7 数组

所谓数组，就是若干相同类型数据的有序集合。一般把这些相同类型的数据的变量采用一个名字命名，然后用有序的编号区分这些变量，这个名字即为数组名，编号为数组的下标。组成数组的各个变量称为数组的分量，或称元素，有时也称下标变量。

数组属于构造数据类型。利用数组名与下标，可以实现对任意数组元素的随机访问。

根据数组下标的维数，可以将数组分为一维、二维和多维数组。使用数组前需要对数组进行定义。

7.0一维数组

7.0.0 一维数组的定义

一维数组的定义方式为：

```
类型说明符 数组名[常量表达式], ...;
```

其中，类型说明符是任意一种基本数据类型或构造类型。数组名是用户定义的数组标识符，方括号中的常量表达式表示数据元素的个数，也称数组的长度。例如：

```
int a[10];          // 表示整型数组名为a，此数组有10个元素
float b[5], c[10];  // 表示两个浮点型数组，b有5个元素，c有10 个元素
```

对于数组的定义，要注意：

- 数组的类型就是数据元素的元素类型。对于同一个数组，所以元素的类型都是相同的。
- 数组名的书写规则与变量名相同，应符合标识符的书写规则。
- 数组名不能与其他变量名相同，例如：

  ```
  int main() {int a; float a[5]; ... return 0;}
  ```

  是错误的，在编译时会报错。
- 数组内元素个数为常量表达式的值，如 a[5] 有 5 个元素。但数组下标是从 0 开始计算的，所以，这五个元素分别为 a[0]、a[1]、a[2]、a[3]、a[4] 。不能使用数组元素 a[5] 。
- 允许在同一类型说明中，说明多个数组和多个变量。例如：

  ```
  int a, b, c[10];
  ```
- 对于采用 C87 标准的C语言，不能在定义数组时使用变量来表示元素的个数，也就是说**不能动态定义数组**，即数组的大小不依赖于程序运行过程中变量的值，只能是符号常量或常量表达式。例如：

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

  是不支持的。但在 C99 标准后，是支持可变长数组的，因此，对于用 C99 标准的C语言是支持上述代码的。

数组元素在逻辑上是连续的，在物理（内存）上也是连续排列的，数组元素在一片内存中按下标从小到大连续排列，每一个数组元素占用相同的字节数。定义好数组后，这一片连续的内存就分配给了这个数组，即使数组内没有初始数据，这一片内存也不会在分配给别的变量。

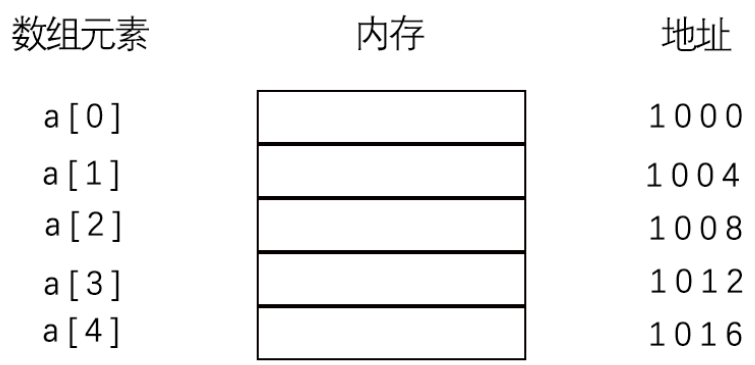

图 7-1 一维数组 a[5] 的存储结构图

由于一维数组是顺序存储在一片连续内存中的，数组名代表数组在内存中的起始地址，每一个数组元素占用的字节数相同，所以可以采用下面的公示来计算每个数组元素的内存地址，并实现对数组元素的随机存取。

```
数组元素地址 = 数组起始地址 + 元素下标*sizeof(数组类型)
```

7.0.1 一维数组的初始化

在 C 语言中，可以用赋值语句或输入语句使数组元素获得值，但是这样会占用一定的运行时间。若采用初始化赋值的方式给数组赋值，即在编译阶段使数组得到初始值，这样减少了运行时间，提高了效率。

