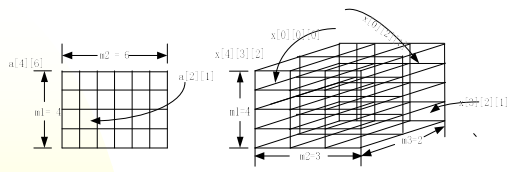
4.3 多维数组



行下标 i
列下标 j

页下标 i
行下标 j
列下标 k

图 二维数组和三维数组逻辑结构示意图

♦二维数组的定义

如果一维数组的每个元素本身也是一个一维数组，则形成了一个二维数组。这时就要用两个下标来表示它的每个数组元素。

二维数组的定义格式为：

类型说明符 数组名 [常量表达式1][常量表达式2];

例如： int a[3][4];

定义一个二维数组，数组名为a，数组元素有3行4列，每个数组元素都是一个基型数据。

```
float c[2][2][3];
  /* 定义一个2×2×3的三维数组 */
```

二维数组定义的注意事项：

1. 元素类型说明符、数组名及常量表达式的要求与一维数组相同。
2. 常量表达式1和常量表达式2各在一个方括号内。

例如：

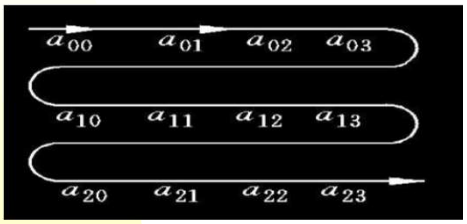
定义不能写成：

```
int  a[3, 4]; ✘
int  a{3}{4}; ✘
```

而应该写成：

```
int  a[3][4]; ✓
```

3.二维数组的元素在内存中的存放顺序为“按行存放”，即先顺序存放第一行的元素，再存放第二行的元素，依此类推。例如：对a[2][4]，其元素的存放顺序,从图中可以看出，最右边的下标变化最快，最左边的下标变化最慢。这一特点也适用于二维以上的多维数组。



例如，对于前面例子中的三维数组c[2][2][3]，其元素的存放顺序为

c[0][0][0]、c[0][0][1]、c[0][0][2]、
c[0][1][0]、c[0][1][1]、c[0][1][2]、
c[1][0][0]、c[1][0][1]、c[1][0][2]、
c[1][1][0]、c[1][1][1]、c[1][1][2]。

4. 可把二维数组可以看成是一种特殊的一维数组，其特殊之处就在于它的元素又是一个一维数组。

例如，二维数组a[3][4]可以理解为：它有三个元素a[0]、a[1]、a[2]，每一个元素都又是一个包含4个元素的一维数组。如图所示。

二维数组名	一维数组名	数组元素
a	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]

二维数组元素的引用格式：

数组名[下标1][下标2]

例如：float a[2][3], b[3][4];

b[2][3]= b[2][3]+a[1][2];

说明：

1. 下标是整型常量、整型变量或整型表达式。
2. 0≤下标值＜对应维的元素个数。

例如：上例 b[2][3]= b[2][3]+a[1][2];

不能写成 b[2][3]= b[2][3]+a[2][3];

3. 注意：数组说明与数组引用的区别。

行下标和列下标都越界

二维数组的赋值（3种方法）

方法1：在定义数组的同时为数组元素赋初值。

♦按行对二维数组赋初值。

●对全部元素赋初值，例如：

int a[3][4]={{8, 7, 9, 5}, {6, 2, 4, 1}, {3, 10, 84, 77}};

●按元素的存储顺序给数组元素赋初值

int a[3][4]={8, 7, 9, 5, 6, 2, 4, 1, 3, 10, 84, 77};

其初始化结果可用一个二维表表示，如图所示：

	[0]	[1]	[2]	[3]
a[0]	8	7	9	5
a[1]	6	2	4	1
a[2]	3	10	84	77

●按行给数组的部分元素赋初值例如：

int a[3][4]={{88}, {0, 0, 76}};

其初始化结果可用一个二维表表示，如图所示：

	[0]	[1]	[2]	[3]
a[0]	88	0	0	0
a[1]	0	0	76	0
a[2]	0	0	0	0

没被赋值的自动置为0值

●按元素的存储顺序给前面部分元素赋初值。

int a[3][4]={88, 0, 0, 76};

	[0]	[1]	[2]	[3]
a[0]	88	0	0	76
a[1]	0	0	0	0
a[2]	0	0	0	0

♦按元素的存储顺序赋初值，不指定第一维的元素个数，但第二维长度不能省略。

例如： int a[][4] = {1, 2, 3, 4, 5, 6};

