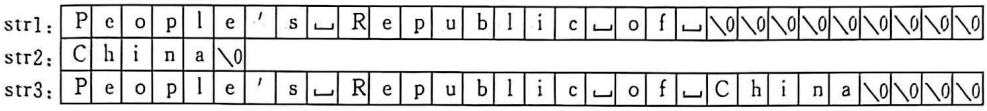

图 6.16

说明:

(1) 字符数组1必须足够大,以便容纳连接后的新字符串。本例中定义str1的长度为30,是足够大的,如果在定义时改用str1[]={"People's Republic of"},就会出问题,因长度不够。

(2) 连接前两个字符串的后面都有'\0',连接时将字符串1后面的'\0'取消,只在新串最后保留'\0'。

4. strcpy和strncpy函数——字符串复制函数

其一般形式为

strcpy(字符数组1,字符串2)

strcpy是STRingCoPY(字符串复制)的简写。它表示"字符串复制函数",作用是将字符串2复制到字符数组1中去。例如:

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

执行后,str1的状态如下:

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

说明:

(1) 字符数组1必须定义得足够大,以便容纳被复制的字符串2。字符数组1的长度不应小于字符串2的长度。

(2) "字符数组1"必须写成数组名形式(如str1),"字符串2"可以是字符数组名,也可以是一个字符串常量。例如:

```
strcpy(str1,"China");
```

作用与前面相同。

(2) 如果在复制前未对 str1 数组初始化或赋值，则 str1 各字节中的内容是无法预知的，复制时将 str2 中的字符串和其后的'\0'一起复制到字符数组 1 中，取代字符数组 1 中的前面 6 个字符，最后 4 个字符并不一定是'\0'，而是 str1 中原有的最后 4 个字节的内容。

(3) 不能用赋值语句将一个字符串常量或字符数组直接给一个字符数组。字符数组名是一个地址常量，它不能改变值，正如数值型数组名不能被赋值一样。如下面两行都是不合法的：

```
str1="China";          //企图用赋值语句将一个字符串常量直接赋给一个字符数组
str1=str2;             //企图用赋值语句将一个字符数组直接赋给另一个字符数组
```

只能用 strcpy 函数将一个字符串复制到另一个字符数组中去。用赋值语句只能将一个字符赋给一个字符型变量或字符数组元素。如下面的语句是合法的：

```
char a[5],c1,c2;
c1='A';c2='B';
a[0]='C'; a[1]='h'; a[2]='i'; a[3]='n'; a[4]='a';
```

(5) 可以用 strncpy 函数将字符串 2 中前面 n 个字符复制到字符数组 1 中去。例如：

```
strncpy(str1,str2,2);
```

作用是将 str2 中最前面 2 个字符复制到 str1 中，取代 str1 中原有的最前面 2 个字符。但复制的字符个数 n 不应多于 str1 中原有的字符(不包括'\0')。

5. strcmp 函数——字符串比较函数

其一般形式为

strcmp(字符串 1,字符串 2)

strcmp 是 STRing CoMPare(字符串比较)的缩写。它的作用是比较字符串 1 和字符串 2。例如：

```
strcmp(str1,str2);
strcmp("China","Korea");
strcmp(str1,"Beijing");
```

说明：字符串比较的规则是：将两个字符串自左至右逐个字符相比(按 ASCII 码值大小比较)，直到出现不同的字符或遇到'\0'为止。

(1) 如全部字符相同，则认为两个字符串相等；

(2) 若出现不相同的字符，则以第 1 对不相同的字符的比较结果为准。例如：

"A"<"B","a">"A","computer">"compare","these">"that","1A">"$20","CHINA">"CANADA","DOG"<"cat","Tsinghua">"TSINGHUA"

说明：如果参加比较的两个字符串都由英文字母组成，则有一个简单的规律：**在英文字典中位置在后面的为"大"**。例如 computer 在字典中的位置在 compare 之后，所以"computer">"compare"。但应注意小写字母比大写字母"大"，所以"DOG"<"cat"。

