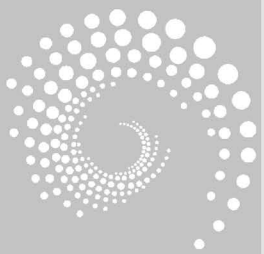
第5章

数　组

CHAPTER 5



内容导引

前面介绍了C语言中常用的基本数据类型——整型、实型和字符型，对于简单问题，使用这些基本数据类型所定义的变量是可以实现程序功能的，但是当处理的数据量很大或者构成较复杂时，使用基本数据类型变量就行不通了，它难以反映出所表示事物的复杂结构特点，也难以进行有效的数据处理。例如要计算某学校全年级学生某门课程的成绩，学生人数会很多，如果采用基本数据类型变量来处理，那就意味着需要定义很多个变量，既烦琐又不能反映出数据之间的内在联系。

为了处理和存储大量的复杂数据，需要学习派生数据类型。派生数据类型又称导出类型或构造类型，是由基本数据类型按照一定的规则组合或转变而来的数据类型。C语言中的派生数据类型有数组类型、结构体类型、共用体类型等。

数组是最基本的构造类型，在处理同类型的大批量数据时具有强大的功能。数组的实质是具有相同名称（称为数组名）不同序号（称为下标）的一批变量的集合，集合中的每个成员都相当于一个变量。因其下标可以是变量，因此具有书写简便，操作灵活等特点，应用非常广泛。本章将学习数组数据类型的使用，其他的派生数据类型将在后续课程中学习。

学习目标

- 理解数组的概念。
- 掌握数组的定义及引用。
- 理解一维数组与二维数组的关系。
- 能够使用数组进行程序设计。
- 掌握字符串处理函数的功能及应用。

5.1　一维数组及使用

内容概述

通过前面编写程序的过程我们可以感受到，计算机程序对数据的处理包括存储、操作都是通过变量来实现的。因为变量是用来操作数据的，因此有什么类型的数据，就有什么类型的变量。我们之前所学到的变量可称之为简单变量，一个变量只能够操作一个数据。对于需要处理大量数据的问题，比如一个班级全部考试成绩数据的处理，甚至全校学生成绩数据的处理，毫无关联的简单变量如 a、b、c、d 等是无法满足要求的。为此 C 语言提供了一种新的数据类型——数组。**数组是用来处理相同类型的批量数据的一种数据类型**，在 C 语言中将这种数据类型归纳为派生数据类型，它是基本数据类型的一种扩充和延伸。本节学习重点为：

- 数组的概念及分类。
- 一维数组的定义、初始化及引用。
- 利用一维数组编写程序的方法。

5.1.1　一维数组的概念及定义


在线视频

数组的使用源自大量数据处理中变量的命名及引用策略。对于一批同类型的数据处理，比如一个班级所有学生一门课程或者一名学生所有课程的成绩数据，用以前讲过的变量存储和操作就很不方便了，此时变量命名面临着以下挑战。

如果用 a，b，c，…，z 等变量表示这些学生的成绩数据，那么第 27 个以后的学生成绩用哪个变量表示？如果需要输出第 17 个学生的成绩，它存放在哪个变量里呢？在数据量多的应用场景下使用这样的变量，在其命名和数据的提取这两个方面都会遇到困难。

换个思路，如果采用字符开头后缀数字序号的方式给变量命名，这样数字序号对应学生的序号，如采用 a1，a2，a3，…依次表示学号为 1，2，3…的学生的成绩，这样的变量命名规则可以使变量与实体对象之间建立对应的次序关系，可解决多于 26 个数据后变量的命名问题，显然可用 a27 表示第 27 个学生的成绩数据。当需要操作第 30 个学生的成绩时，显而易见 a30 中存放的就是该学生的成绩。但如果需要对学生成绩求和或求平均分，则成绩相加的表达式 s＝a1＋a2＋a3＋…＋an 就比较烦琐了，假如只有 10 个、20 个尚能勉强对付，如果有千人、万人呢？显然这样的表达式书写就不可行了。

借用字符开头后缀数字序号这个能够反映变量之间内在次序关系的变量命名规则的思想，为了与之前使用的变量在形式上有所区分，把序号用方括号[]括起来，这样就构成了一种新的变量的表示方法——数组，其形式为：数组名[序号]。当序号是变量形式时，通过改变这个变量的值可以操作数组中不同的元素，如 s[1]，s[2]，s[3]，…，s[i]等，通常使用循环来操作数组中的成员变量，如下例中的 a[i]。

```
for(i = 1;i <= n;i++)
    s = s + a[i];
```

从这个程序段中可以看出，当使用数组时，n 个数据的求和由 s = s + a[i]取代了多个变量相加的表达式 s = a1 + a2 + a3 + … + an，无论 n 的值如何增大，由数组构成的累加器都是

一样的简洁可行。

我们把这种用于**存储相同类型批量数据的变量的集合称为数组**。其含义可进一步表述为数组是变量的集合,这些变量拥有一个共同的名字为**数组名**,这些变量是按下标有序排列的,是用来存储同类型大批量数据的。数组中的每个成员称为**数组元素**,习惯上也称为**下标变量**;相应地把之前一个个单独使用的变量称为**简单变量**。数组元素以不同的下标来标志。在 C 语言中,数组中的元素在内存中是以连续的存储单元按顺序依次存放的,第一个数组元素对应低位地址,最后一个数组元素对应高位地址。

只有一个下标的数组称为一维数组,用于表示一行或一列数据;显然,**具有两个下标的数组称为二维数组**,其中一个表示行,另一个表示列,可用于表示一张二维表格中的数据。理论上还可以有多维数组,本节学习一维数组的使用。

如前所述,数组是变量的集合,也就是说数组具有变量的所有属性。变量必须先定义后使用,同样数组也必须定义后方可使用。

一维数组定义的语法格式如下:

```
数据类型 数组名[常量表达式];
```

示例:int m[10];

数组定义语法说明如下。

(1) 数据类型:C 语言中可用的任意数据类型,既可以是基本数据类型,也可以是构造数据类型。示例中的数组类型为 int。

(2) 数组名:合法的标识符,其命名规则与变量完全相同,示例中的数组名为 m。

(3) 常量表达式:**其值必须是整型常量**,不可以使用变量,表示数组中包含元素的个数,也称数组长度。示例中为 10。

示例中,定义了一个名称为 m 的数组,其中的每个元素可用于存放整型数据,共有 10 个数组元素,可使用的数组元素(也称下标变量)为:m[0]、m[1]、m[2]、m[3]、m[4]、m[5]、m[6]、m[7]、m[8]、m[9]。

(4) 一个定义语句中可以同时定义多个数组,也可以与简单变量混合定义。如:

```
float m,n,x[20],x[30],y[50];
```

数组定义有以下几点需要注意。

(1) 定义数组语句中的下标值表示数组中的元素个数,可用下标总是从 0 开始。

设数组定义语句中的下标值为 n,则共定义了 n 个数组元素,可用下标为 0～n－1。

假设每个 int 类型数据占用 4B,10 个元素共需要占用 4B * 10＝40B 存储空间,并且这些存储空间必须连续,其存储方式如表 5-1 所示。

表 5-1　一维数组存储空间表示

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

本质上,数组定义与变量定义意义相同,都是按照数据类型对内存的使用要求为数组元素申请相应的存储空间,不同的是数组定义中一个定义语句可以申请数量较多的元素个数,因此消耗的存储空间较大,使用时本着够用为准的原则,避免因此造成的内存浪费。

(2) 在一个函数中不允许出现数组名与变量名相同的情况。

例如：

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

是错误的。

(3) 常量表达式可以是一般常量或符号常量,但不可以为变量。

例如：

```
#define  N  20
int a[N];
```

是正确的。

而试图用以下方法定义数组

```
int x;
int a[x];
```

是错误的。

5.1.2　一维数组的引用


在线视频

所谓**数组的引用就是在程序中使用数组元素**。与简单变量相比,本质上一个数组就是同名的一批变量,一个数组元素就是一个变量,**凡是能使用变量的地方都可以使用数组元素**。在书写形式上,数组元素与简单变量有所不同,需要在名称标识符后增加一个方括号。要注意区别 a1 与 a[1]的不同含义：a1 是简单变量,而 a[1]是数组元素。引用一维数组元素的一般形式如下：

数组名[下标]

例如：

```
t = a[2];
if(a[i++]> 0)
   printf("%d",a[i]);
```

都是合法的数组元素引用。

以下三种情况为常见的数组引用形式。

(1) 赋值引用——数组元素出现在赋值语句的左右。

以下都是使用赋值运算符对数组元素形成的引用示例：

```
a[0] = t;                    //为数组元素赋值
t = a[0];                    //引用数组元素的值
a[2] = a[2] + b[2];          //数组元素运算赋值
```

(2) 关系表达式引用——数组元素出现在关系表达式中。

以下是数组元素用于表达式中的引用示例。

```
if(a[i]> 0)                  //数组元素用于构成分支条件表达式
while(a[i]> = a[j])          //数组元素用于构造循环条件表达式
```

(3) 输入输出引用——数组元素是输入输出的对象。

以下是数组元素作为输入输出项的引用示例。

```
printf("%d",a[i]);
```

```
scanf("%d %d",&a[i],&b[i]);
```

