第5章 数　组

前面章节中使用的数据都属于简单数据类型，使用单一的存储单元来存储一个变量，如整型、浮点型、字符型。但是在实际应用中，常常需要处理大量同类型的相关数据，如1000个学生的成绩等，若定义1000个简单类型的变量肯定是不合适的。类似这样的数据在C语言中可以通过数组来处理。

本章主要介绍一维数组和二维数组的定义及使用，其中着重介绍一维数组，因为一维数组的使用更加频繁。同时，考虑到字符型数组与数值型数组在某些使用方式上会有区别，因此，对字符数组和字符串将单独进行介绍。

一维数组的定义与引用

5.1 一维数组

数组是含有多个数据项的数据结构，这些数据项称为数组元素，同一数组中的每个元素都属于同一个数据类型，可以根据元素在数组中所处的位置把它们一个个区分开来。

一个数组是具有相同类型的数据项的集合。

5.1.1 一维数组定义

最简单的数组类型就是一维数组。假设有10个学生的成绩需要处理，可以定义一个长度为10的一维数组：

```
int score[10];
```

编译器将10个存储单元与数组名score相关联，这些存储单元在内存中是相邻的，每一个单元都可以存储一个int型的数据，见图5-1。

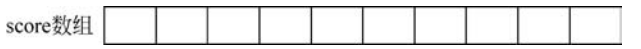


图5-1　有10个元素的一维整型数组示意图

与其他变量一样，数组也需要“先定义后使用”。为了定义数组，需要指明数组元素的类型和个数，数组元素的类型可以是C语言允许的任何类型，数组元素的个数可以用任何整数类型的常量表达式来指定。

一维数组的定义形式见表5-1。

表 5-1　一维数组的定义形式

语　法	类型标识符 数组名[数组长度];
示　例	int score[10];　//定义一个长度为 10 的整型数组
说　明	(1) 数组名的命名规则和变量名相同,遵循标识符的命名规则。 (2) 方括号内的数组长度表示元素个数。注意不是圆括号,如 int a(10)是错误的。 (3) 数组长度是整型的,可以是常量(表达式)或符号常量,不能是变量

C 语言不允许对数组的大小做动态定义,如以下语句是错误的。

```
int len;
scanf("%d",&len);  //企图临时输入数值作为数组大小
int a[len];        //用变量 len 作为数组长度,为错误的应用
```

C99 中,数组的长度可以用不是常量的表达式指定。前提是数组不具有静态存储期限且数组定义时未进行初始化。

一个数组中包含多个数组元素,为了存取特定的某个数组元素,可以在数组名的后面加一个用方括号括起来的整型数值来表示具体的数组元素,这种引用数组元素的方法称为“下标法”,即:

```
数组名[下标]
```

假设 a 数组的长度为 10,则其各个元素用下标法表示如图 5-2 所示,分别为 a[0]、a[1]、a[2]、…、一直到 a[9]。

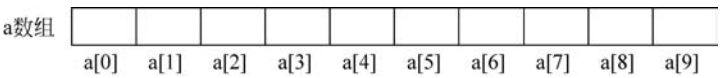


图 5-2　下标法表示一维数组中各个元素

需要特别注意的是,C 语言中,对数组元素引用时的下标是从 0 开始,而不是从 1 开始。即长度为 N 的数组,其第 1 个元素用 a[0]表示,第 2 个元素用 a[1]表示,……,第 N 个元素用 a[N−1]表示。

数组元素的作用相当于一个同类型的简单变量,因此,该类型的简单变量能进行的运算,数组元素也能进行。例如:

```
int score[10];
score[0] = 80;          //给下标为 0 的元素赋整数值
score[1] = score[0];    //将 score[0]元素的值赋给 score[1]元素
```

需要注意,定义数组时用到的“数组名[整型常量表达式]”和引用数组元素时用到的“数组名[下标]”从形式上看是相同的,但其实含义不同,例如:

```
int score[10];   //这里的 score[10]表示一个叫 score 的数组,其包含 10 个元素
temp = score[9]; //这里的 score[9]表示引用 score 数组中序号为 9 的那个元素,即第 10 个元素
```