/* 等价：int a[2][4]= … */

1	2	3	5
5	6	0	0

♦用按行赋初值方法，对各行的部分或全部元素赋初值，并省略第一维的元素个数。

例如：int a[][4] = {{1, 2}, {0}};

/* 等价：int a[2][4]= … */

1	2	0	0
0	0	0	0

方法2：在程序运行时，用赋值语句为数组元素赋值。

例如：
```
for (i=0;i<=10;i++)
    for(j=0;j<=20;j++)
        a[i][j]=i+j;
```

方法3：在程序运行时，用输入语句为数组元素赋值。

例如：
```
for (i=0;i<=10;i++)
    for (j=0;j<=20;j++)
        scanf("%d ",,&a[i][j]);
```

- 多维数组程序示例

有一个3×4的矩阵，求出其中值最大的那个元素及其所在的行号和列号.

```
#include <stdio.h>
void main()
{ int i,j,row=0,col=0,max;
  int a[3][4]={{1,2,3,4},{9,8,7,6},
               {-10,10,-5,2}};
  max=a[0][0];
```

```
for (i=0;i<=2;i++)
  for (j=0;j<=3;j++)
    if (a[i][j]>max)
    {  max=a[i][j];
       row=i;
       col=j;
    }
    printf("max=%d,row=%d,column=%d\n",
           max,row,col);
}
```

输出结果为：max=10,row=2,column=1

二维数组编程示例

【例4.12】把某月的第几天转换成年的第几天。

题解：

- 为确定一年中的第几天，需要一张每月的天数表，该表给出每个月份的天数。
- 由于二月份天数因闰年和非闰年有所不同，为程序处理方便，事先将每个月份的天数存在二维数组中，以区别闰年(leap year)和非闰年。

计算闰年的条件

是leap!=0，leap的计算为

```
leap= year%4==0 && year%100!=0||year%400==0;
```

程序实现：

```
#include <stdio.h>
int days[][12] =
{{31,28,31,30,31,30,31,31,30,31,30,31},//非闰年
{31,29,31,30,31,30,31,31,30,31,30,31}};//闰年
void main(){ int year, month, day, leap, i;
  printf("Input year, month, day.\n");
  scanf("%d,%d,%d", &year, &month, &day);
  leap= year%4==0 && year%100 || year%400==0;
  if(month<1||month>12||day<1||day> days[leap][month-1])
     return; /* 非法的月份或者日子 */
  for(i = 0; i < month-1; i++)day+=days[leap][i];
  // 将该月之前的天数累加
  printf("\nThe days in year is %d.\n", day);
}
```

【第二版例4.14】输入年与年中的第几天，计算这一天是这年的哪一月和哪一日。

【解题思路】

为从一年中的第几天day，确定这天的月month和日day，与例4.13一样，也需要一张每月的天数表。为了计算month和day，程序首先确定这一年是平年还是闰年，然后根据各月的天数表，逐一用每月的天数与day比较，如果day比当前月的天数大，就从day中减去这月的天数，并继续考虑下一个月。如果day小于等于当前月的天数，那么当前月就是要求的月份，day就是要求的这一月的日。

```
#include <stdio.h>

int dayTable[][12] =
{{31, 28, 31, 30, 31, 30, 31, 31, 30, 31,30, 31},
{31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

char monthname[][10] =
    { "January", "February", "March",
      "April", "May", "June", "July",
      "August", "September", "October",
      "November", "December"
    };
```

```
int main()
{  int year, day, leap, i;
   printf("输入年和年中的日。\n");
   scanf("%d%d", &year, &day);
   leap = year%4  == 0 && year%100 ||
year%400 == 0;
   for(i = 0; day > dayTable[leap][i]; i++)
      day -= dayTable[leap][i];
   printf("\n这月的英文名是 %s, 这天是 %d日
\n", monthname[i], day);
   return 0;
}
```

【例4.13】输入整数 n ，输出以下形式的二维数组。

1	2	3	…	n
n	1	2	…	n-1
n-1	n	1	…	n-2
			…	
2	3	4	…	1

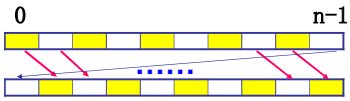
当n＝5时，

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

▪注：上面问题有多种解法

算法(1)：

1. 先给出第 0 行；
2. 利用第i行与上一行的关系，求出第i行；
   (1) 先求第0列（为上一行最后一列）；
   (2) 后求第1列至最后一列（上一行左边一列）。