【例 5-1】 依次存储 10 个整数 1～10,然后按逆序输出这些值。

程序如下:

```
#include<stdio.h>
int main()
{   int i,a[10];                //定义数组
    for(i=0;i<10;i++)
       a[i]=i+1;                //赋值引用
    for(i=9;i>=0;i--)
       printf("%4d",a[i]);      //输出引用
    printf("\n");
    return 0;
}
```

运行结果:

```
10 9 8 7 6 5 4 3 2 1
```

结果分析:

按题目要求,需要存储 10 个整数,采用数组最为方便。

先定义一个整型数组 a,包含 10 个元素。

需要存放的数据初始值为 1,并按递增 1 的规律递变,因此这些数据可通过调整循环变量的值产生并存储在数组中,采用的表达式为:a[i]=i+1。

数据存储在数组中后要实现逆序输出,只需要改变数据的读出次序,即先读取最后一个元素中的数据,最后读取首元素中的数据。

【例 5-2】 编写一个歌手选拔赛评分程序。要求 5 位评委现场输入评分值,其中去掉最高分和最低分后的平均分作为最终成绩。需要程序输出最高分、最低分和最终得分。

```
#include<stdio.h>
int main()
{
    int i,max,min,a[6];
    float sum=0.0,ave;
    printf("请输入选手得分,数据之间以空格分隔:\n");
    for(i=1;i<=5;i++)
       scanf("%d",&a[i]);    //输入得分数据
    max=min=a[1];            //假设第一个分值是最高分和最低分的起始值
    for(i=2;i<=5;i++)        //从第二个数据开始依次挑选最高分和最低分
    {   if(a[i]>max)         //后续评分大于 max 的值,更新 max
           max=a[i];
       if(a[i]<min)          //后续评分小于 min 的值,更新 min
           min=a[i];
    }
    for(i=1;i<=5;i++)        //对所有评分求和
       sum=sum+a[i];
    ave=(sum-max-min)/3.0; //最后得分 ave=(总分-最高分-最低分)/3.0
    printf("本参赛选手的得分为:\n");
    for(i=1;i<=5;i++)        //输出所有得分,读取数组元素中的数据
      printf("%d ",a[i]);
    printf("\n 去掉最高分:%3d 去掉最低分:%3d\n",max,min);
```

```
    printf("选手最后得分为: %.2f\n",ave);
    return 0;
}
```

运行结果：

```
请输入选手得分,数据之间以空格分隔:
89 78 90 99 100
本参赛选手的得分为:
89 78 90 99 100
去掉最高分: 100 去掉最低分: 78
选手最后得分为: 92.67
```

结果分析：

本程序用到了数组的定义、数组元素的输入、计算、输出等常规引用方法。程序核心工作大概分为以下四个阶段。

第一阶段为数据的输入，由第一个 for 循环控制 scanf 函数完成。

第二阶段为查找最值，由第二个循环控制两个分支语句分别实现找最大值和最小值，这个循环的循环体是两个分支语句，因此必须用复合语句标志{}把两个分支括起来。循环结束后，从数组元素中把最大值存于 max 中，把最小值存于 min 中。

第三阶段为计算得分，由第三个循环控制实现，先读取数组中各元素的值进行求和存储于 sum 中，然后减去第二阶段获得的最高分 max 和最低分 min 并求平均值。

第四阶段为输出结果，由第四个循环控制输出 a 数组的全部元素的值，并分别输出计算得到的 max、min 和 ave 的值，也就是最高分、最低分和最后得分。

在使用数组进行数据操作时，大多涉及多个数据，因此有数组的编程大多会采用循环结构控制数组元素的输入、处理和输出操作。

5.1.3　一维数组的初始化


在线视频

与变量相同，**未经初始化或赋值的数组元素的值是不确定的**。如果希望数组在运算时有确定的值，比如用于累加时设置初值为 0，或用于乘积运算时初值为 1 等，需要给数组元素赋初值，称为**数组的初始化**。数组赋初值的基本方法是把元素的初值依次写在一对花括号内，数据之间用逗号分隔。数组初始化的一般语法格式如下：

数据类型　数组名[常量表达式]={初值列表};

例如：int a[6]={1,2,3,4,5,6};

【例 5-3】 定义两个数组，一个不进行初始化，一个进行初始化，分别输出两个数组中各元素的值，查看结果有何不同。

```
#include <stdio.h>
int main()
{
    int i,a[10];                //定义了整型数组 a 未赋值
    int b[10]={0};              //定义了整型数组 b 并赋初值 0
    printf("a 数组元素的值为: \n");
    for(i=0;i<5;i++)            //输出 a 数组元素的值
       printf("%d ",a[i]);
```

```
    printf("\n");
    printf("b 数组元素的值为: \n");
    for(i = 0;i < 5;i++)        //输出 b 数组元素的值
       printf(" % d ",b[i]);
    printf("\n");
    return 0;
}
```

运行结果:

```
a 数组元素的值为:
- 858993460   - 858993460   - 858993460   - 858993460   - 858993460
b 数组元素的值为:
0 0 0 0 0
```

结果分析:

a 数组定义时未经赋值,从数组中读取的值是相应内存单元中原先存放的数据,是无实际意义的随机数据; b 数组定义时赋了初值,因此输出的数据是有意义的预赋值 0。

数组初始化说明如下。

(1) 当初值的个数与数组长度相同时,可以省略数组长度; 若定义数组时没有指定初值,则数组长度不能省略。

示例

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

是正确的,系统自动根据初值个数定义了等长的数组元素个数。

```
int a[];
```

是错误的,系统不能确定数组元素的个数。

(2) 初值数量可以少于数组长度。

当指定的初值个数少于元素个数时,从首元素开始,把初值依次赋给数组元素,不足部分补 0。例如:

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

数组元素 a[0]、a[1]依次赋值 1、2,而其余的三个元素 a[2]、a[3]、a[4]自动赋值为 0。为此可采用以下方式将一个数组中全部元素赋初值为 0。以下两个语句等价:

```
int a[5] = {0};                 //其实是默认不足的元素补 0
int a[5] = {0,0,0,0,0};
```

(3) 初值数量不可多于数组长度。

当初值数量多于数组元素时,将会产生编译错误。如:

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

程序编译时产生错误报告:“error C2078:初始值设定项太多”。

(4) 除 0 以外,不可对数组整体赋值。

例如要给 5 个数组元素全部赋初值 1,须按以下格式书写:

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

而不能写成:

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

此语句实质是 int a[5]={1,0,0,0,0};

(5) 初值只能为常量,不能是变量,即使是已赋值的变量也不行。例如

```
int n = 10, a[2] = {n};
```

是错误的。

(6) 数组的初始化必须在定义数组的同时进行。以下两种方法有所不同。

方法一

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

是错误的。

方法二

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

是正确的。

5.1.4　一维数组程序设计示例


在线视频

【例 5-4】 初始化一个包含有 10 个整数的数组,把这个数组中的数据逆序存储,分别输出逆序前后的数据。

```
#include <stdio.h>
int main()
{
    int t,i,a[10] = {1,2,3,4,5,6,7,8,9,10};
    printf("原数据为: \n");
    for(i = 0;i < 10;i++)
       printf("%d ",a[i]);
    printf("\n");
    for(i = 0;i < 10/2;i++) //数据交换次数为数据总数的一半
    {    t = a[i];
         a[i] = a[9 - i];
         a[9 - i] = t;
    }
    printf("逆序后的数据为: \n");
    for(i = 0;i < 10;i++)
       printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

运行结果:

```
原数据为:
1 2 3 4 5 6 7 8 9 10
逆序后的数据为:
10 9 8 7 6 5 4 3 2 1
```

结果分析:

程序中先定义了整型数组 a,包含 10 个元素并赋初值。第一个循环的功能是输出数组

的原始值。第二个循环是本程序的难点,是程序的核心算法所在。for 循环从 i = 0 开始到 i = 4 结束,共执行 5 次循环,每次都将原始数据中前半部分依次与后半部分进行交换,即 a[0]与 a[9]交换、a[1]与 a[8]交换,以此类推。第三个循环完成逆序后的数组数据输出。

此例中,如果把 10 改为 N,即可变成对任意多个数据进行逆序存储的程序。

另外,无论 N 为奇数或偶数算法无差别,只是在奇数的情况下最后一个要交换的数为正中间的数据,自己完成一次交换操作,结果正确。

【例 5-5】 设以数组下标表示学生的学号,从键盘输入 10 名学生某门课的成绩,查找最高分和最低分及对应的学生学号并输出。

实现代码如下:

```
#include <stdio.h>
int main ()
{   int m1,m2,i;
    float max,min,score[11];
    printf("请输入 10 名学生成绩:\n");
    for(i = 1;i <= 10;i++)
       scanf("%f",&score[i]);
    max = min = score[1];
    m1 = m2 = 1;
    for(i = 2;i <= 10;i++)
    {   if(max < score[i])
       {   max = score[i];
           m1 = i;
       }
       if(min > score[i])
       {   min = score[i];
           m2 = i;
       }
    }
    printf("最高分为: %.2f \t 学号为: %d\n",max,m1);
    printf("最低分为: %.2f \t 学号为: %d\n",min,m2);
    return 0;
}
```

运行结果:

```
请输入 10 名学生成绩:
68 78 88 98 86 65 79 89 99 100
最高分是: 100.00     学号为: 10
最低分是: 65.00      学号为: 6
```

结果分析:

本程序先定义了数组 score,包含 11 个元素,0 号元素不使用。

第一个循环的功能是输入学生成绩。在数组有数据的情况下,先把 max 和 min 初始化为 score[1] 的值,用于记录最好成绩和最差成绩的变量 m1、m2 赋初始值为 1。

第二个循环是本程序的重点,循环变量从 i=2 开始到 i=10 共执行 9 次,每次分别把 max 和 min 的值与成绩数据进行比较,以两条线索获得最大值及最小值,同时记录成绩数据所对应的序号存储于 m1、m2 中,最后完成数据输出。

5.2　二维数组及使用

内容概述

一维数组能够处理批量的同类数据，但只限于数据为一行或一列的情况。现实中有很多问题所涉及的数据需要采用二维表格进行处理，其中的每个数据元素都由行号和列号两个维度的序号确定，C 语言中采用二维数组可以实现对这种数据的操作。本节学习重点为：

- 二维数组的定义、初始化及引用。

5.2.1　二维数组的定义


在线视频

假设有一个班 40 名学生一门课程的成绩需要处理，使用一维数组即可；可是如果需要处理一个班 40 名学生 5 门课程的成绩，就需要使用二维数组。

二维数组的定义与一维数组类似，其语法格式如下：

数据类型 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，长度必须是整型常量或表达式。

例如：

```
float x[40][5];
```

为加深理解和方便表述，我们定义如下数组：

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

该示例语句定义了一个 3 行 4 列的二维数组，共有 12 个元素，分别为 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]，二维数组中每个元素都有两个下标，都必须放在单独的方括号内。二维数组定义中常量表达式 1 表示该数组具有的行数，常量表达式 2 表示该数组具有的列数；两个数字的乘积是该数组的元素个数。

数组元素的存储理论上可分为**按行优先**和**按列优先**两种。C 语言采用**按行优先**存储的操作规则，即在内存中先顺序存放第 1 行的元素(按下标值也可以说是第 0 行元素)，接下来存放第 2 行的元素，以此类推。

二维数组 a[3][4]的存储空间如表 5-2 所示。

表 5-2　二维数组存储空间

	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

二维数组也可以看作是其元素是数组的一维数组。

如 a[3][4]可认为是由 a[0]、a[1]、a[2]这 3 个一维数组构成的数组,每个元素又都是一个一维数组,各自包含有 4 个元素,其中 a[0]的成员有 a[0][0]、a[0][1]、a[0][2]、a[0][3],a[1]和 a[2]的成员如表 5-3 所示。

表 5-3　二维数组的另一种理解

	0	1	2	3
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]

从表 5-3 中可以看出,可以把 a[0]、a[1]、a[2]看作 3 个一维数组的名字,每个一维数组中又包含 4 个元素。

这样解析二维数组的意义在于可以通过降低维数的方法来理解多维数组的概念。

依此进一步拓展,三维数组也可以看作是其元素是二维数组的一维数组,n 维数组可看作是其元素是 n－1 维数组的一维数组。在 C 语言编译系统中,以下数组定义是正确的。

```
int a[2][3][4];
int a[2][3][4][5];
```


在线视频

5.2.2　二维数组的初始化

二维数组的初始化与一维数组的初始化相比,语法规则大体上相同,其语法格式如下:

类型名 数组名[行数][列数]={初值表};

与一维数组相同,**二维数组的初始化必须在定义时进行。行数与列数必须是常量形式的数据,不可以为变量**。

二维数组的初始化实质上是使各个初始常量数据按顺序排列,与数组各元素建立对应关系并在内存中顺序存储的过程。分为以下几种具体的形式。

(1) 分组初始化。

书写形式上,将同一行的数据按序写在一对花括号内,可看作是建立了一个分组,不同行的数据分别建立分组。初始化过程为按照书写顺序,将第 1 组数据赋给第 1 行元素,第 2 组数据赋给第 2 行元素…以此类推,示例如下:

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

如果某一行元素为 0,可以按省略形式写成:

```
int i[2][3]={{1,2,3},{0}};
```

注意:分组初始化时,分组数及数据个数一定要与数组的结构一致,否则会出现错误。

```
int a[2][3]={{1},{2},{3}};        //分组数与数组定义的行数不一致
int a[2][3]={{1},{0,2,0,4}};      //第二行数组元素个数与初始值个数不一致
```

(2) 按行初始化。

书写形式上,将所有数据按序写在同一个花括号内,编译系统依照数据书写次序,先对首行各元素赋初值,然后再对下一行元素赋初值…以此类推。示例如下:

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

如果所有元素均为 0,则可以写成:

```
int i[2][3] = {0};
```

但以下初始化格式是错误的:

```
int a[2][3] = 0;                    //初始化值无花括号
```

二维数组初始化特别说明如下。

(1) 如果对全部元素赋初值,则定义数组时第一维长度可以省略(编译程序可以计算出长度),但是第二维长度在任何情况下都不可省略。

例如:

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

下例是错误的:

```
int i[2][ ] = {1,2,3,4,5,6};   //第二维缺失
```

(2) 可以只对部分元素赋初值,未赋值的元素自动取 0 值。

```
int i[2][3] = {{1},{4}};        //第一行元素为 1,0,0,第二行元素为 4,0,0
int i[2][3] = {1,2};            //第一行元素为 1,2,0,第二行元素为 0,0,0
```

可对数组中的特定位置赋初值,其他元素默认为 0,假如按以下格式定义及初始化数组:

```
int i[4][4] = {{1},{0,2,0},{0,0,3},{0,0,0,4}};
```

初始化之后数组元素的数据如下所示:

```
1 0 0 0
0 2 0 0
0 0 3 0
0 0 0 4
```

(3) 如果只对部分元素赋初值并且省略了第一维的长度,必须采用分组赋初值方式。否则会使初值因错位而不正确。例如:

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

编译系统根据初始化数据来确定数组的结构及初值。由于有三组数据,可确认该数组共有 3 行,每行有 5 个数据元素。初始化之后数组元素的数据如下所示:

```
1 1 1 0 0
0 0 0 0 0
0 0 3 3 3
```

(4) 二维数据的初始化也必须在定义数组的同时进行。这个规则与一维数组完全相同。如以下初始化是错误的:

```
int a[2][4];
a[2][4] = {{1},{0,2,0,0}};
```


在线视频

5.2.3　二维数组的引用

与一维数组相同,二维数组引用也是以数组元素为对象,一般引用格式如下:

数组名[下标 1][下标 2]

例如:

```
t = a[2][0];
```

引用结果是把 a 数组中行序号为 2、列序号为 0 的元素值赋给 t 变量。

二维数组的引用场景与一维数组也类似,从语法角度看,凡是可以使用变量的地方都可以使用二维数组。以下三种情况为常见的二维数组的引用形式。

(1) 赋值引用——二维数组元素出现在赋值语句的左右。

以下是使用赋值运算符对数组元素形成的引用示例。

```
a[0][0] = t;                                    //为数组元素赋值
t = a[0][0];                                    //引用数组元素的值
a[2][5] = a[2][0] + b;                          //数组元素运算赋值
```

(2) 关系表达式引用——二维数组元素出现在关系表达式中。

以下是数组元素用于表达式中的引用示例。

```
if(a[i][j]> 0)                                  //数组元素用于构成分支条件表达式
while(a[i][j]> = a[j][i])                       //数组元素用于构造循环条件表达式
```

(3) 输入输出引用——二维数组元素作为输入输出对象。

以下是二维数组元素作为输入输出项的引用示例。

```
scanf(" % d",&a[i][j]);
printf(" % d ",a[i][j]);
```

【例 5-6】 已知一个 4×4 的矩阵,第一行和第三行的值通过赋初值给定,第二行数据从键盘输入,第四行的值为前三行相应元素之和,用二维数组存储数据并输出。

程序如下:

```
# include < stdio.h >
int main()
{   int a[4][4] = {{2,4,6,8},{0},{3,5,7,9}};      //数组的初始化
    int i,j;
    printf("请输入矩阵第二行数据,以空格分隔!\n");
    for(i = 0;i < 4;i++)
       scanf(" % d",&a[1][i]);                    //数组的输入引用
    for(i = 0;i < 4;i++)
       a[3][i] = a[0][i] + a[1][i] + a[2][i];     //数组的赋值引用
    printf("矩阵中存储的数据为: \n");
    for(i = 0;i < = 3;i++)
    {  for(j = 0;j < 4;j++)
          printf(" % 4d",a[i][j]);                //数组的输出引用
       printf("\n");
    }
    return 0;
}
```

运行结果：