对数组元素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“{}”中的各个数据值即为各元素的初值，各值之间用逗号隔开。例如：

```
int a[5] = {1, 2, 3, 4, 5};
```

相当于 a[0]=1, a[1]=2, a[2]=3, a[3]=4,a[4]=5 。

C语言中对数组的初始化还应注意以下几点：

- 只能按顺序给数组元素逐个进行赋值，不能给数组整体赋一个初始值。例如，给有5个元素的整型数组a全部赋值1，只能写为：

  ```
  int a[5] = {1, 1, 1, 1, 1};
  ```

  而不能写为：

  ```
  int a[5] = 1;
  ```

- 可以只给数组的部分元素赋值。当“{}”中值的个数少于数组元素个数时，只能按顺序给数组前面部分元素赋值，后面没指定赋值的元素自动赋初值 0 。例如：

  ```
  int a[5] = {2, 3};
  ```

  相当于 a[0]=2, a[1]=3, a[2]=0, a[3]=0, a[4]=0 。

- 如果给数组的全部元素都赋了初值，则在数组定义时可以不给出数组元素的个数。即数组元素个数由所给定的初值个数决定。例如：

  ```
  int b[] = {1, 2, 3, 4, 5, 6};
  ```

  等价于：

  ```
  int b[6] = {1, 2, 3, 4, 5, 6};
  ```

- 如果在定义数组时，不给数组进行初始化，则数组的每个元素均为不确定的值。例如：

  ```
  int a[5], b[5]={};
  ```

  相当于数组 a 中的五个值不确定，没有意义，数组 b 中五个元素都赋初值为0 。

- ```
  int a[5] = {[0]=1, [2]=3};
  ```

  相当于 a[0]=1, a[1]=0, a[2]=3, a[3]=0, a[4]=0 。

7.0.2 数组的使用

定义完毕的数组就可以使用了。数组元素是组成数组的基本单元，每一个元素都可以看做是一个变量。在C语言中只能逐个地引用数组元素而不能一次引用整个数组。如果要引用整个数组，可以利用一个循环来引用所以的元素。对于每一个数组元素的使用都采用如下的形式引用：

```
数组名[下标]
```

其中，下标可以是整型常量或整型表达式。下标表示元素在数组中的序号。

n 个元素的数组的下标范围为 0-(n-1) 。所以在使用下标的时候要注意不要下标越界。但其实有些编译器不会提示下标越界，但如果使用了越界的下标，就会覆盖其他内存中的数据，产生数据，所以不能使用越界的下标。

```c
#include <stdio.h>
// 数组的定义、初始化与使用示例
int main()
{
    int i, a[5], b[5]={1, 2, 3, 4, 5};

    for (i=0;i<5;i++) {printf("%d\t", a[i]);}
    printf("\n");
    for (i=0;i<5;i++) {a[i] = b[i] + 2;}
    for (i=0;i<5;i++) {printf("%d\t", b[i]);}
    /* 运行结果，先打印的都为无意义的随机值
    6356884 4200862 4200768 54      8
    1       2       3       4       5
    */
    return 0;
}
```

```c
#include <stdio.h>
// 数组的正向、逆向输出
int main()
{
    int i, a[5]={1, 3, 5, 7, 9};
    for (i=0;i<5;i++) {printf("%d\t", a[i]);}
    printf("\n");
    for (i=5-1;i>=0;i--) {printf("%d\t", a[i]);}
    /* 运行结果
    1       3       5       7       9
    9       7       5       3       1
    */
    return 0;
}
```

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

```c
// 迭代求得斐波那契数列
#define N 12
int main()
{
    int i, a[N]={1, 2};
    for (i=2;i<N;i++) {
        a[i] = a[i-1] + a[i-2];
    }
    for (i=0;i<N;i++) {
        printf("%d\t", a[i]);
    }
    /* 运行结果
    1       2       3       5       8       13      21      34      55
89      144     233
    */
    return 0;
}
```