程序片段：

```
for(j = 0; j < n ; j++)
    a[0][j] = j+1;          /* 先生成第0行 */
for(i = 1; i < n; i++){     /* 从第1行起 */
   a[i][0]=a[i-1][n-1];     /* 生成i行的第0列 */
for(j = 1; j < n; j++)      /* 生成i行的其余列 */
   a[i][j]=a[i-1][j-1];/* 为上1行左边1个元素 */
}
```

(算法2)将i行的内容分成两部分：
i~n-1列的值为1~n-i，0~i-1列的值为n-i+1~n。为了简便，引入赋值变量k，k的初值为1，每次赋值后增1。

实现代码如下：

```
for(i = 0; i < n; i++){
   for(k = 1,j = i; j < n; j++)
       a[i][j] = k++; /*右上部分
*/
   for(j = 0; j < i; j++)
       a[i][j] = k++; /*左下部分
*/
}
```

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

算法(3)：

将解法2中的变量k去掉，i~n-1列的j列元素a[i][j]的值是j-i+1；0~i-1列的j列的值是j-i+1+n。

得到以下实现代码：

```
for(i = 0; i < n; i++)
{
   for(j = i; j < n; j++)
     a[i][j] = j - i + 1;
   for(j = 0; j < i; j++)
     a[i][j] = j - i + 1 + n;
}
```

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

算法(4)：

将解法3中的两个j循环合并：

```
for(i = 0; i < n; i++)
  for(j = 0; j < n; j++)
     a[i][j]=j<i?j-i+1+n:j-i+1 ;
```

算法(5)将4中的条件表达式简化：

```
for(i = 0; i < n; i++)
  for(j = 0; j < n; j++)
      a[i][j] = (j - i + n) % n  +  1
```

对于第 0 行，a[0][j] = (j - 0 + n) % n + 1 = j + 1

对于第 1 行，a[1][0] = (0 - 1 + n) % n + 1 = n

a[1][1] = (1 - 1 + n) % n + 1 = 1

a[1][n-1] = (n - 1 - 1 + n) % n + 1 = n - 1

对于第 n-1 行，

a[n-1][0] = (0 - (n - 1) + n) % n + 1 = 2

a[n-1][1] = (1 - (n - 1) + n) % n + 1 = 3

a[n-1][n-2] = (n - 2 - (n - 1) + n) % n + 1 = n

a[n-1][n-1] = (n - 1 - (n - 1) + n) % n + 1 = 1

对照题意，发现"a[i][j] = (j - i + n) % n + 1" 成立。

算法(6)：

先给出0行，然后由i行与0行的关系，填写i行。

设i行的j列元素a[i][j]等于0行的k列元素a[0][k]。

则k的计算式如下：

当j >= i时：k = j - i

当j < i时：k = j - i + n

计算k的代码可以写成

```
k = j - i >= 0 ? j - i : j - i + n;
```

或简写成

```
/*  k = (j - i + n) % n  */
```

实现代码如下：

```
for(j = 0; j < n; j++) a[0][j] = j + 1;
  for(i = 1; i < n; i++)
    for(j = 0; j < n; j++)
  /* k = ?; (1)  j - i >= 0 : k = j - i
            (2)  j - i < 0  : k = j - i + n
     k = j - i >= 0 ? j - i : j - i + n;
     或 (j - i + n) % n */
      a[i][j] = a[0][(j - i + n) % n];
```

算法(7)：

对i行，从i列开始顺序将整数1至n填入到数组中。

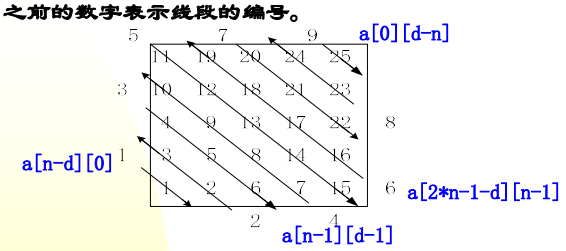
为数组的i行元素填写数字，可从i列开始顺序填写n次。

j从i开始顺序递增，当j大于等于n时，用运算符%，从j减去n，改为从0列开始，即填写的元素是a[i][j%n]。要填写的值等于j-i+1。

```
for(i = 0; i < n; i++)
   for(j = i; j < i+n; j++)
/* i行，从j列开始填数，共填数n次 */
      a[i][j%n] = j-i+1;
```

