

数组可以看作是相同类型变量的集合,数组元素在内存中连续存储,数组名代表数组的首地址。数组的大小必须在定义时确定,程序中不可更改。

5.1 一 维 数 组

5.1.1 一维数组的定义和初始化

一维数组的定义格式:

```
[static]数据类型 数组名[数组大小] [ = {初值列表}];
```

例 1:

```
float cj[10];
```

数组名为 cj,有 10 个元素,分别为 cj[0]~cj[9],相当于同时定义了 10 个单精度浮点型变量,变量未赋初值。

例 2:

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

或:

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

定义 5 个元素的整型数组 a,初始化 a[0]=1, a[1]= 2, a[2]=3, a[3]= 4, a[4]=5。如果定义数组时省略数组的大小,则以初始化列表中初值的个数作为数组大小。

例 3:

```
int b[10] = {1,2,3,4,5};
```

定义 10 个元素的整型数组 b,初始化 b[0]~b[4]为 1~5,没有初值的 5 个元素 b[5]~b[9]自动赋以随机数,概率率为 0。

例 4:

```
static int b[10] = {1,2,3,4,5};
```

定义 static 静态数组,没有初值的整型元素自动赋初值 0,字符型则自动赋初值 NULL,不再是赋值随机数。

例 5:

```
char x[5] = {'h','e','l','l','o'};
```

定义 5 个元素的字符型数组 x, 字符型加单引号引一个字符, 数值型直接写数字, 不加引号。

说明:

- (1) 定义中, 数组大小必须是常量或常量表达式, 不可以是变量。
- (2) 数组在内存中连续存放, 占用的空间是每一个元素占用空间的和。

5.1.2 一维数组的使用

引用一维数组元素的方式:

数组名[元素编号]

说明:

- (1) 数组下标是从 0 开始的整数, 不是从 1 开始。
- (2) C 语言对数组下标不做越界检查, 使用时自行管理。如定义 `int a[5]`, 则数组元素最大是 `a[4]`, `a[5]` 就越界了。
- (3) 数组不能整体输入和输出, 只能对其数组元素进行输入和输出(字符型数组字符串除外)。
- (4) 使用数组元素时, 下标不仅可以是整型常量或常量表达式, 还可以是整型变量或变量表达式, 只要表达式计算结果是整数, 并且不越界即可。例如: `a[2 > 3]` 相当于 `a[0]`。
- (5) 数组元素和普通变量一样使用。
- (6) 给数组元素赋值和输出, 经常用到 for 循环。
- (7) 数组元素在内存中连续存储, 用 `sizeof` 运算符可以计算一个数组元素或整个数组所占内存空间大小。表达式 `sizeof(数组名)/sizeof(数组元素类型|任意数组元素名)` 可以计算出数组的长度。例如, 将数组 b 的全部元素置为 0, 可以写成:

```
for(i = 0; i < sizeof(b) / sizeof(b[0]); i++)  
    b[i] = 0;
```

5.2 二维数组

存一门课的成绩用一维数组, 存多门课的成绩则需要用二维数组。一维数组类似于电影院的一排座位, 二维数组类似于电影院的多排座位。

5.2.1 二维数组的定义

二维数组的定义格式:

[static]数据类型 数组名[一维数组个数][一维数组长度]

相当于:

[static]数据类型 数组名[矩阵行数][矩阵列数]

例如：

```
int m[3][5];
```

定义了二维数组 `m`，相当于定义了 3 行 5 列共 15 个整型变量，效果如图 5-1 所示，二维数组行、列下标都是从 0 开始。内存中没有二维的概念，存放顺序是先行后列，先存储第一行 `m[0][0]`、`m[0][1]`、 $\cdots$ ，最后存储 `m[2][4]`，数组元素连续存放。

<code>m[0][0]</code>	<code>m[0][1]</code>	<code>m[0][2]</code>	<code>m[0][3]</code>	<code>m[0][4]</code>
<code>m[1][0]</code>	<code>m[1][1]</code>	<code>m[1][2]</code>	<code>m[1][3]</code>	<code>m[1][4]</code>
<code>m[2][0]</code>	<code>m[2][1]</code>	<code>m[2][2]</code>	<code>m[2][3]</code>	<code>m[2][4]</code>

图 5-1 二维数组 `m[3][5]` 示意

5.2.2 二维数组的初始化

二维数组初始化格式：

```
[static]数据类型 数组名[矩阵行数][矩阵列数][ = {初值列表}];
```

例 1：

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

这是分行赋值法，相当于嵌套了多行一维数组，每一行的初值用第二层大括号括起来。赋值结果：`m[0][0]=1`，`m[0][1]=2`，`m[1][0]=3`，`m[1][1]=4`，`m[2][0]=5`，`m[2][1]=6`。