数组下标:在数组名后的方括号内的数值或表达式,用于区分数组中的不同元素。

长度为 N 的数组,其各个元素的下标应该是从 0 到 N−1,而不是从 1 到 N。

5.1.2 一维数组初始化

在定义数组的同时，可以给各数组元素赋值，这称为数组的初始化。数组初始化的形式有多种，下面通过几个简单的例子进行示范。

【例 5-1】 给全部数组元素赋初值。某学生期中考试 4 门课程的成绩分别为 88，91，80，79，求其本次考试的平均成绩。

```
#include <stdio.h>
int main()
{
  float a[4] = {88,91,80,79};          //给全部数组元素赋初值
  float sum = 0,ave = 0;
  sum = a[0] + a[1] + a[2] + a[3];     //求总分
  ave = sum/4;                         //求平均成绩
  printf("Average = %.1f\n",ave);
  return 0;
}
```

运行结果：

```
Average = 84.5
```

本例采用在定义数组时“给全部元素赋初值”的形式初始化数组，语句：

```
float a[4] = {88,91,80,79};
```

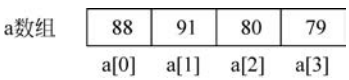


图 5-3 全部元素初始化后一维数组中各元素的取值

将数组元素的值按顺序放在一对大括号内，经过这样的初始化后，a 数组各元素的取值如图 5-3 所示。

对于数值型数组而言，只能给数组元素逐个赋值，而不能给数组整体赋值。比如要对长度为 4 的数组中所有元素都赋值 3，则应写成：

```
int a[4] = {3,3,3,3};                  //给每个元素都赋值 3
```

而不能贪图方便写成：

```
int a[4] = {3};                        //本例仅表示给 a[0]元素赋值 3,而其他三个元素赋值 0
```

也不能写成：

```
int a[4] = 3;                          //这是错误的数组初始化形式
```

【例 5-2】 给部分数组元素赋初值。求某学生 4 门课程的平均成绩，并将平均成绩放置在数组的最后一个元素中。

```
#include <stdio.h>
int main()
{
  float a[5] = {88,91,80,79};          //给前 4 个元素赋初值
  float sum = 0;
  sum = a[0] + a[1] + a[2] + a[3];     //计算总成绩
```

```
    a[4] = sum/4;                    //计算平均成绩并存储到数组的最后一个元素 a[4]中
    printf("Average = %.1f\n",a[4]);
    return 0;
}
```

本例的运行情况与例 5-1 相同。本例的数组定义及初始化语句如下。

```
float a[5] = {88,91,80,79};
```

此处将学生 4 门课程成绩和平均分都放在同一个数组中，因此数组的长度应为 5。其中，4 门课程的成绩是已知的，而平均分暂时未知，可见数组前 4 个元素初值是可以确定的，而最后一个元素的初值不能确定，因此对数组中的“部分元素赋初值”，初始化后各元素取值情况如图 5-4 所示。

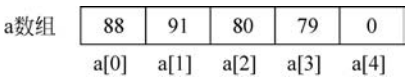


图 5-4　部分元素初始化后一维数组中各元素的取值

这时候，未初始化的元素 a[4]的值为 0，当程序后面部分的语句计算出平均值，并将平均值赋给 a[4]元素后，a[4]元素才获得更新后的值。

- 在定义数值型数组时，指定了数组长度并对其初始化，将按照大括号内的次序依次对数组中的各元素初始化，而未被赋初值的元素系统自动将它们初始化为 0。
- 若要对所有数组元素赋初值为 0，可以采用如下方式。

```
int a[10] = {0,0,0,0,0,0,0,0,0,0};
```

或

```
int a[10] = {0};
```

除了前面提到的“给全部元素赋初值”和“给部分元素赋初值”以外，还有一种初始化的方式，即在对全部数组元素初始化时，由于数据个数已确定，因此可以不指定数组长度。例如，在例 5-1 中的语句：

```
float a[4] = {88,91,80,79};          //给全部数组元素赋初值
```

可以写成：

```
float a[] = {88,91,80,79};           //给全部数组元素赋初值
```

在第二种写法中，大括号内有 4 个数，此时，虽然没有在前面的方括号中指定数组长度，但系统会根据大括号中数据的个数确定 a 数组有 4 个元素。但是，如果数组实际长度与提供初值的个数不相同，则方括号内的数组长度不能省略。例如，想定义一个长度为 10 的数组，其中有 5 个数值是已知的，则写成如下的语句是错误的。

```
float a[] = {80,90,80,70,60};        //该数组实际长度只有 5
```

必须写成：

```
float a[10] = {80,90,80,70,60};      //定义一个长度为 10 的数组，只初始化前 5 个元素，后 5 个
                                     //元素为 0
```