还有许多种解法，请读者自己想出方法，并写出相应的实现代码。

【例4.14】形成以下形式的二维数组(以下示例是n=5)，并输出。示例中有向斜线段表示填数方向，有向斜线段之前的数字表示线段的编号。



解题思路：n*n的二维数组，共有2*n-1条斜线，将它们顺次编号为1至2*n-1。

第奇数条斜线从左上往右下；第偶数条斜线从右下往左上。

程序按从左下角至右上角的顺序逐条斜线填数。斜线的填数方向按斜线的奇偶性交替变化。

对于下三角，第奇数条斜线从左上往右下，起始行号为n-d（d为斜线编号，下同），起始列号为0；第偶数条斜线从右下往左上，起始行号为n-1，起始列号为d-1。

对于上三角，第奇数条斜线从上往下，起始行号为0，起始列号为d-n；第偶数条斜线从下往上，起始行号为2*n-1-d，起始列号为n-1。按以上情况分类，写出程序如下：

```
#include  <stdio.h>
#define MAXN 10
int a[MAXN][MAXN];
void main()
{ int i, j, k, d, n;
  while (1) {  /* n 不超过10 */
    printf("Enter n   "); scanf("%d", &n);
    if (n >= 1 && n <= 10) break;
    printf("Error! n must be in [1, 10]\n");
  }
```

```
for(k = d = 1; d <= 2*n-1; d++)
{   //共有2*n-1条斜线
    if (d <= n-1)  /* 左下三角 */
     if (d % 2) /* 奇数号斜线，从左上往右下 */
        for(i = n-d, j = 0; i < n; i++, j++)
          a[i][j] = k++;
     else  /* 偶数号斜线，从右下往左上 */
        for(i=n-1,j = d-1; i >= n-d; i--, j--)
          a[i][j] = k++;
```

```
   else   /* d >= n，右上三角 */
    if (d % 2)
      for(i=0,j=d-n; i<=2*n-1-d; i++, j++)
         a[i][j] = k++;
    else
      for(i=2*n-1-d,j=n-1; i>=0; i--, j--)
         a[i][j] = k++;
  }
  for(i = 0; i< n; i++) {
    for(j=0; j<n;j++) printf("%4d", a[i][j]);
   printf("\n");
 }
}
```

●多维数组

二维数组实际上是一种最简单的多维数组。C语言允许使用高于二维的多维数组，如三维数组、四维数组甚至更高维数的数组，允许使用的数组的最大维数由不同的C编译器决定。

在实际应用中，经常用到的是一维、二维和三维数组，四维以上的数组极少使用。

♦多维数组的定义格式为：

类型说明符 数组名[常量1][常量2]...[常量n];

其中，维数由常量表达式的个数n来决定。若n为3，则是一个三维数组。

例如： int a[2][3][4];

定义了一个三维数组，可以理解为：三维数组a包含2个二维数组(a[0]和a[1])，每个二维数组包含3个一维数组，而每个一维数组包含4个float型的数组元素(如a[0][0]包含a[0][0][0]、a[0][0][1]、a[0][0][2]和a[0][0][3])。

int a[2][3][4];

对三维数组a组成的理解如下图所示：

三维数组名	二维数组名	一维数组名	数组元素
a	a[0]	a[0][0]	a[0][0][0],a[0][0][1],a[0][0][2],a[0][0][3]
		a[0][1]	a[0][1][0],a[0][1][1],a[0][1][2],a[0][1][3]
		a[0][2]	a[0][2][0],a[0][2][1],a[0][2][2],a[0][2][3]
	a[1]	a[1][0]	a[1][0][0],a[1][0][1],a[1][0][2],a[1][0][3]
		a[1][1]	a[1][1][0],a[1][1][1],a[1][1][2],a[1][1][3]
		a[1][2]	a[1][2][0],a[1][2][1],a[1][2][2],a[1][2][3]

对于一个多维数组，它的元素在内存中的存放顺序有这样特点：第一维的下标变化最慢，最右边的下标变化最快。

引入多维数组可以使编程更为灵活，因为多维数组的每一维都可以根据实际情况的不同而赋予不同的含义，从而使多维数组能描述比较复杂的数据结构。

例如：

float score[班级数][学号数][科目数];

在这个数组中的一个元素

score[班级][学号][科目]可以定义成班级、学号所确定的那名学生的某个科目的成绩。

例 某年级共有4个班，每班各有30名学生，有6个科目的考试成绩。求各班每个学生的平均成绩并输出之。

```
#include <stdio.h>
void main( ){
float score[4][30][6],studav[4][30];   int i,j,k;    float sum;
for(i=0;i<4;i++)
  for(j=0;j<30;j++)
     for(k=0;k<6;k++)
        { printf("请输入%d班学号为%d的学生的科目%d成绩", i+1,j+1,k+1);
          scanf("%f",&score[i][j][k]);
        }
 for(i=0;i<4;i++)
    for(j=0;j<30;j++)
    {   sum=0;
        for(k=0;k<6;k++)
            sum=sum+score[i][j][k];
        studav[i][j]=sum/6;
       printf("%d班学号为%d的学生的平均成绩studav[%d][%d]
为:\n",i+1,j+1,i,j,studav[i][j]);
    }
}
```

4.4字符串处理技术基础

●字符数组与字符串

字符串：用双引号括起来的一串字符。

例如:"How are you! "或"Hello, world!"