```
请输入矩阵第二行数据,以空格分隔!
1 2 3 4
矩阵中存储的数据为:
  2 4 6 8
  1 2 3 4
  3 5 7 9
  6 11 16 21
```

结果分析：

本例主要验证了二维数组的初始化、输入、计算及输出引用的基本功能，初步形成利用二维数组进行程序设计的基本要领。从示例中可以看出，由于数组元素的顺序性和数量多等特点，无论是输入、计算还是输出，凡涉及数组引用，基本都需要通过循环结构控制基本语句完成相关操作。一维数组使用单重循环控制实现元素引用，由于二维数组有两个下标，因此一般需要使用双重循环对数组元素引用，如 printf()语句中的 a[i][j]，本例中计算第四行元素值的语句 a[3][i] = a[0][i] + a[1][i] + a[2][i]中，由于问题中明确了第三行的元素数据是由前三行的对应元素之和构成，依据算法实现要求，行元素下标为已知数，只有列下标是变动的，因此只需要单重循环即可实现操作功能。

【例 5-7】 输出如图 5-1 所示的杨辉三角形。

```
                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
```

图 5-1　杨辉三角形的输出结构

程序如下：

```
#include <stdio.h>
#define N 9
#define L 18
int main()
{
   int a[N][N];
   int i,j,k;
   for(i = 0;i < N;i++)                          //控制行数
   {   for(j = 0;j < N;j++)                      //控制列数
       {  if(j == 0||j == i)                     //行列相同,或第 1 列,数组元素赋值为 1
             a[i][j] = 1;
          else
             a[i][j] = a[i - 1][j - 1] + a[i - 1][j]; //其他元素赋值
       }
   }
   for(i = 0;i < N;i++)
   {   for(k = 1;k < L - i * 2;k++)              //按逐行递减两个空格的方式输出空格
             printf(" ");
       for(j = 0;j < i + 1;j++)                  //输出数据
             printf("%4d",a[i][j]);
       printf("\n");
   }
   return 0;
}
```

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
```

结果分析：

按题目要求输出的杨辉三角形形式如图 5-1 所示，实际上图 5-1 的数据关系以图 5-2 的形式计算更加直观。

第一步，定义二维数组 a[9][9]，用来存储杨辉三角形各元素的值。

第二步，按杨辉三角形的数据关系，填写二维数组中各元素的值。本例中需要填写 9 行 9 列的一个下三角矩阵，用双重循环控制赋值语句填写数组元素，分别以 i、j 变量控制行号和列号。核心算法描述为：各行的第一列元素以及行号和列号相同的元素值为 1，即 a[i][0] = 1 或 a[i][j] = 1(当 i = j 时)，其他元素的值为上一行前一列和上一行本列元素的和，即 a[i][j] = a[i - 1][j - 1] + a[i - 1][j]。

```
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
```

图 5-2　杨辉三角形的数据存储结构

第三步，按规定格式输出各数组元素的数据。观察可以发现，所要求的输出形式是把图 5-2 所示的下三角矩阵数据输出为图 5-1 所示的等腰三角形。可通过在输出行左侧按逐行递减两个空格字符的方式输出一个字符串，然后再依次输出数组数据即可形成所要求的形式。

5.3　字符数组及使用

内容概述

字符型数据是 C 语言中的一种基本数据类型，在内存中字符型数据是以 ASCII 码形式存储的，每个字符占一字节。由于一个简单变量只能存放一个字符，因此之前学习过程中只涉及单个字符的使用方法。实际工作中，应用更为广泛的是字符串，如小到一个词，大到一句话或一篇文章，因此字符串的应用更加普遍和重要。字符数组的每一个元素都存放一个字符，连接的多个元素可以存储和处理字符串。本节学习重点为：

- 字符数组的定义及初始化。
- 字符串的操作及应用。


在线视频

5.3.1　字符数组的定义

字符串是由双引号括起来的字符组成的序列，以'\0'作为结束标志。在 C 语言中，只有字符型数据和变量，没有字符串型数据和变量，字符型数据使用类型符 char 进行定义。一

个字符占用一字节的存储空间，一个字符串需要连续的多个字节来存储，只有一个字符的字符串，也需要两字节存储，其中一个存放字符串结束标志('\0')。数组的多个元素为存储字符串提供了便利，因此字符串一般采用字符型数组操作。

字符数组就是用于存储字符型数据的数组，数组中的每个元素相当于一个字符型变量。字符数组既可以使用一维数组，也可以使用二维数组。与其他数组使用方法相同，字符数组也需要先定义后使用。

字符数组的定义与前面介绍的其他类型数组的定义格式相同，使用的类型标识符为 char。

(1) 一维字符数组的定义形式：

char 字符数组名[常量表达式];

例如：

```
char ch[5];
```

定义了一个长度为 5 的字符数组，其中 ch 是数组的名字，中括号内的 5 表示该数组包含 5 个元素，分别是 ch[0]、ch[1]、ch[2]、ch[3]、ch[4]，每个元素都是 char 变量，需要 1 字节来存储，所以数组 ch 需要占用 5 字节的存储单元，并且这 5 字节的存储空间必须是连续的。

(2) 二维字符数组的定义形式：

char 字符数组名[常量表达式 1][常量表达式 2];

例如：

```
char stu[5][8];
```

定义了一个字符型的二维数组 stu，由 5 行 8 列元素组成。

该数组用于存储字符数据时，能够存储 5 个字符串数据，每个字符串最多可以包含 8 个字符，如果以字符串形式赋值，因最后一个数组元素须存放字符串结束标志，因此实际存储的字符数比最大下标号少 1 个。

特别说明：由于 C 语言中字符型数据是以 ASCII 码形式存储的，因此也可以用整型数组来存放字符数据。

int a[5]与 char a[5]可以等价使用。

5.3.2　字符数组的初始化


在线视频

字符数组的初始化与前面介绍的其他数组的初始化相似，对字符数组的初始化就是给字符数组中各数组元素存储确定的字符值。字符数组初始化形式如下。

1. 对一维数组元素逐个初始化

字符数组的初始化也是在定义数组的同时进行的，一般语法格式如下：

char 字符数组名[常量表达式] = {初值表};

在定义字符数组时，把初值用花括号括起来，数据之间用逗号分隔，最后一个字符数据后面不需要逗号，花括号中的字符数据必须以单引号引起来才能表示单个字符。

例如：

```
char ch[5] = {'a','b','c','d','e'};
```

定义了一个长度为 5 的一维字符数组，并把 5 个字符依次赋给 ch[0]、ch[1]、ch[2]、ch[3]、ch[4]，如图 5-3 所示。

ch[0]	ch[1]	ch[2]	ch[3]	ch[4]
a	b	c	d	e

图 5-3　一维字符数组存储空间示意图

2. 对二维数组元素逐个初始化

二维字符数组的初始化与其他数据类型的二维数组初始化方法相同，一般格式如下：

```
char 字符数组名[常量表达式 1][常量表达式 2] = {{初值表 1},{初值表 2},…};
```

例如：

```
char stu[5][8] = {{'H','o','w'},{'S','t','u','d','e','n','t'},{'C'}, {'Y','o','u'},{'m','e'}};
```

定义了一个字符型二维数组，其构成为 5 行 8 列，赋值后初值表与数组元素的对应关系如图 5-4 所示。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
stu[0]	H	o	w	\0				
stu[1]	S	t	u	d	e	n	t	\0
stu[2]	C	\0						
stu[3]	Y	o	u	\0				
stu[4]	m	e	\0					

图 5-4　二维字符数组存储空间示意图

特别注意：

(1) 如果定义字符数组时没有进行初始化，则数组中各元素的值是不确定的。

(2) 如果初值表提供的初值个数(即字符个数)多于数组长度，则出现语法错误。

(3) 如果在花括号里将字符数组元素的所有初值都列举出来，则数组的长度可以省略；若只对部分元素初始化，则数组的长度不能省略，系统将字符常量值从序号小的元素开始赋值，其余的元素自动定义为空字符('\0')。

(4) 也可以用字符的 ASCII 码值给元素赋值。

如执行语句 char a[N] = {'1',65,'s'};后，a[0] = '1',a[1] = 'A',a[2] = 's'，数组中的其余元素均获得空字符。

3. 字符数组元素的引用

字符数组元素的引用方法与其他类型数组元素相同，引用字符数组中的一个元素，可获得一个字符。字符数组引用的一般语法形式如下：

```
字符数组名[下标]　　字符数组名[下标 1][下标 2]
```

例如：

```
t = ch[2];　　//这里的 ch[2]表示引用 ch 数组中下标为 2 的字符
```

```
t1 = ch1[1][1];                    //引用 ch1 数组中下标为[1][1]的元素
```

【例 5-8】 给定一个字符串，从键盘输入一个字符，查找该字符是否存在于给定串中，如果存在则返回其所在位置序号，如果不存在则返回找不到的提示信息。

```
#include < stdio.h >
int main()
{
   char x,a[20] = {'S','t','u','d','e','n','t','s'};
   int k = - 1,i = 0;
   printf("请输入查找字符:\n");
   scanf(" % c",&x);
   do
   {   if(x == a[i])              //表达式引用
       {  k = i;
          break;
       }
       i++;
   }
   while(a[i]!= '\0');
   if (k!= - 1)
       printf("所查找的字符 % c 是第 % d 个字符!",a[k],k + 1);      //输出引用
   else
       printf("查无此字符 % c!",x);
   printf("\n");
   return 0;
}
```