比较的结果由函数值带回。

(1) 如果字符串 1 与字符串 2 相同，则函数值为 0。

(2) 如果字符串 1＞字符串 2，则函数值为一个正整数。

(3) 如果字符串 1＜字符串 2，则函数值为一个负整数。

注意：对两个字符串比较，不能用以下形式：

```
if(str1>str2)
    printf("yes");
```

因为 str1 和 str2 代表地址而不代表数组中全部元素，而只能用

```
if(strcmp(str1,str2)>0)
    printf("yes");
```

这时，系统分别找到两个字符数组的第一个元素，然后顺序比较数组中各个元素的值。

6. strlen 函数——测字符串长度的函数

其一般形式为

strlen（字符数组）

strlen 是 STRing LENgth(字符串长度)的缩写。它是测试字符串长度的函数。函数的值为字符串中的实际长度(不包括'\0'在内)。例如：

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

输出结果不是 10，也不是 6，而是 5。也可以直接测试字符串常量的长度，例如：

```
strlen("China");
```

7. strlwr 函数——转换为小写的函数

其一般形式为

strlwr（字符串）

strlwr 是 STRing LoWeRcase（字符串小写)的缩写。函数的作用是将字符串中大写字母换成小写字母。

8. strupr 函数——转换为大写的函数

其一般形式为

strupr（字符串）

strupr 是 STRing UPpeRcase(字符串大写)的缩写。函数的作用是将字符串中小写字母换成大写字母。

以上介绍了常用的 8 种字符串处理函数，应当再次强调：库函数并非 C 语言本身的组成部分，而是 C 语言编译系统为方便用户使用而提供的公共函数。不同的编译系统提供的函数数量和函数名、函数功能都不尽相同，使用时要小心，必要时查一下库函数手册。当然，有一些基本的函数(包括函数名和函数功能)，不同的系统所提供的是相同的，这就为程序的

通用性提供了基础。

列出以上字符串函数是为使读者了解怎样用字符串函数去处理字符串的运算。如果不了解这些函数,难以正确有效地进行字符串的运算。但是不必死记,从这些函数规定的名字大体可以猜到它们的含义,用到时查一下即可。

注意:在使用字符串处理函数时,应当在程序文件的开头用

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

把 string.h 文件包含到本文件中。

6.3.7 字符数组应用举例

【例 6.8】 输入一行字符,统计其中有多少个单词,单词之间用空格分隔开。

解题思路:问题的关键是怎样确定"出现一个新单词了"。可以采取这样的方法:从第 1 个字符开始逐个字符进行检查,判断此字符是否是新单词的开头,如果是,就使变量 num 的值加 1(用变量 num 统计单词数),最后得到的 num 的值就是单词总数。

判断是否出现新单词,可以由是否有空格出现来决定(连续的若干个空格作为出现一次空格;一行开头的空格不统计在内)。如果测出某一个字符为非空格,而它的前面的字符是空格,则表示"新的单词开始了",此时使 num(单词数)累加 1。如果当前字符为非空格而其前面的字符也是非空格,则意味着仍然是原来那个单词的继续,num 不应再累加 1。用变量 word 作为判别当前是否开始了一个新单词的标志,若 word=0 表示未出现新单词,如出现了新单词,就把 word 置成 1。

前面一个字符是否为空格可以从 word 的值看出来,若 word 等于 0,则表示前一个字符是空格;如果 word 等于 1,意味着前一个字符为非空格,可以用图 6.17 表示。

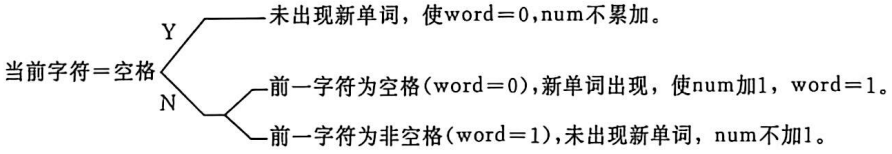

图 6.17

以输入"I am a boy."为例,说明在对每个字符作检查时的有关参数状态,见表 6.1 所示。

表 6.1 输入"I am a boy.",有关参数状态

当前字符	I		a	m		a		b	o	y	.
是否空格	否	是	否	否	是	否	是	否	否	否	否
word 原值	0	1	0	1	1	0	1	0	1	1	1
新单词开始否	是	否	是	否	否	是	否	是	否	否	否
word 新值	1	0	1	1	0	1	0	1	1	1	1
num 值	1	1	2	2	2.	3	3	4	4	4	4