字符数组：用来存放字符型数据的数组。

例如：char c[13];

字符串结束标志：

C语言规定了一个字符串结束标志'\0'，代表ASCII码为0的字符，它是一个“空操作，即什么也不干。由它来作为字符串结束标志，其作用是用来测定字符串的实际长度。

●字符串的存储：

♦C语言中没有专门存放字符串的变量，而是采用字符数组存放。字符数组中的每个元素存放字符串的一个字符。

♦字符数组在存放字符串时，字符串末尾的结束符'\0'也一并存放。所以，一个字符串用一维数组来装入时，数组元素个数一定要比字符数多一个。

例如：

char c[13]="How are you! ";

定义了一个字符数组c，存放的字符串中字符数为12，考虑到还有1个字符串结束符'\0'，定义的数组长度不能少于13。

●字符数组的定义

定义与普通数组类似，一般形式为：

char 字符数组名[元素个数];

例如：char s[5] ;

表示数组s有5个元素，每个元素能存放一个字符，整个数组最多可存放5个字符。

例如:s[0]='C';s[1]='h';s[2]='i';s[3]='n'; s[4]='a';

'C'	'h'	'i'	'n'	'a'

字符数组是存储字符串的一种常用的形式。在字符数组中，每个数组元素存放一个字符。与字符串常量不同的是字符数组并不规定必须在字符串的末尾加上字符串结束符'\0'。但当字符数组内存储的内容需要作为字符串时，就必须要有标志符'\0'。

●字符数组初始化

用字符串常量对字符数组初始化时，系统会在字符列末尾添加一个字符串结束符。

例如

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

或简写为 `char a_str[] = "I am happy!";`

数组a_str[]有12个元素，其中a_str[11]的值是字符串的结束标志符'\0'。

```
char str_list[][30]={"I am happy!",
        "I am learnning c language."};
```

字符数组str_list[0]和字符数组str_list[1]各可存储30个字符，现分别存储着有11个有效字符的字符串，和有26个有效字符的字符串。

称最后有字符串结束符的字符序列为字符串。当字符数组内存储的是字符串时，可用"%s"格式输出，若是普通的字符序列，则它不能用格式"%s"，而只能用格式"%c"。例如

```
char s1[]= "student";//字符串
char s2[]={'s','t','u','d','e','n','t'};
```

则 `printf("%s", s1);` 是正确的。

而 `printf("%s", s2);` 是错误的。

实际上字符数组：

s1有8个元素；

s2只有7个元素。

字符串常量的书写形式为

"字符序列"

其中字符序列可由零个或多个字符组成，如字符串常量"I am a student."含15个有效字符，而字符串常量""不含任何有效字符，其长度为0，习惯称为空字符串。

在字符串常量的书写形式中，双引号"只充当字符串的界限符，不是字符串的一部分。如果字符串要包含字符'"'，则可经过转义序列（如 \" ）来实现，其他转义序列（如\n, \t)也可以作为单个字符出现在字符串常量中。如 "\tThis is a string.\n"

通常字符串写在一行内。如果一个字符串常量在一行内写不下时，可用字符串常量的串接规则把字符串分成连续多行形式书写。

字符串常量串接规则有两条：

(1)在键入字符 '\' 之后紧接键入回车键。

例如

```
"I am a st\ (回车换行符)
 ring."
```

就是字符串常量 "I am a string."。

(2)连续两个紧接的字符串常量相当于一个字符串常量。如 `"I am " "a string."`

也是字符串常量 "I am a string."。

字符串的输入输出可以有两种方式：

(1)用格式 "%c"，结合循环结构逐个字符输入或输出。

(2)用格式 "%s"，将字符串整体地输入或输出。

例如：

```
char s[] = "C language";
```

用第一种方法输出：

```
for(i = 0; s[i]; ++i)
    printf("%c", s[i]);
```

用第二种方法输出：

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

注意以下几点：

(1)字符串与存储字符串的字符数组有区别。

字符串的有效字符是指从所指位置的第一个字符开始至字符串结束标志符之前的那些字符。例如

```
char s[50] = "Pas\0cal Cobol Fortran C";
printf("%s\n", s);
```