运行结果：

```
请输入查找字符:
t
所查找的数据是第 2 个字符!
```

```
请输入查找字符:
w
查无此字符 w!
```

结果分析：

本例中在条件语句 if(x == a[i])的关系表达式中，循环语句 while(a[i]!= '\0')的关系表达式中以及输出语句 printf("%c",a[k])中，都引用了字符数组。从中可以看出字符数组的引用方法与其他数值型数组的引用方法相同。从语法的角度看，凡是能使用字符型变量的地方都可以引用字符数组元素。

【例 5-9】 有一个菱形图案如图 5-5 所示，编写程序，用字符型数组存储并输出该图案。

```
#include < stdio.h >
int main()
{   char a[5][7] = {{' ',' ',' ','* '},
    {' ',' ','* ','* ','* '},
    {'* ','* ','* ','* ','* ','* ','* '},
    {' ',' ','* ','* ','* '},
    {' ',' ',' ','* '}};
```

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

图 5-5　菱形图案

```
    int i,j;
    for(i=0;i<5;i++)
    {   for(j=0;j<7;j++)
            printf("%c",a[i][j]);
        printf("\n");
    }
    return 0;
}
```

程序分析：

首先定义一个字符型的二维数组 a[5][7]，用于存储 5 行 7 列的字符图案，并以初始化的方式把菱形图案放于其中。本程序以分组逐个赋值的方式把图案字符赋给数组元素，遇到没有 * 的位置以空格代替。

然后用双重循环控制输出语句读出数组元素的值，外循环控制行数，内循环控制列数，共 5 行 7 列，按行依次输出，每行结束后输出换行符。


在线视频

5.3.3　字符串的使用

以上对数组的操作都是基于单个字符数据的，字符常量是以单引号标记的。实际应用中，字符串的操作更加普遍。字符串是以双引号括起来的字符序列，与单引号标记的字符型数据不同的是，字符串数据存储时系统会在尾部自动加'\0'标志。字符型数组是存储和处理字符串的有力工具，下面学习字符串数组及应用。

1. 用字符串常量对数组批量初始化

用字符数组存放字符串是最恰当的字符串处理方式，与字符处理相同，为了实现字符串的操作，经常需要先把已知的字符串常量值存储，然后再进行其他处理，这就需要对字符串数组进行初始化操作。

字符数组的初始化一般语法格式如下：

```
char 字符数组名[常量表达式]={"字符串"};
```

或

```
char 字符数组名[常量表达式]="字符串";
```

例如：

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

以上两种形式是等价的，都是将字符串“China”存放在字符数组中，但字符数组的长度应该至少为 6 个，因为要有一个元素用来存放字符串结束标志'\0'。此外，字符串必须用双引号括起来，外面可以有花括号，也可以省略花括号。

需要特别注意区分以下示例语句对字符串的操作：

```
char ch1[]={'a','b','c','d','e'};
char ch2[]={"abcde"};
```

以上两个语句是不等价的，字符数组 ch1 的长度为 5。而字符数组 ch2 的长度为 6。只要是以双引号形式赋初值，系统一定在字符串末尾自动补充字符串结束标志'\0'。

那么当字符串常量值与数组长度不一致时系统是如何安排数据存储的？若有以下初始化语句：

(1) char str1[20]="program!";

(2) char str2[]={"program!"};

(3) char str3[5]={"program!"};

3 个字符数组的存放情况是不同的。(1)中，由于数组长度大于字符串长度，多余的元素值都为空，字符长度为 0，也可以理解为存放字符结束标志'\0'。(2)中，由于未定义数组大小，数组元素个数与字符长度是吻合的，即数组元素个数为字符长度加 1。(3)中，由于数组的长度小于字符串的长度，只截取与数组等长的字符存放在数组相应元素中，因数组长度不足，所以未能按规则在字符串尾部存放字符结束标志'\0'，因此这个字符串存储属于异常情况，以字符串格式“%s”输出数组元素的值，结果是不正确的。

2. 字符串结束标志

从以上的示例可知，数组是存储字符串的有力工具，数组的长度是在程序开始处必须先定义的，且一旦定义其长度不可改变；而数组中存放的字符串在程序运行过程中是会经常发生变化的，两者在空间方面要达到完全的匹配是不可能的；为此 C 语言借用了 ASCII 码值为 0 的字符'\0'(也称为空字符)作为字符串结束标志。

字符串结束标志不计入字符长度，但会占用一字节的存储空间。换句话说就是字符串结束标志不会产生附加的有效字符长度，比如前面(1)中的字符串长度为 8 而不是 9。程序中是通过检测'\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，这样编程人员不必担心由于两者长度的不一致而导致错误。一般来说，只要保证数组长度大于字符串长度，存储就不会出现错误。如果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字符串的长度。

字符串结束标志'\0'是一个符号，它是可进行存储的，如定义并初始化数组：

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

字符串长度为 10，但数组长度默认为 11，系统按以下存储格式存放字符串：

0	1	2	3	4	5	6	7	8	9	10
C		p	r	o	g	r	a	m	.	\0

如果执行如下语句：

```
ch[5] = '\0';
```

则数组的存储将更新如下：

0	1	2	3	4	5	6	7	8	9	10
C		p	r	o	\0	r	a	m	.	\0

执行如下字符输出程序：

```
int i = 0;
while(i < = 10)
```

```
{    printf("%c",ch[i]);
     i++;
}
```

输出结果为：

```
C pro ram.
```

执行以下字符串输出程序：

```
int i=0;
while(ch[i]!='\0')
{    printf("%c",ch[i]);
     i++;
}
```

输出结果为：

```
C pro
```

显然，系统以第一个字符串结束标志之前的字符为字符串内容，会忽略之后的字符。


在线视频

5.3.4　字符数组的引用

与数值型数组类似，字符数组的引用可采用元素引用形式，凡是能使用字符型简单变量的语法中都可以使用字符型数组元素。与数值型有所不同的是，因为字符型数组可以存储字符串，所以字符型数组还可以通过数组名引用方式批量处理字符串。字符数组的主要引用方式包括赋值引用、条件表达式引用和输入输出引用。

1. 字符数组元素的引用——逐个字符引用

(1) 赋值引用。

通过引用字符数组中的一个元素，获取一个字符。

例如：

```
t=ch[2];
ch[0]=ch[9];
```

(2) 条件表达式引用。

例如：

```
if(a[i]!='\0')
while(a[i]=='y')
```

(3) 输入输出引用。

- 利用标准输入输出函数 scanf()和 printf()中格式符%c 进行字符输入输出。

例如：

```
for(i=0;i<=10;i++)
     printf("%c",ch[i]);
```

又例如：

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

- 使用专用的字符处理函数输入输出单个字符。

C 语言编译系统提供了两个函数：getchar()和 putchar()，专门用于单个字符的输入与输出操作，使用这两个函数时，需要在源文件中包含头文件声明 #include <stdio.h>。

输入单个字符函数的语法格式：

变量 = getchar(void);

该函数功能为从标准输入设备(如键盘)获取一个字符并赋给变量。从格式中可以看出它不需要任何参数。

如果变量采用数组元素形式，通过循环控制重复调用函数，可输入一个字符串并存于数组中。这种应用场景下不会自动保存字符串结束标志，为了保证字符串操作的准确性，需要用户专门保存'\0'到字符串末尾。

例如：

```
char chr[20];
for(i = 0;i < 10;i++)
    chr[i] = getchar();                    //重复输入并存储字符到数组中
chr[i] = '\0';                             //循环结束时,i 的值为 10,刚好指示字符串结束标志
                                           //的位置
```

输出单个字符函数的语法格式：

putchar(char 表达式);

与输入单个字符函数对应，该函数功能为在标准输出设备(如显示器)输出一个字符。

与输入函数不同的是它需要带参数，参数可以是字符型常量、变量、数组元素，也可以是一个符合输出值类型的表达式，如果表达式采用数组元素形式，也可以通过循环控制重复调用函数，输出一个字符串。

例如：

```
char str[20] = "ABCD",t = '8';
int k = 97;
putchar('X');                              //字符型常量
putchar(t);                                //字符型变量
putchar(str[0]);                           //字符型数组元素
putchar(k);                                //a 的 ASCII 码,将以字符形式输出
```

输出结果为：

```
X8Aa
```

【例 5-10】 使用字符输入输出函数，输入一个字符串(中间可以包含空格字符)，然后原样输出，确认输入的字符串是否保存成功。

```
#include <stdio.h>
int main ()
{   int i = 0;
    char str[20];
    printf("请输入一个字符串: \n");
    while((str[i] = getchar())!= '\n')     //先输入一个字符并赋给数组元素
       i++;                                //然后判断是否为回车符,不是则继续
    str[i] = '\0';                         //输入回车符后单独在串尾存储字符串结束标志
    i = 0;
    while(str[i]!= '\0')                   //重复执行输出字符函数输出字符串值
    {  putchar(str[i]);
```

```
        i++;
    }
}
```

运行结果：

```
请输入一个字符串:
Hello
Hello
```

2. 用字符数组引用字符串——字符串整体引用

与单个字符相比，存放字符串的数组一般会涉及多个元素，像赋值、关系表达式引用，直接操作整个数组是不可行的，需要借用相应的标准函数。但对于输入输出，可使用“%s”格式控制符批量实现字符引用，此时输入输出项要以**数组名引用**，而不可采用数组元素引用。