数组这种可以存储多个数据元素的数据结构，这样的结构利于对数据进行查找和排序，和后面的结构体链表，都是常用于数据结构与算法设计的数据类型。具体的查找排序算法在数据结构与算法中探讨。

7.1 二维数组

在实际问题中很多数据是二维或多维的，为此，C语言允许构造多维数组。多维数组元素有多个下标，用以标识它在数组中的位置。为此，了解了二维数组，就可以依次推得更高维度的数组。

7.1.0 二维数组的定义

二维数组的定义与一维数组类似，其一般形式为：

```
类型说明符 数组名[常量表达式1][常量表达式2];
```

其中，常量表达式1表示第一维下标的长度，常量表达式2表示第二维下标的长度。与一维数组下标的范围一样，都是从0开始编排，最大取值都是各自长度减1。二维数组的元素个数采用如下公式计算得到：

```
二维数组元素个数 = 常量表达式1 * 常量表达式2
```

例如：

```
int a[3][4];
```

这条语句定义了一个有3行4列共12个元素的整型二维数组a 。数组 a 的各个元素为：

```
a[0][0],a[0][1],a[0][2],a[0][3]
a[1][0],a[1][1],a[1][2],a[1][3]
a[2][0],a[2][1],a[2][2],a[2][3]
```

二维数组在形式上类似数学中的矩阵，一般将数组的第一个下标称为行下标，第二个下标称为列下标。虽然二维数组的下标在两个方向上变化，但数组在物理内存中还是按一维线性连续排列的。在C语言的数组中，按照行优先的规则存储数据，即优先邻接同一行的数据，存储完第一行，再存储第二行。所以，二维数组中每个元素的内存地址也可以由公式得到（设二维数组 a[M][N]，其任意元素为：a[i][j] (0<=i<=M-1，0<=j<=N-1)）：

```
数组元素a[i][j]的地址 = 数组a的起始地址 + (M*i+j)*sizeof(数组类型)
```

7.1.1 二维数组的初始化

二维数组及多维数组的初始化和一维数组类似，初始化形式为：

```
类型说明符 数组名[下标1][下标2] = {常量1, 常量2, 常量, ... , 常量n}
```

例如：

```
int a[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

- 二维整型数组 a 的初始化还可以分行进行，即

  ```
  int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
  ```

- 如果对全部元素赋初值（即提供全部初始数据），则定义数组时对第一维的长度可以不指定，但第二维的长度不能省略。例如：

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

  等价于

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

  系统会根据数据总个数分配存储空间，一共9个数据，每行3列，当然可以确定为3行。

- 可以只对部分元素赋初值，未赋值的元素自动取0值。例如：

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

  数组a初始化后的数组元素如下：$\begin{vmatrix} 1 & 0 & 0 \\ 2 & 4 & 0 \\ 3 & 5 & 6 \\ 0 & 0 & 0 \end{vmatrix}$，

  但如果将上面的语句改为：

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

  则数组a各元素为：$\begin{vmatrix} 1 & 0 & 0 \\ 2 & 4 & 0 \\ 3 & 5 & 6 \end{vmatrix}$

7.1.2 二维数组的使用

二维数组的元素的表示形式为：

```
数组名[下标1][下标2]
```

如 a[2][3] 。下标可以是整型表达式，也可以是符号常量。同一维数组一样，下标也不能越界。但同时，在二维数组中，a[1] ，等属于变量名，不能直接引用得到数据。

```c
#include <stdio.h>
#include <stdlib.h>
// 二维数组的初始化，
int main()
{
    int i, j, a[3][2]={1, 2, 3, 4, 5, 6};
    printf("Print in 2D:\n");
    for (i=0;i<3;i++) {
        for (j=0;j<2;j++) {
            printf("%d\t", a[i][j]);
        }
```