将只输出

Pas

而实际上，数组str[]在字符串结束符之后还存有其它许多字符。

(2)用 "%s" 格式输出字符串时，不包括字符串结束标志符。

对应的输出项是字符串或字符串名。字符数组名可作为字符串名。

不能写为：printf("%s", s[]); ×

由于在C语言中规定，数组名代表该数组的首地址。因此整个数组是以首地址开头的一块连续的内存单元。如有字符数组char s[8]，在内存可表示如图。

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]

设数组s的首地址为2000，也就是说s[0]单元地址为2000。则数组名s就代表这个首地址。在执行函数printf("%s",s) 时，按数组名s找到首地址，然后逐个输出数组中各个字符直到遇到字符串终止标志'\0'为止。

(3) 同理在调用scanf()为字符数组输入字符串时，输入项是数组名，不要加地址运算符&。

例如， scanf("%s", s);

在s前面不能再加地址运算符&。如写作scanf(“%s”,&s);则是错误的。 ×

另外:用 "%s" 格式输入字符串时，掠过前导的空白类字符，只能输入一串不含空白类字符的字符序列，遇空白类字符，或输入了格式指定的字符个数，就结束输入。

若要输入一串包括空格符在内的一行字符，要用字符行输入函数gets()。

(4)若用"%c"格式结合循环输入字符序列，若程序又想将输入的字符序列构成字符串，则程序必须用赋值语句在字符列之后存入字符串结束标志符。如

```
char s[20]; int k;
for(k = 0; k < 5; k++)
    scanf("%c", &s[k]);//scanf("%c", s+k);
s[k] ='\0'; /*使输入的k个字符构成字符串*/
```

当字符串是由程序逐个字符生成时，程序也必须在它最后接上字符串结束标志符。如

```
for(k = 0; k < 26; k++)
    s[k] ='a'+k;//生成a～z
s[k] ='\0';
```

●字符串的输入和输出

▪用输入函数获得字符串

如果使用输入函数读入字符串，字符数组获得的必然是一个字符串（包含结束符’\0’），而不是字符序列。

输入函数scanf()对字符串的识别标志是空字符：

空格 <SP>，制表符<TAB>和回车符<CR>。

例如键入内容为： a student<CR>

对以下语句： char s[80];
scanf("%s", s);

则数组 s 中获得的是包括结束符’\0’在内的2 个字符：’a’，’\0’。

如果使用输入函数 gets()，则对字符串的识别标志只是回车符。

例如将输入语句改为：gets(s);

则表示将读入完整的一行字符并存入数组 s(包含结束符’ \0’)。该函数调用返回s的存储开始地址。

●字符串输入函数gets()

函数gets(s)的功能是从终端上读入一行字符串到字符数组，以换行符为结束标志，并且将读入的换行符改为'\0'加在在最后。函数返回s的开始地址。

如果输入出错或者没有获得输入值，该函数返回NULL(空)。

程序：

```
#include <stdio.h>
void main()
{ char c;
  while((c=getchar())!= EOF)
   putchar(c);
}
```

将使得每次输入的字符(不超过30个)存入数组s，并且得到同样的输出结果。

```
char s[30];
while(gets(s))
printf("%s\n", s);
```

EOF：字符表示结束符，请同时按下：CTRL/z

●字符串输出函数 puts()

函数调用puts(s)将s的字符串输出到终端，并将s中的\0字符转换成换行符’ \n’。即输出字符串内容后，并换行。所以puts(s)相当于printf("%s\n",s)。

```
char s[100];
while( gets(s) )
    puts(s);//printf("%s\n", s);
```

●一些常用字符串处理函数

(1)求字符串长度函数 strlen()

函数功能： strlen(s)返回str中的有效字符(不包括'\0')个数。

```
char s1[30] = "C language";int i;
i = strlen(s1);//10
```

方案一：用下标编程实现

```
int k = 0;char str[100];
                ...
while(str[k]) k++;
```

方案二:用调用函数实现

```
int k ;char str[100];
                ...
K= strlen(s);
```

(2)字符串拷贝函数strcpy()

函数功能:strcpy(s1,s2)将字符串s2复制拷贝到字符数组s1中，包括’\0’ 。其中，s1不能是字符串常量。

如果复制成功，返回 s1 的值（其对象的地址）；如果复制失败，返回NULL。

限定字符数组s1要足够大，以便能容纳被拷贝的s2的全部内容。程序实现： 方案一：用下标编程实现

方案二:用调用函数实现

```
char s1[50],s2[20] =
     "this is a string";
strcpy(s1 ,s2);
```

```
int i=0;char s1[50],s2[30];
while( s2[i] ){
       s1[i] = s2[i];
       i++;
    }
s1[i] = '\0';
```