(1) 使用标准函数引用字符数组。

• 输入字符串

例如：

```
char ch[10];
scanf("%s",ch);
```

从键盘输入：

```
Program!
```

系统将按以下形式存储字符串：

	0	1	2	3	4	5	6	7	8	9	10
ch[ ]	P	r	o	g	r	a	m	!	\0	\0	\0

又例如：

```
char st1[10],st2[10];
scanf("%s%s",st1,st2);
```

从键盘输入：

```
Hello world!
```

系统将按以下形式分别在 st1、st2 数组中存储字符串：

	0	1	2	3	4	5	6	7	8	9	10
st1[]	H	e	l	l	o	\0	\0	\0	\0	\0	\0
st2[]	w	o	r	l	d	!	\0	\0	\0	\0	\0

字符串输入过程中，系统自动以空格或回车符作为分隔符，因此输入字符串中如果包含有空格，则系统将以此空格作为第一个字符串输入结束信息，其后的字符将以另一个字符串值处理。

例如：

```
char st1[6],st2[6],st3[6];
scanf("%s",st1);
```

从键盘输入：

```
How are you?
```

假如输出字符数组 st1 的值，结果为：

```
How
```

输入函数 scanf()中只有一个输入项字符串数组，因此从第一个空格开始，之后的字符没有被存储。系统将按以下形式存储字符串：

	0	1	2	3	4	5
st1[]	H	o	w	\0	\0	\0

• 输出字符串

在 printf()中使用“%s”格式可输出字符数组中各元素的值，直到遇见'\0'。如对输入字符串第 2 个例子中输入字符串后的数组执行如下输出语句：

```
printf("%s\n",st1);
```

输出结果为：

```
Hello
```

【例 5-11】 使用 scanf()输入一个字符串，然后用 printf()函数原样输出，验证输入的字符串是否保存成功。

```
#include <stdio.h>
int main()
{   char str[20];
    printf("请输入一个字符串: \n");
    scanf("%s",str);                    //此处使用数组名,为地址引用
    printf("存储在数组中的字符串为: \n");
    printf("%s\n",str);                 //此处同样为地址引用
}
```

运行结果：

```
请输入一个字符串:
Abcd1234
存储在数组中的字符串为:
Abcd1234
```

(2) 使用专用函数引用字符数组。

C 编译系统提供了专门用于输入输出字符串的函数 gets()和 puts()，它们依赖于字符数组，可存储和读取一串完整的字符串，输入函数自动保存字符串结束标志'\0'，而不像 getchar()必须手动处理，因此输出函数也会借此标志正确结束字符串的读取。

使用这两个函数时，也需要在源文件中包含头文件声明#include <stdio.h>。

• 字符串输入函数——gets()

语法格式：

gets(字符数组名);

该函数可作为独立的语句调用,其功能为从键盘接收一个字符串,存放在字符数组中,并在字符串末尾自动加上结束标志'\0'。

例如：

```
gets(ch);
```

调用该函数时,假设从键盘输入：computer,即可将字符串值赋给字符数组ch。数组中共有9个字符,而不是8个字符。末尾自动加上结束标志'\0'。

注意：使用gets()接收字符串时,它将空格字符当作普通字符接收,而不像scanf()以空格作为字符串结束。

• 字符串输出函数——puts()

语法格式：

puts(字符数组名);

该函数可作为独立的语句调用,其功能为将一个字符串数组中的字符或字符串常量输出到屏幕,无论是数组还是常量,都要求确保以'\0'结束。

例如：

```
char ch[ ] = {"computer program"};
char str[ ] = {"one\ttwo"};
puts(ch);
puts("abc123");
puts(str);
```

调用该函数即执行上述语句后,屏幕上显示：

```
computer program
abc123
one        two
```

注意,puts()函数输出的字符串中可以包含转义字符,也可以包含空格。但如果数组中不包含字符串结束标志将会出现错误的输出。

例如：

```
char s[10];
s[0] = '1',s[1] = '2',s[2] = '3',s[3] = '4',s[4] = '5';
puts(s);
```

运行结果：

```
12345 烫烫烫烫烫烫烫
```

执行结果属于输出错误。原因在于字符数组是按元素初始化的,系统不会自动存储字符串结束标志,初始化值也没有人为添加此标志；而puts()函数输出规则默认为遇到该标志结束输出,因此出现数组引用越界。若初始化改为以下语句：

```
s[0] = '1',s[1] = '2',s[2] = '3',s[3] = '4',s[4] = '5',s[5] = '\0';
```

输出结果为：

```
12345
```

【例 5-12】　使用 gets()输入一个字符串，然后用 puts()函数输出。

```
#include <stdio.h>
int main()
{
    char str[20];
    puts("请输入字符串: ");
    gets(str)
    puts("输出结果为: ");
    puts(str);
}
```

运行结果：

```
请输入字符串:
Abc def  \n123
输出结果为:
Abc def  \n123
```

可在字符串中包含空格或转义符，观察输出结果；试修改程序段，以赋初值的方式给定包含有转义符的字符串，再次观察输出结果，从而分析原因。

5.4　字符串专用库函数的使用

内容概述

在 C 语言库函数中提供了专门用于处理字符串的函数，这些函数给程序设计带来很大的方便。使用字符串处理函数与其他库函数方法相同，需要在源程序中加入相应的预处理头文件声明：#include <string.h>。本节学习重点为：

- 常用字符串处理函数的使用。

5.4.1　strcpy()——字符串复制函数


在线视频

调用本函数的语法格式：

strcpy(字符数组,字符串数据);

例如：

```
char str1[10] = "0",str2[10] = "abc123";
strcpy(str1,str2);
strcpy(str1,"汉字字符串常量!");
strcpy(str1,"English");
```

函数功能：把由第二个参数所表示的字符串值复制到字符数组(参数 1)中。

说明如下。

(1) 参数 1 必须为字符数组名，参数 2 可以为字符数组名，也可以为字符串常量。

(2) 参数 1 的数组必须有足够的长度来存储参数 2 中的字符。

(3) 复制过程遇到参数 2 中的字符串结束标志'\0'为止。复制后的字符串包含'\0'。

(4) 不可以把字符串常量或字符数组直接赋值给字符数组。可通过 strcpy()函数实现

这一功能。

以下所示代码段是错误的：

```
char str1[10],str2[10] = "abc123";
str1 = "Hello";
str1 = str2;
```

要进一步明确，字符常量、变量或字符数组元素之间是可以赋值的。以下所示代码段是正确的。

```
char str1,str2 = 'a';
str1 = 's';
str1 = str2;
```

【例 5-13】 测试 strcpy()函数的功能。

```
#include <string.h>
int main ()
{
    char str1[30] = "0",str2[10] = "abc123";
    puts("复制前的字符串值为: ");
    puts(str1);
    strcpy(str1,str2);
    puts("复制后的字符串值为: ");
    puts(str1);
    strcpy(str1,"汉字字符串常量!");
    puts(str1);
    strcpy(str1,"English!");
    puts(str1);
}
```

程序结果：

```
复制前的字符串值为:
0
复制后的字符串值为:
abc123
汉字字符串常量!
English!
```

从运行结果可见，参数 2 的值可以是字符型数组名，也可以是汉字或英语字符常量。


在线视频

5.4.2 strncpy()——指定长度的字符串复制函数

调用本函数的语法格式：

strncpy(字符数组 1,字符型数据,int n);

例如：

```
strncpy(str1,str2,3);
```

函数功能：strncpy()函数把第二个参数中位于最前面的 n 个字符复制到字符数组 1 中最前面位置处。结果是字符数组 1 中最前面的 n 个字符被第二个字符型数据中最前面的 n 个字符取代。

如有以下程序段：

```
char str1[20] = "123456789",str2[10] = "abcdefg";
```

```
strncpy(str1,str2,3);
puts(str1);
```

则输出结果为：

```
abc456789
```

如果第二个参数是字符串常量，结果也是正确的，下例的输出结果与上例相同。

```
strncpy(str1,"abcd",3)
```

说明如下。

(1) 本函数与 strcpy()功能相同，使用条件也相同，只在于多了第三个参数 n，用以指定复制的字符个数。

(2) 第三个参数 n 的值逻辑上不应该大于前面任一个字符数组中字符的长度。

5.4.3　strcat()——字符串连接函数

调用本函数的语法格式：

```
strcat(字符数组,字符型数据);
```

例如：

```
char str1[30] = "Hello,",str2[ ] = "how are you!";
strcat(str1,str2);
strcat(str1,"world!");
```

函数功能：把由第二个参数所表示的字符串值连接在字符数组(参数 1)的末尾处，并自动去除字符数组字符串末尾的'\0'。

说明如下。

(1) 连接后的字符串存储在参数 1 中。

(2) 参数 1 必须为字符数组名，参数 2 可以为字符数组名，也可以为字符串常量。

(3) 参数 1 所属数组必须有足够的多余空间来存储参数 2 的字符。

(4) 复制过程为从字符数组的结束标志处开始依次存放参数 2 的字符串值，参数 2 的结束标志保留，作为整个字符串的结束标志'\0'。显然参数 1 的'\0'被覆盖，整个字符串的长度为两个参数中字符串长度之和。