```
    }
    printf("\n");
    printf("Print in 1D:\n");
    for (j=0;j<6;j++) {
            printf("%d\t", a[0][j]);
    }
    printf("\n");
    /* 运行结果
    Print in 2D:
    1       2       3       4       5       6
    Print in 1D:
    1       2       3       4       5       6
    */ // 由于编译器不会检验下标是否越界，并且在内存中的排列邻接，所以可以把二维数组当成一维数组来进行调用，结果与调用二维数组结果相同
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
// 打印杨辉三角前9行
#define N 10
int main()
{
    int a[N][N], i, j;
    for (i=0;i<N;++i) {
        a[i][0] = 1;
        a[i][i] = 1;
    }
    for (i=2;i<N;++i) {
        for (j=1;j<i;++j) {
            a[i][j] = a[i-1][j] + a[i-1][j-1];
        }
    }
    for (i=0;i<N-1;++i) {
        for (j=0;j<=i;++j) {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    /* 运行结果
1
1       1
1       2       1
1       3       3       1
1       4       6       4       1
1       5       10      10      5       1
1       6       15      20      15      6       1
1       7       21      35      35      21      7       1
1       8       28      56      70      56      28      8       1
    */
    return 0;
}
```

7.2 字符数组

用来存放字符型数据的数组称为字符数组。字符数组中的一个元素存放一个字符。

7.2.0 字符型数组的定义与初始化

与数值型数组的定义相同。例如：

```
char a[10];
```

定义了一个有10个元素的字符数组 c 。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。字符型数组也可允许在定义时作初始化赋值。例如：

```
char a[6] = {'s','h','u',' ','z','u'};
```

把9个字符分别赋值给字符型数组 a 的6个元素；a[0]~a[5] 。同样，如果初始值个数小于数组长度，则直将这些字符赋值给数组前面的那些元素，其余元素自动赋为空字符（即'\0'）。

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 7-2 字符数组的存储示意图

在初始化字符数组时，也可以省去长度说明，系统会根据初始值个数确定数组长度。

7.2.1 字符型数组和字符串的关系

C语言中，通常用一个字符数组来存放一个字符串。由字符数组的初始化可知，如果字符串的长度小于字符数组的长度时，多余的字符数组元素都会被赋值为空字符。事实上，人们关心的是有效字符串的长度而不是数组的长度。为了测定字符串的实际长度，C语言中规定了一个"字符串结束标志"，以字符'\0'表示。如果有一个字符串，其中第10个字符为'\0'，则此字符串的有效字符数为9.也就是说，在遇到字符'\0'时，表示字符串结束，由它前面的字符组成字符串。

前面的字符串常量也说明，字符串也总是以'\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。有了结束符后，就不必再用字符数组的长度来判断字符串的长度了。当然，在定义字符数组时应估计字符串长度，以确保数组长度始终大于字符串的实际长度。

C语言运行使用字符串的方式对数组作初始化赋值。例如：

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

可以写为

```
char c[] = {"bian cheng"};
```

或更简略的写为

```
char c[]= "bian cheng";
```

用字符串方式赋值比用字符逐个赋值要多占一个字节，用于存放字符串结束标志'\0' 。系统针对以上后两种初始化时，将自动在其后面追加一个结束符。

但字符数组并不要求最后一个字符是否为'\0'，如果没有结束符，它就仅仅是一个字符数组，而不是一个字符串。

7.2.2 字符数组的输入输出

字符数组的输入输出可以采用如下两种方法：

- 逐个字符输入输出：采用格式“%c”输入或输出一个字符。例如：

```
char c[10] = {'b','i','a','n',' ','c','h','e','n','g'};
for (int i=0;i<10;i++) {
    printf("%c", c[i]);
}
```

- 将整个字符串一次输入或输出：采用“%s”格式符。例如：

```
char c[] = "bian cheng";
printf("%s",c);
```

这也是仅有的可以使用数组名直接输出整个数组的方式。

```
#include <stdio.h>
#include <stdlib.h>
// 字符数组和字符串的输出
int main()
{
    int i;
    char c[11] = {'b','i','a','n',' ','c','h','e','n','g','.'};
    char d[] = "bian cheng.";
    for (i=0;i<11;i++) {
        printf("%c",c[i]);
    }
    printf("\n");
    printf("%s\n", c);
    printf("%s\n", d);
    /* 运行结果
bian cheng.
bian cheng.•
bian cheng.
    */ // 第二行的输出因为结束符的编码而产生一个乱码
    return 0;
}
```

采用“%s”格式符输出字符串时，还需要注意以下几点：

- 输出字符不包括空字符'\0'。
- printf 函数中的输出项是字符数组名，而不是数组元素名。所以如下形式是错误的：