(3) 字符串拷贝函数strncpy()

函数功能：strncpy(s1, s2, n)的作用是将s2中的前n个字符拷贝到s1(并加’\0′)。

其中n是整型表达式，指明欲拷贝的字符个数。如果s2中的字符个数不多于n，则函数调用strncpy(s1, s2, n)

等价于strcpy(s1, s2)。

其中，s1不能是字符串常量。

(4)字符串连接函数 strcat()

函数功能： strcat(s1, s2)将s2内容拷贝接在字符数组s1中的字符串的后面。

其中，s1不能是字符串常量。该函数调用返回s1的开始地址。注意，字符串连接前，s1和s2都各自有’\0′，连接后，s1中原来的’\0′在拷贝时被覆盖掉，而在新的字符串有效字符之后再保留一个’\0′。例如

```
char s1[30] = "Beijing ";
char s2[30] = "Shanghai";
strcat(s1, s2); printf("%s\n", s1);
```

将输出 BeijingShanghai

(5)字符串比较函数 strcmp()

函数功能：strcmp(s1, s2)比较两个字符串的大小，对两个字符串自左至右逐对字符相比较(按字符的ASCII代码值的大小)，直至出现不同的字符或遇到\0字符为止。如直至\0字符，全部字符都相同，则认为相等，函数返回0值；

若出现不同的字符，则以这第一个不相同的字符比较结果为准,若s1的那个不相同字符小于s2的相应字符，函数返回一个负整数；反之，返回一个正整数。

注意，对字符串不允许施行相等"==" 和不相等"!="运算，必须类似于本函数那样，通过逐个字符的比较来实现。

例如，定义了以下数组：

char a[20], b[30];

if(a != b) 或者

if(a == b) 都是错误的。

if(strcmp(a, b) !=0) 或者

if(strcmp(a, b) ==0) 都是正确的。

(6)字符串大写英文字符转换成小写英文字符函数 strlwr()

函数功能： strlwr(s)将存于字符数组s的字符串内的大写英文字符转换成小写英文字符。其中，s不能是字符串常量。

```
#include  <stdio.h>
#include <string.h>
void main()
{     int i=0;  char s[100]; gets(s);
      while(s[i])
      {  if(s[i]>='A'&&s[i]<='Z')s[i]=s[i]+'a'-'A';
         i++;
      }
      puts(s);
}
```

(7)字符串小写字母转换成大写字母函数 strupr()

函数功能： strupr(s)将存于字符数组str的字符串内的小写英文字符转换成大写英文字符。其中，s不能是字符串常量。

```
#include  <stdio.h>
#include <string.h>
void main()
{     int i=0;  char s[100];     gets(s);
      while(s[i])
      {   if(s[i]>='a'&&s[i]<='z')s[i]=s[i]+'A'-'a';
          i++;
      }
      puts(s);
}
```

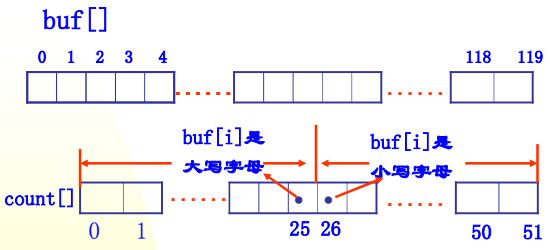
【第二版例4.18】输入字符行，统计各英文字母出现的次数。

题解

设置一个存储字符行的数组 buf[120] 。

设置一个统计 52 个英文字母出现次数的计数器数组count[52]，其中大写字母使用前26个计数器(count[0]-count[25])，小写字母使用后26 个计数器(count[26]-count[51])。

首先，存放输入的字符行。然后顺序扫视buf[]中的字符，若当前字符是英文字母时，对应的计数器count[j]值加1，否则掠过，直至遇字符串结束符时为止。



程序如下：

```
#include  <stdio.h>
void main()
{
  char buf[120]; int  i, count[52];
  printf("Enter letter line.\n");
  gets(buf); /* 读入一行字符 */
  for(i = 0; i < 52; i++)
       count[i] = 0; /* 计算器初始化，清零 */
  for(i=0;buf[i];i++)//顺序扫视buf[]中的字符
     if(buf[i]>= 'A'&& buf[i]<='Z')
        count[buf[i]-'A']++;//出现大写字母
```

```
    else if (buf[i] >= 'a' && buf[i] <= 'z')
        count[buf[i]-'a'+ 26]++;// 出现小写字母
    for(i = 0; i < 52; i++)
        if (count[i]) /* 未出现的字母不输出 */
           printf("%c(%d)\t", i < 26 ? i+'A' :
              i-26+'a', count[i]);
    printf("\n\n");
}
```