例 2：

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

这是顺序赋值法，赋值结果与例 1 相同。

例 3：

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

定义二维数组时，如果给出了初值列表，则可以省略行下标，但不可以省略列下标。赋值结果与例 1 相同，根据初值数量决定行数。

例 4：

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

如果初始化式子没有足够初值，那么其他元素自动赋以随机数，大概率为 0。赋值结果如图 5-2 所示。

<code>m[0][0] = 1</code>	<code>m[0][1] = 2</code>	<code>m[0][2] = 3</code>	<code>m[0][3] = 0</code>
<code>m[1][0] = 2</code>	<code>m[1][1] = 3</code>	<code>m[1][2] = 4</code>	<code>m[1][3] = 5</code>
<code>m[2][0] = 0</code>	<code>m[2][1] = 0</code>	<code>m[2][2] = 0</code>	<code>m[2][3] = 0</code>

图 5-2 二维数组 `m[3][4]` 赋值示意

例 5:

```
static int m[3][4] = {1, 2};
```

static 静态整型数组,所有元素自动赋初值 0,赋值结果: $m[0][0]=1, m[0][1]=2$, 其他元素 $m[0][2], m[1][0] \sim m[2][3]$ 都是 0。

例 6:

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

只给第 1 和第 3 行前两个元素赋初值,第 2 行未赋值。

5.2.3 二维数组的使用

引用二维数组元素的方式:

数组名[行元素编号][列元素编号]

说明:

(1) 数组行、列下标是从 0 开始的整数,数组 `int m[3][4]` 最小下标元素是 $m[0][0]$,最大下标元素是 $m[2][3]$ 。

(2) 数组经常用到 for 循环,一维数组用一层 for 循环,二维数组用两层 for 循环。

(3) 二维数组元素 $m[2][3]$ 不可以写为 $m[2,3]$,否则会按照逗号表达式的规则等价于 $m[3]$ 。

5.3 字符型数组

字符型数组可以是一维数组,也可以是二维数组,特殊性在于 C 语言没有字符串类型,字符串存在字符型数组中,需要预留一字节存储 `'\0'` (空字符 NULL),标识字符串结束。

字符型数组存字符串时可以整体输入和输出,一维数组可以存一个字符串,二维数组可以存多个字符串,数组名代表字符串的首地址。

5.3.1 字符型数组的定义和初始化

一维数组:

```
char 数组名[数组大小] [ = {初值列表}];
```

二维数组:

```
char 数组名[矩阵行数][矩阵列数][ = {初值列表}];
```

例 1:

```
char word[6] = "Hello";
```

为字符型数组初始化字符串时可以省略大括号,直接用双引号括起来字符串。与 `char word[6] = {"Hello"};` 效果一样。

例 2:

```
char word[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

以一般数组的形式初始化字符串,后面加上字符'\0'就表示存入的是字符串。

例 3:

```
char word[] = "Hello";
```

定义字符型数组时没有指定大小,则根据初值确定大小,初值中有 5 个字符再加上字符串结束标志'\0',则数组 word 长度为 6,同例 1、例 2 效果一样。

例 4:

```
char word[6] = {'H', 'e', 'l', 'l', 'o'};
```

此数组存入 5 个字符,后面没加字符'\0',不是字符串,不能整体输入和输出,在程序中需要单个元素使用。

例 5:

```
char word[5] = "Hello";
```

越界异常,因为无处存储字符'\0'。

例 6:

```
char a[3][10] = {"china", {" is "}, {"gread!"}};
```

定义二维字符型数组,并赋初值。3 行 10 列的二维字符型数组可以存 3 个字符串,每个字符串最多 9 个字符,留一字节存 NULL 字符。

说明:

(1) 定义字符型数组存字符串时不要忘记多留一字节存 NULL 字符。

(2) 二维数组也可以理解为多个一维数组,可以只用行下标表示,如 a[1]表示第 2 行的一维数组,指向字符串" is "的首地址。

5.3.2 字符型数组的使用

字符型数组存入字符串时,除了可以用"%c"格式符逐个字符输入输出之外,还可以用"%s"格式符整体输入和输出,而且还有专门的字符串输入输出函数 gets()和 puts()。

scanf()函数与 gets()函数读字符串区别: scanf()函数读字符串,遇到空格终止,无法读入空格和制表符。gets()函数可以读入带空格的字符串,遇到回车符才终止。

printf()函数与 puts()函数输出字符串的区别: printf()函数输出不自动换行,需要在格式符中加"\n"换行,puts()函数输出字符串后自动换行。

5.3.3 字符串处理函数

字符串与其他类型数据不同,不可以用双等号==比较大小,也不可以用一个等号=赋值,必须用字符串处理函数完成相关操作。字符串处理函数在 string.h 头文件中定义。

1. strcpy(): 复制字符串函数

例如:

```
strcpy(name, "Apple");
```

作用：将字符串"Apple"赋值到字符型数组 name 中,并在串尾加上结束标志。注意数组 name 大小至少为 6,可以存 5 个字符和 1 个字符串结束标志。

2. strcat(): 拼接两个字符串函数

例如：

```
char name[20] = "hello!";  
strcat(name, "Apple");
```

作用：将字符串"Apple"连接到字符型数组 name 后面,并去掉 name 中原有的结束标志,在串尾加上结束标志。数组 name 中的值变为"hello! Apple"。数组 name 长度必须足够存下拼接后的字符串和字符串结束标志。

3. strcmp(): 比较两个字符串的大小

语法形式：

```
strcmp(字符串 1, 字符串 2);
```

如果字符串 1 与字符串 2 相同,则函数值为 0;如果字符串 1 > 字符串 2,则函数值为正整数;如果字符串 1 < 字符串 2,则函数值为负整数。

4. strlen(): 计算字符串长度函数

例如：

```
char name[20] = "hello!";  
printf("%d", strlen(name));    //结果为 6, 不计算\0
```

5. strlwr()、strupr(): 大小写转换函数

例如：

```
char a[] = "AbCd";  
printf("小写: %s\n", strlwr(a)); //结果为: abcd  
printf("大写: %s\n", strupr(a)); //结果为: ABCD
```

5.4 程序示例

【例 5-1】 编程输入 5 个整数作为成绩存入数组,计算平均分,并输出大于平均分的成绩。

参考代码：

```
#include <stdio.h>  
#define N 5                //符号常量,用于定于数组长度  
int main( )  
{  
    int a[N], i, sum = 0;    //i 为循环变量,sum 记录成绩总和  
    float avg = 0.0;        //记录平均分  
    printf("请输入五个整数成绩:\n");
```

```

for(i = 0; i < N; i++)          //第一次循环,输入 N 个数
    scanf("%d", &a[i]);
for(i = 0; i < N; i++)          //第二次循环,计算 N 个数之和
    sum = sum + a[i];
avg = sum/N;                    //计算平均分
printf("平均分: %.2f\n", avg); //输出平均分
for(i = 0; i < N; i++)          //第三次循环,找高于平均分的
{ if (a[i] > avg)                //判断大于平均分则输出
    printf("%d 大于平均分\n", a[i]); }
}

```

运行效果:

请输入五个整数成绩:

78 67 32 90 45

平均分: 62.00

78 大于平均分

67 大于平均分

90 大于平均分

【例 5-2】 编写程序,输入三名学生的两门课的成绩,并输出。要求按课程输入,按学生输出。

参考代码:

```

#include <stdio.h>
#define M 2                                //符号常量,用于行下标,表示课程数
#define N 3                                //符号常量,用于列下标,表示学生数
int main( )
{ int s[M][N], i, j;
  for(i = 0; i < M; i++)                    //M 表示课程数,按课程输入
  { printf("输入第 %d 门课成绩:\n", i + 1);
    for(j = 0; j < N; j++)                  //N 表示学生数
    { printf("第 %d 名学生:", j + 1);
      scanf("%d", &s[i][j]);               //读入成绩
    }
    printf("\n 输出学生成绩:\n");
    for(j = 0; j < N; j++)                  //N 表示学生数,按学生输出
    { printf("第 %d 名学生成绩:", j + 1);
      for(i = 0; i < M; i++)                //M 表示课程数
      { printf("%d ", s[i][j]);             //输出成绩
      }
      printf("\n");
    }
  }
}

```

运行效果:

输入第 1 门课成绩:

第 1 名学生: 79

第 2 名学生: 78
第 3 名学生: 77
输入第 2 门课成绩:
第 1 名学生: 95
第 2 名学生: 97
第 3 名学生: 96

输出学生成绩:
第 1 名学生成绩: 79 95
第 2 名学生成绩: 78 97
第 3 名学生成绩: 77 96

程序说明:

(1) 存一门课程成绩用一维数组,存三名学生的两门课成绩需要用二维数组,二维数组是三行二列或者二行三列都可以,只要分清行和列中,谁代表课程数,谁代表学生数即可。

(2) 要求按课程输入成绩,所以输入成绩时以课程数作外层循环,学生数作内层循环。输出成绩时则相反。

【例 5-3】 编程输入一个以回车符为结束标志的字符串(少于 80 个字符),统计其中数字字符的个数。

参考代码:

```
#include <stdio.h>
int main( )
{
    int count = 0, i;                                //count 计数数字字符,i 为循环变量
    char str[80];                                     //声明字符数组,存输入的字符串
    printf("请输入字符串:\n");
    gets(str);                                        //用 gets 读入字符串可以读入空格

    for(i = 0; str[i] != '\0'; i++)                  //数组下标从 0 开始,到字符串结束
    {
        if(str[i] >= '0' && str[i] <= '9')           //判断每个元素是否数字
            count++;                                  //是数字则累加 1
    }
    printf("数字有 %d 个\n", count);                 //输出统计结果
}
```

运行效果:

请输入字符串:
abc123 hello456 I love Chain
数字有 6 个

【例 5-4】 阅读程序,了解二维字符型数组的使用。程序功能: 输出二维字符型数组中的多个字符串,同时输出字符串长度。

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

```

#include <string.h>                                //加入头文件使用字符串函数
int main( )
{
    char m[3][8] = {"China","Japan","America"}; //定义二维字符型数组赋值三个字符串
    int len,i;                                    //len 存字符串长度,i 循环变量
    for(i = 0; i<3; i++)                          //三行循环三次
    {
        len = strlen(m[i]); //计算字符串长度,二维数组只用行下标表示每行的字符串
        printf(" %s, %d 个字符\n",m[i],len);      //输出字符串及其长度
    }
}

```

运行效果:

China,5 个字符
 Japan,5 个字符
 America,7 个字符

5.5 常见错误

表 5-1 中列举了对数组的几种常见错误写法。

表 5-1 对数组的常见错误写法

序号	常见错误写法	错误原因	正确写法
1	<pre>int n = 4; int a[n];</pre>	定义数组时,数组长度必须是常量,不可以是变量	<pre>int a[4]; 或者: #define N4 int a[N];</pre>
2	<pre>int b[5]; b = {10, 20, 30, 40, 50};</pre>	数组初始化时可用大括号整体赋值,在程序中必须单个元素赋值	<pre>int b[5] = {10,20,30,40,50}; 或: int b[5]; b[0] = 10; b[1] = 20; ...</pre>
3	<pre>int a[2], b[2]; a[0] = 1; a[1] = 2; b = a;</pre>	不能对数组整体输入输出, $b = a$; 语句错误,只能逐个元素输入输出(字符型数组存字符串除外)	<pre>int a[2], b[2]; a[0] = 1; a[1] = 2; for(i = 0; i < 2; i++) b[i] = a[i];</pre>
4	<pre>#define N 5 ... int arr[N]; for(i = 0; i <= N; i++) { arr[i] = i; }</pre>	数组越界异常,N 个元素数组的数组元素下标最大为 $N - 1$,不是 N	<pre>#define N 5 ... int arr[N]; for(i = 0; i < N; i++) { arr[i] = i; }</pre>

实验8 数组使用练习

一、实验目的与要求

1. 掌握一维数组的定义及使用方法。
2. 掌握字符数组的定义、初始化方法及元素的引用方法。
3. 掌握常见字符串处理函数的使用。

二、实验环境

Visual C++ 2010/Visual C++ 6.0。

三、实验内容

1. 编程定义三个同样大小的整型数组,为其中两个数组读入数值,然后计算两个数组中对应数值之和存入第三个数组,最后输出三个数组中的内容。



运行效果:

请输入第一个数组中5个数值:

第1个数: 12
第2个数: 13
第3个数: 14
第4个数: 15
第5个数: 16

请输入第二个数组中5个数值:

第1个数: 21
第2个数: 23
第3个数: 24
第4个数: 24
第5个数: 25

a 数组中内容: 12 13 14 15 16

b 数组中内容: 21 23 24 24 25

c 数组中内容: 33 36 38 39 41



2. 输入一个字符串,存在字符型数组中,再输入一个字符,在字符数组中查找该字符。若找到,输出该字符第一次出现的数组下标,否则输出-1。

运行效果:

input a string: hello how are you?	input a string: how are you?
input a char: o	input a char: x
下标: 4	下标: -1



3. 编程实现输入一个含空格的字符串,统计其中有多少个单词(空格分隔单词),输出单词个数和字符串长度。(注意:测试最后是空格和不是空格两种情况。)

运行效果:

请输入一个英文句子:	请输入一个英文句子:
hello how are you?	hello how are you?↵有空格
单词个数: 4	单词个数: 4

字符串长度: 18

字符串长度: 19

4. 理解冒泡排序算法,用冒泡排序对 10 个整数升序排序,先读入 10 个无序整数存入数组,对数组重新排序后输出数组中数值,输出结果 5 个一行显示。

拓展: 随机数生成 10 个 100 以内整数存入整型数组,按从小到大的顺序排序并输出(冒泡排序)。