  ```
  printf(“%s”, c[0]);
  ```

- 如果数组长度大于字符串实际长度，也只能输出第一个空字符之前的字符。

  可以使用scanf函数接收从键盘输入的一个字符。例如：

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

其中输入项c是字符数组名，它必须在之前已经被定义了。且输入字符串长度应该小于字符数组的长度。还要注意：

- scanf 函数中的输入项是字符数组名，不必再加地址符 &，因为在C语言中数组名代表该数组的起始地址。
- 当使用 scanf 函数时，字符串中如果含有空格的话，将以空格作为串的结束符。如果后面还有字符，将被忽略而不能正常输出。如果想要读取一整行，可以使用 gets 函数或者使用 %[^\n] 的格式

描述符。

7.2.3 字符串处理函数

C语言提供了丰富的字符串处理函数，大致可分为字符串的输入、输出、复制、合并、比较、转换等类型。使用这些函数可大大减轻编程的负担。用于输入输出的字符串函数，在使用前应包含头文件“stdio.h”，使用其他字符串函数则应包含头文件“string.h”。

7.2.3.0 输入字符串：gets()函数

1. 调用方式：gets(字符数组)
2. 函数功能：从标准输入设备（stdin）——键盘上，读取1个字符串（可以包含空格），并将其存储到字符数组中去，自动在末尾添加字符串结束标识符‘\0’。
3. 使用说明：
    1. gets() 读取的字符串，其长度没有限制，使用者要保证字符数组有足够大的空间，存放输入的字符串。
    2. 该函数输入的字符串中允许包含空格，遇到空格并不停止输入，而是以回车作为输入结束。

7.2.3.1 输出字符串：puts()函数

1. 调用方式：puts(字符数组)
2. 函数功能：把字符数组中所存放的字符串，输出到标准输出设备中去，并用‘\n’取代字符串的结束标志‘\0’。所以使用 puts() 函数输出字符串时，不要求另加换行符。
3. 使用说明：
    1. 字符串中允许包含转义字符，输出时产生一个控制操作。
    2. 该函数一次只能输出一个字符串，而 printf 函数也可以用来输出字符串，且一次能输出多个。当需要按一定格式输出时输出时，通常使用 printf 函数。

```c
#include <stdio.h>
// gets() 函数与 puts() 函数示例
int main()
{
    char st[30];
    printf("Input a string:\n");
    gets(st);
    printf("Output the string:\n");
    puts(st);
    return 0;
}
```

7.2.3.2 拷贝字符串：strcpy()函数

1. 调用方式：strcpy(字符数组，字符串)。其中字符串可以是字符串常量，也可以是字符数组。
2. 函数功能：将字符串完整地复制到字符数组中，字符数组中原有内容被覆盖。
3. 使用说明：
    1. 字符数组必须定义得足够大，以便容纳复制过来的字符串，即字符数组的长度不小于字符串的长度。复制时，连同结束标志‘\0’一起复制。
    2. 不能用赋值运算符“=”将一个字符串直接赋值给一个字符数组（除非在定义变量时初始化），只能用 strcpy() 函数来处理。例如，如下语句是错误的：

```c
char c[20];
c = "Hello!";    // 不能这样将字符串赋值给字符数组
```

如果要用赋值语句，只能一个一个字符的赋值给字符数组的元素。

3. 还可以用 **strncpy()** 函数将字符串中前若干个字符赋制到字符数组中去（不是strcpy函数）。例如：