【第二版例4.19】读入若干字符行，以空行(即只键入回车符的行)结束，输出其中最长的行。

【解题思路】

程序的主要工作是一个循环，每次循环输入一个字符行，求出输入字符的长度，接着判输入字符行是否是空行，空行是长度为0的行。如果是空行，则结束循环；如果不是空行，与早先求得的最长行比较，如果比早先求得的最长行更长，就用当前行更新已求得的最长行。直至循环结束，程序输出最长行。为了程序处理方便，最长行的信息以最长行内容和最长行的长度两项信息比较适宜，便于直接比较最长行的长度。

```
#include <stdio.h>
#include <string.h>
#define MAXLINE 256
int main()
{ int  len, max;   char line [MAXLINE], save [MAXLINE];
        max = 0;  for (; ;) { printf("Enter a line(return to finish).\n");
        gets(line);
        if ((len = strlen(line)) == 0) /*求当前行长度，并判是否为空行*/
            break; /*如果是空行，跳出循环*/
        if (len > max) {/*当前行是更长的行*/
            max = len; /*保存行的长度*/
            strcpy(save, line);
            }
        }
     if (max > 0) /*如果有非空行读入过的话，输出找到的最长行*/
           printf("The max length of line is:\n%s\n", save);
   return 0;
  }
```

【例4.17】输入字符串。统计其中单词个数。约定单词由英文字母组成，其他字符只是用来分隔单词。

例如：输入：

wer yyyuiu jjkjkj fdhj

There are 4 words in the line.

例如：输入：

abc bbc qwe r 123 ytu t78tyu tyu

There are 8 words in the line.

【解题思路】

先将一行字符读到数组中，然后顺序考察字符。对程序来说，数组中的字符只有两类：

单词的分隔符和英文字母。

程序在逐个访问字符的过程中，处于两种不同的状态：正在处理一段连续的英文字母字符和正在处理一段连续的其它字符。不妨记前者为“单词中”，后者为“不在单词中”。识别程序对当前考察的字符也分成两类：是英文字母字符和非英文字母字符。两个状态和两类字符的组合有四种情况。（11，10，01，00）

1、如果当前状态正在单词中，当前字符又是英文字母，则继续保持原状态。对于程序来说，没有额外要做的工作。(11)

2、如果当前状态正在单词中，当前字符不是英文字母，则程序状态变成不在单词中。(10)

3、如果当前状态不在单词中，当前字符又是英文字母，则程序状态变成在单词中。为了统计单词个数，这时单词计数器应增1。(01)

4、如果当前状态不在单词中，当前字符又不是英文字母。(00)

```
{ 设置单词计数器为0，当前状态为不在单词中等初值；
    输入一行字符（如用gets()函数）；
    顺序扫视输入的字符行（如用for语句）{ 判定当前字符是否是字母；
    if (当前状态在单词中) {
    if (当前字符不是字母) 设置当前状态为不在单词中；/* 若当前字符还是字母，应掠过当前字符，所以程序不作任何处理 */
    }
    else /* 当前状态为不在单词中，而当前字符又不是字母，应掠过当前字符；如果当前字符是字母，则一个新的单词开始 */
      if (当前字符是字母) { /* 一个新的单词开始 */
          设置当前状态为在单词中；
          单词计数器增 1 ；
        }
    }
  }
```

```
#include <stdio.h>
void  main()
{  char c, line[120];   int  i,   words, /* 单词计数器 */
   inword, /* 当前状态在单词中的标志变量 */
   letter; /* 当前字符是字母的标志变量 */
   words = 0;   inword = 0; /* 预置当前状态不在单词中的标志 */
   printf("Input a line.\n");  gets(line);//输入字符串
   for(i = 0; line[i]; i++) {  c = line[i];
        letter = ((c>='a' && c<='z') || (c>='A' && c<='Z'));
        if (inword) { if (!letter)  inword = 0;   }
       else if (letter) {  inword = 1;  words++; }
       }
   printf("There are %d words in the line.\n\n",words);
}
```

小结

- 数组的基本概念
- 一维数组

  定义、元素引用、初始化
- 多维数组

  二维、三维及多维定义、元素引用、初始化
- 字符串处理技术