例如：

```
char ch1[ ] = "stu",ch2[ ] = "dent";
strcat(ch1,ch2);
puts(ch1);
puts(ch2);
```

执行上述语句，屏幕输出结果为：

```
student
dent
```

5.4.4　strcmp()——字符串比较函数

调用本函数的语法格式：

strcmp(字符串 1,字符串 2)

例如：

```
char a[ ] = {"abcd"},b[ ] = {"12345"};
if(strcmp(a,b)> 0)
   printf("a > b\n");
else if (strcmp(a,b) == 0)
   printf("a == b\n");
else
   printf("a < b\n");
```

函数功能：比较两个参数字符串 1 和字符串 2 的大小。函数的返回值为一个整数，其取值按如下规则确定。

(1) 如果字符串 1 > 字符串 2，则函数值为正整数 1。

(2) 如果字符串 1 = 字符串 2，则函数值为 0。

(3) 如果字符串 1 < 字符串 2，则函数值为负整数－1。

字符串比较过程：从两个字符串的首字符开始依次比较对应位置字符的 ASCII 码，直到遇到不相同字符出现或者遇到字符串结束标志停止比较，根据比较情况函数返回并获取相应的返回值。

【例 5-14】 字符串比较规则测试。

```
#include < stdio.h >
#include < string.h >
int main()
{
   printf("a 与 1ab 比较结果为: %d\n",strcmp("a","1ab"));
   printf("abayz 与 abd 的比较结果为: %d\n",strcmp("abayz","abd"));
   printf("abc 与 bcd 的比较结果为: %d\n",strcmp("abc","bcd"));
   printf("a321 与 a321 的比较结果为: %d\n",strcmp("a321","a321"));
   return 0;
}
```

程序结果：

```
a 与 1ab 比较结果为: 1
abayz 与 abd 的比较结果为: -1
abc 与 bcd 的比较结果为: -1
a321 与 a321 的比较结果为: 0
```

5.4.5　strlen()——求字符串长度函数

调用本函数的语法格式：

strlen(字符串);

例如：

```
char a[ ] = {"abc12345"};
printf("字符串长度为: %d\n",strlen(a));
printf("字符串占用的空间大小为: %d Byte\n",sizeof(a));
```

函数功能：计算由参数指定的字符串的个数(长度)，以函数返回值获得，其值不包括字符串结束标志。

本例的输出结果为：

```
字符串长度为：8
字符串占用的空间大小为：9 Byte
```

5.4.6　strupr()——小写字母转换为大写字母函数

调用本函数的语法格式：

strupr(字符串);

例如：

```
char a[] = {"Uppercase letter 其他字符 123 !"};
printf("字符串值为：%s\n",a);
printf("转换为大写字母后为：%s\n",strupr(a));
```

函数功能：将由参数指定的字符串中的小写字母转换为大写字母，其他字符不变。

本例中的输出结果为：

```
UPPRECASE LATTER 其他字符 123!
```

可以看出，本函数只对英语字母起作用，对汉字、数字、标点符号等其他字符无改变。

5.4.7　strlwr()——大写字母转换为小写字母函数

调用本函数的语法格式：

strlwr(字符串);

例如：

```
char a[] = {"Lowercase LETTER 其他字符 123 !"};
printf("字符串值为：%s\n",a);
printf("转换为小写字母为：%s\n",strlwr(a));
```

函数功能：将由参数指定的字符串中的大写字母转换为小写字母，其他字符不变。

本例中的输出结果为：

```
lowercase letter 其他字符 123!
```

与 strupr()函数相同，本函数也只对英语字母起作用。

【例 5-15】 以初始化方式确定一个字符串，从键盘输入一个查找字符，编程实现在字符串中统计包含查找字符的个数，输出字符串值和查找字符个数。

```
#include <stdio.h>
int main()
{   int i,count = 0;
    char str[80] = {"sbfAca1f5dkl6ejpp7d8fqi9wcwaxlfsalpyzye"},ch;
    printf("请输入一个要查找的字符：");
    ch = getchar();                          //输入一个字符存储在 ch 中
    for(i = 0;i <= strlen(str);i++)
       if(str[i] == ch) count++;             //统计字符串中查找字符出现的次数
    printf("字符串值为：\n%s\n",str);
    printf("查找字符 %c 在字符串中出现的次数为：%d\n",ch,count);
    return 0;
}
```

运行结果：

```
请输入一个要查找的字符：a
字符串值为：
sbfAca1f5dkl6ejpp7d8fqi9wcwaxlfsalpyzye
查找字符 a 在字符串中出现的次数为：3
```

【例 5-16】 已知两个字符串值由初始化实现了存储，编程实现：不使用标准库函数使两个字符串连接，构成一个新的完整的字符串存储并输出；分别统计原串与新串的字符个数并输出。

```
#include <stdio.h>
int main()
{   int i;
    int k,length = 0,length1 = 0,length2 = 0;
    char str[50] = {"The first string"},s[20] = {"The second one!"},ch;
    for(i = 0;s[i]!= '\0';i++)              //统计串的长度
       length2++;
    for(i = 0;str[i]!= '\0';i++)            //统计串的长度
       length1++;
    k = i;                                  //此时的 i 值正好是需连接的串首字符所在序号
    for(i = 0;s[i]!= '\0';i++)              //在串中逐个连接串的字符
    {   str[k] = s[i];
       k++;
    }
    for(i = 0;str[i]!= '\0';i++)            //统计连接后的总串长
       length++;
    printf("字符串原长度分别为：%d %d\n",length1,length2);
    printf("新字符串长度为：%d\n",length);
    printf("连接后的字符串为：%s\n",str);
    return 0;
}
```

运行结果：

```
字符串原长度分别为：16 15
新字符串长度为：31
连接后的字符串为：The first stringThe second one!
```


在线视频

5.5　数组实训案例

内容概述

为进一步巩固数组相关知识，这里甄选了利用数组进行程序设计的实训案例，旨在提升运用数组相关知识分析问题的能力，强化编程技巧。本节学习重点为：

- 数组在编程中的应用。

【案例 5-1】 求出矩阵对角线最大值。

【案例描述与分析】

用一个整型二维数组，存放一个 5×5 的矩阵，要求从键盘输入矩阵的值，找出主对角线

上其值最大的元素。

程序首先读入矩阵的值，然后将其存储到 5×5 的整型二维数组中。使用循环结构遍历主对角线上的元素，获取最大值并输出。

【案例实现】

```
#include <stdio.h>
int main()
{
   int a[5][5], i, j, max;
   for (i = 0; i < 5; i++)
      for (j = 0; j < 5; j++)
          scanf("%d", &a[i][j]);
   max = a[0][0];
   for (i = 0; i < 5; i++)
      for (j = 0; j < 5; j++)
          if ((i == j) && (max < a[i][j]))
              max = a[i][j];
   printf("max = %d", max);
   return 0;
}
```

运行结果：

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
max = 5
```

【案例 5-2】 求缩写词。

【案例描述与分析】

缩写词由一个短语中每个单词的第一个字母组成。例如 CPU 是短语 Central Processing Unit 的缩写；ATM 是短语 Automatic Teller Machine 的缩写。请编写程序实现对输入的字符串求解并输出缩写词。

可以假设输入的字符串单词之间以空格进行分隔。程序从键盘读得字符串，将其存储在字符数组中，使用循环结构遍历该数组，对于每个字符，判断其是否为单词的首字母，如果是则将其转换为大写字母存储于数组并输出。首字母的判断依据是：如果当前字符是字符串的第一个字符，或者前一个字符为空格，则认为是一个单词的首字母。

【案例实现】

```
#include <stdio.h>
int main()
{   int i;
    int word;
    char str[200];
    while (gets(str) != NULL)
    {
```

```
        word = 0;
        for (i = 0; str[i] != '\0'; i++)
        {   if (str[i] == ' ')
              word = 0;
            else if (word == 0)
            {
              word = 1;
              if (str[i] >= 'a' && str[i] <= 'z')
                 str[i] = str[i] - 32;
              printf("%c", str[i]);
            }
        }
        printf("\n");
    }
    return 0;
}
```

运行结果：

```
central processing unit
CPU
```

5.6　数组实践项目

内容概述

对于批量数据的处理，必须先对数据进行存储，然后才可以实施其他操作。数组对于批量数据的存储具有很好的功效。本次实践项目基于数组存储一卡通系统中用户的信息。本节学习重点为：

- 数组对于数据的存储及操作功能。

【项目分析】

本节中的开发重点是完成“新卡注册”功能设计。

在系统开发过程中，变量的规划是非常重要的环节。为了便于编写程序，应该对变量做出科学规划，本次开发中关于数组的规划如下所述。

```
int flag[N] = {0};                  //是否建卡标志,初始值均为 0,表示所有元素未建卡
char personname[N][10] = {"\0"};     /* 存放姓名,二维数组可以一维方式引用: 第一维代表用
户序号,如 N = 1 表示序号为 1 的用户姓名; 第二维用于存储姓名字符串 */
int cardnum[N] = {0};               //存放卡号,初始值为 0
float cardmoney[N] = {0.0};         //用于存储卡中金额
```

在充分体现本章中关于数组应用的前提下，考虑到篇幅的原因以及后续章节中会有更科学的方法实施本项目开发，在此只实现选项 1 的功能，为了使程序能够正确退出，也保留了选项 7 的功能。

【项目实现】

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