```
strncpy(str,"China",2);
```

7.2.3.3 连接字符串：strcat()函数

1. 调用方式：strcpy(字符数组，字符串)。
2. 函数功能：将字符串连接到字符数组中的字符串尾端。字符数组中原来的结束标志，被字符串的第一个字符覆盖。
3. 使用说明：
    1. 由于没有越界检查，所以要保证字符数组定义的足够大，以便容纳连接后的目标字符串；否则，会因长度不够而产生问题。
    2. 连接前两个字符串都有结束标志'\0'，连接后字符数组中存储的字符串的结束标志'\0'被字符串的第一个字符覆盖，只在目标串的最后保留结束标志符'\0'。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// strcpy() 函数与 strcat() 函数示例
int main()
{
    char st_1[30] = "My name is ", st_2[10], st_3[10];
    printf("Input your name:\n");
    gets(st_2);
    strcat(st_1, st_2); strcat(st_1, ".");
    puts(st_1);
    strcpy(st_1,"Hello! "); strcpy(st_3, st_2); strcat(st_1, st_3);
strcat(st_1, ".");
    puts(st_1);
    return 0;
}
```

7.2.3.4 字符串比较：strcmp()函数

1. 调用方式：strcmp(字符串1,字符串2)。其中字符串可以是字符串常量，也可以是字符数组。
2. 函数功能：比较两个字符串的大小。如果：
    1. 字符串1等于字符串2，函数返回值等于0；
    2. 字符串1小于字符串2，函数返回值为负整数；
    3. 字符串1等于字符串2，函数返回值为正整数。
3. 使用说明：
    1. 字符串比较规则为：对两个字符串从左到右逐个字符相比较（按ASCII码值大小比较），直到出现第一个不同的字符或遇到'\0'为止。如果全部字符相同，则认为相等；若出现不同的字符，则以第一个不相同的字符的比较结果为准；如果一个字符串是另一个字符串从头开始的子串，则母串为大。
    2. 不能使用关系运算符"=="来比较两个字符串，只能用 strcmp() 函数来处理。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// strcmp 函数示例
```

```c
int main()
{
    char s1[20]="Bian cheng", s2[20]="jiao cheng", s3[20];
    strcpy(s3, s1);
    printf("s1 = %s\ns2 = %s\ns3 = %s\n", s1, s2, s3);
    printf("strcmp(s1, s2) = %d\n", strcmp(s1, s2));
    printf("strcmp(s2, s1) = %d\n", strcmp(s2, s1));
    printf("strcmp(s1, s3) = %d\n", strcmp(s1, s3));
    /* 运行结果
s1 = Bian cheng
s2 = jiao cheng
s3 = Bian cheng
strcmp(s1, s2) = -1
strcmp(s2, s1) = 1
strcmp(s1, s3) = 0
    */ // 1和-1不是绝对的，视编译器而定
    return 0;
}
```

7.2.3.5 求字符串长度：strlen()函数

1. 调用方式：strcmp(字符串)。
2. 函数功能：求字符串（或字符常量）的实际长度，不包括结束标志符'\0'。
3. 使用说明：该函数返回的是一个无符号正数。所以要注意无符号运算过程中与有符号运算发区别。例如：

```
if (strlen(s1) >= strlen(s2)) ...
if (strlen(s1) - strlen(s2) >= 0) ...
```

   看似两者的作用是一样的，但二式无符号的减法得到的结果永远是非负的，所以不可能小于0，条件即为永真。

7.2.3.6 将字符串中大写字母转换为小写：strlwr()函数

1. 调用方式：strlwr(字符串)。
2. 函数功能：将字符串中的大写字母转换成小写，其他字符不转换。

7.2.3.7 将字符串中小写字母转换为大写：strupr()函数

1. 调用方式：strupr(字符串)。
2. 函数功能：将字符串中的小写字母转换成大写，其他字符不转换。

7.2.4 字符型数组与字符串的简单应用

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// 实现简单的密码检测
int main()
{
    char pass[80];
    char pwd[80] = "password";
    int i = 0;
    while (1) {
        printf("请输入密码：\n");
        gets(pass);
```

```
        if (strcmp(pass, qwd) != 0) {
            printf("密码错误，按任意键继续...");
        } else {
            printf("OK!\n");
            break;
        }
        i++;
        if (i==3) {
            printf("\n已经输入三次错误的密码！\n")
        }
    }
    return 0;
}
```