```
#define N 20
main()
{    int k,i,num = 1,cardnumber;
     float cardmone = 0.0;
     int flag[N] = {0};                                          //是否建卡标志
     char personname[N][10] = {"\0"};                            //姓名
     int cardnum[N] = {0};                                       //卡号
     float cardmoney[N] = {0.0};                                 //卡余额
     char choose = '\0';
     while(1)
     {   system(" cls");
         printf("\n\t||==================================||");
         printf("\n\t||                欢迎使用          ||");
         printf("\n\t||                一卡通系统        ||");
         printf("\n\t||==================================||");
         printf("\n\t||              功能菜单选项        ||");
         printf("\n\t||----------------------------------||");
         printf("\n\t||              1.新卡注册          ||");
         printf("\n\t||              7.退出系统          ||");
         printf("\n\t||----------------------------------||");
         printf("\n\t 请输入选项: ");
         scanf(" %c",&choose);
         switch (choose)
         {
           case'1':printf("\n\t 您选择");
              for (i = 1;i < num;i++)
                 if (flag[i] == 1)
                 //查找是否有无效卡,有就用该卡号新开,无就顺延
                   break;
              cardnumber = i;
              printf(" \n\t 请输入姓名:");
              scanf(" %s", personname[i]);
              printf(" \n\t 请输入要充值到卡内的金额:");
              scanf(" %f", &cardmoney);
              cardnum[i] = cardnumber;                           //存储卡号
              //strcpy(prdnumber].pname, personname);
              cardmoney[i] = cardmoney;                          //存储金额
              flag[i] = 0;                                       //卡号已分配标志
              if (cardnumber == num)
                 (num)++;
              printf("\t 卡号\t 姓名\t 余额\n");
              for(k = 1;k < num;k++)
                 printf("\t %-d\t %s\t %.2f\n",
                    cardnum[k],personname[k],cardmoney[k]);
              getchar();
              getchar();
              break;
         case'7':printf("\n\t 您选择\n");
            exit(0);
            break;
         default: printf("\n\t 输入错误,请重新输入.");
         }
     }
}
```

运行结果：

```
||==================================||
||              欢迎使用            ||
||              一卡通系统          ||
||==================================||
||              功能菜单选项        ||
||----------------------------------||
||              1. 新卡注册         ||
||              7. 退出系统         ||
||----------------------------------||
请输入选项: 1
您选择 1
请输入姓名: 李江
请输入要充值到卡内的金额: 100
卡号   姓名     余额
1     张国     123.00
2     李江     100.00
```

习题与实训 5

一、概念题

1. 什么是数组,其优点是什么?
2. 数组是存储和处理字符串最有力的手段,请说明原因。
3. 字符与字符串的输入输出方法有何区别? 有哪些方法可以使用? 举例说明。
4. 数组在定义之后是否可以动态更改? 请加以说明。
5. 数组获取值的方式有哪些? 请举例说明。

二、选择题

1. 有数组 int a[10],则对 a 数组元素的正确引用是(　　)。

   A. a[10]　　B. a[3.5]　　C. a(5)　　D. a[0]

2. 下列数组声明不正确的是(　　)。

   A. int n = 10;int a[n]={0};　　B. #define n 10 int a[n] = {0};

   C. int a[5 * 6] = {0};　　D. int a[5] = {0};

3. 以下对字符数组 s 声明不正确的是(　　)。

   A. char s[] = "Hello";

   B. char s[6] = {'H', 'e', 'l', 'l', 'o'};

   C. char s[4] = {'H', 'e', 'l', 'l', 'o'};

   D. char s[] = {'H', 'e', 'l', 'l', 'o'};

4. 以下代码输出结果是(　　)。

```
char s[] = "Hello,\0world";
printf("%s", s);
```

A. Hello,\0world　　B. Hello,0world

C. Hello,world　　D. Hello,

5. 以下正确的定义语句是(　　)。

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

B. float x[3][ ]={{1},{2},{3}};

C. long b[2][3]={{1},{1,2},{1,2,3}};

D. int y[ ][3]={0};

6. 若有声明 int a[][2] = {{1}, {2}};,数组 a 中包含了多少个数组元素?(　　)

A. 2　　B. 3　　C. 4　　D. 1

7. 若有声明 int a[][2] = {{1}, {2}},数组 a 中 值为 0 的元素有多少个?(　　)

A. 没有　　B. 1　　C. 2　　D. 4

8. 已知 int a[3][4];,则对数组元素引用正确的是(　　)。

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

9. 表示字符串结束标志的是(　　)。

A. '\n'　　B. '\r'　　C. '\0'　　D. NULL

10. 下面程序段的输出结果是(　　)。

```
int i;
int x[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
for (i = 0; i < 3; i++)
printf("%d", x[i][2 - i]);
```

A. 1 5 9　　B. 1 4 7　　C. 3 5 7　　D. 3 6 9

三、填空题

1. 在 C 语言中,二维数组在内存中的存储顺序是按_____存放。

2. 在 C 语言中,引用数组元素时,其数组下标的数据类型是_____。

3. C 语言中,无字符串数据类型和变量,只能通过_____操作字符串。

4. 若 int i[5]={1,2,3}则 i[2]的值为_____。

5. 在 C 语言中,定义了一个数组后,就确定了它所容纳的具有_____数据类型元素的_____。

6. 若有说明 int b[5][6] = {0};数组 b 中每个元素均可得到初值_____。

7. 字符变量只能处理单个字符,相比之下,字符数组可以处理_____。

8. 假定 int 类型变量占用两字节,且有定义 int x[10]={0,2,4};,则数组 x 在内存中所占字节数是_____。

9. 已知 char x[]="hello", y[]={'h','e','a','b','e'};, 则关于两个数组长度的正确描述是 x_____y。

10. 判断两字符串 s1,s2 是否相等,应使用函数_____。

四、判断题

1. 如果想使一个数组中全部元素的值为 0,可以采用如此语句: int a[10]={0}。

(　　)

2. 未指定长度的数组声明时,依据初始值列表来确定数组长度。 ( )

3. 有声明 int a[6]={1,2,3,4}; 则 a[6]=0。 ( )

4. C语言的数组下标从0开始。 ( )

5. C语言中,数组元素在内存中是顺序存放的,它们的地址是连续的。 ( )

6. C语言中,数组名是一个常量,是数组首元素的内存地址,可以重新赋值。 ( )

7. 已知字符数组 str1 的初值为"China",则语句 str2=str1; 执行后字符数组 str2 也存放字符串"China"。 ( )

8. C语言中,gets()函数的返回值用于存放输入字符串的字符数组首地址。 ( )

9. 使用字符串处理函数 strcmp()需要包含头文件 string.h。 ( )

10. 设有 int a;char abc[5]="abcd";,则 a=strlen("ABC");执行后 a 的值为5。 ( )

五、程序补充题

1. 以下程序的功能是:用数组求 Fibonacci 数列前20项,每五项换行。

```
#include <stdio.h>
int main()
{   int i;
    int f[20] = { 1,1 };
    for(i=2;i<20;i++)
        ____________________;
    for (i=0;i<20;i++) {
        if(i%5==0)
           printf("\n");
        printf("%6d",_________________);
    }
    return 0;
}
```

2. 以下程序的功能是:计算成绩数组的平均值,并将计算结果返回。

```
#include <stdio.h>
double grade_statistics (double grades[10])
{
   double grade = 0;
   for (int i = 0; i<10; i++) {
       _________________;
   }
   grade = grade / 10;
   return grade;
}

int main ()
{
   double grade, grades[10]
       = {77.7,79.6,85.3,92.4,89.0,97.9,76.8,100.0,57.9,65.8};
   grade = _________________(grades);
   printf("%f", grade);
   return 0;
}
```

3. 以下程序的功能是：统计输入字符串中数字的个数，并打印输出这些数字。

示例输入：and24adfg8saf

示例输出：

```
248
3
#include <stdio.h>
int main()
{   char str[100];
    int count = 0;
    int i;
    printf("请输入字符串: ");
    ____________________________;
    for (i = 0; str[i] != '\0'; i++)
    {   if (str[i] >= '0' && str[i] <= '9')
        {   printf("%c", str[i]);
            ________________;
        }
    }
    printf("\n%d\n", count);
    return 0;
}
```

六、应用题

1. 编程分别计算一个 5×5 整型矩阵主对角线元素和次对角线元素之和。

2. 现有一堆苹果，以 3×4 矩阵排列，每个苹果上都标有编号表示其果型大小，且每个苹果大小不一。现小王想要根据苹果上的编号，找出最大的苹果，并根据其行列拿出这个苹果。设计程序帮助小王找出最大的苹果。

3. 输入一个字符串，判断该字符串是否为回文，如果是输出“yes”；否则输出“No”。回文就是字符数组中心对称，即从左向右读和从右向左读的内容是一样的。

4. 输入一个长度不超过 100 的字符串存储于 string 数组中，分别统计出这个字符串包含的英文字母、空格、数字和其他字符的个数。

5. 设计一个算法，用一维数组存储学生成绩，查找其中最高分获得者的学号及成绩，并输出。假设数组下标值即为学号。