- 在定义数组的同时进行初始化时，尽量在方括号内给出数组长度，而不要省略长度

的描述。

如果定义数组时没有给数组元素初始化，数组元素的值是一个不确定的数据。

5.1.3 用循环结构存取数组

5.1.2节通过初始化的方式使数组元素获得值。但是，在实际应用中，数组元素的值往往需要通过交互方式获得，因此，在定义数组时进行初始化的方式并不是很实用，更多的是通过输入语句对数组中的元素赋值。

考虑到数组中的元素都是依次存放的，下标也是有规律可循的，因此可以结合循环语句来存取数组元素。

【例5-3】 一维数组基本练习。(nbuoj1149)

已知某学生期中考试4门课程的成绩，请将这4个成绩存放到数组中，然后计算其本次考试的平均成绩并输出，输出保留1位小数。

```
#include<stdio.h>
int main()
{
  float score[4];                    //数组长度为4,可存放4个成绩
  int i;
  float sum=0;
  for(i=0;i<4;i++)                   //用循环处理,控制数组元素下标从0到3进行变化
  {
    scanf("%f",&score[i]);           //输入一个成绩,保存到数组元素score[i]中
    sum+=score[i];                   //每次输入的成绩及时累加到sum变量
  }
  printf("%.1f\n",sum/4);
  return 0;
}
```

运行结果：

```
80 80 90 90↵
85.0
```

变量i是元素下标的计数器，它决定在每次循环过程中要操作数组的哪个元素。通常使用循环控制变量来担任这一角色，因为循环控制变量自增1更新以后，下一个数组元素就被自动选中了。

C语言不要求检查下标的范围，当下标超出范围时，编译器不会给出错误信息，程序也能正常运行，但会读取一些非法的内存空间从而得到错误的结果。

下标超出范围的主要原因是忽略了长度为N的数组其元素的下标是从0到N－1，而不是从1～N，因此在使用过程中常会出现以下两种错误情况。

(1) 将数组元素a[1]作为数组的起点，而忽略了元素a[0]。

(2) 将数组的最后一个元素的下标误认为是N。

错误形式见例5-4。

【例5-4】 一维数组逆序显示。(nbuoj1155)

输入10个整数保存到数组中，再逆序显示这10个数据。以下代码是错误的。

```
#include<stdio.h>
int main()
{
  int a[10],i;
  for(i=0;i<10;i++)      //输入 10 个元素,下标从 0 到 9
    scanf("%d",&a[i]);
  for(i=10;i>=0;i--)   //下标出错,误引用 a[10]元素了
    printf("%d ",a[i]);
  printf("\n");
  return 0;
}
```

(错误的) 运行结果:

```
1 2 3 4 5 5 4 3 2 1↵
1703792 1 2 3 4 5 5 4 3 2 1
```

本例在输出语句中误用了元素下标,对长度为 10 的数组出现了下标为 10 的元素,即 a[10],这种情况下编译不会出错,程序也能运行,但 a[10]元素的值不是我们期望的内容,如果将该元素列入计算范围的话,就会对结果造成影响。正确的做法是将输出语句改成如下形式。

```
for(i=9;i>=0;i--)      //长度为 10 的数组,元素下标只能是从 0 到 9
  printf("%d ",a[i]);
```

下面再看一个用数组解决斐波那契问题的例子。

【例 5-5】 斐波那契数列。(nbuoj1125)。

输入一个整数 n(1≤n≤12),输出斐波那契数列的前 n 项。

```
#include<stdio.h>
#define N 12
int main()
{
  int f[N]={1,1};          //对最前面两个元素 f[0]和 f[1]初始化为 0
  int n,i;
  scanf("%d",&n);
  for(i=2;i<n;i++)         //从第三个元素 f[2]开始,用公式计算
    f[i]=f[i-1]+f[i-2];
  for(i=0;i<=n-1;i++)  //输出前 n 项,即从 f[0]到 f[n-1]这 n 个数组元素
    printf("%d ",f[i]);
  printf("\n");
  return 0;
}
```

运行结果:

```
6↵
1 1 2 3 5 8
```

可以发现,用数组解决斐波那契数列问题比用简单变量来得更加方便和直观。

对数值型数组的输入或输出只能针对单个元素操作,不能整体地输入或输出一个

数组，假如有定义“int a[10];”，则不能用“scanf("%d",a);”，也不能用“printf("%d",a);”试图对整个数组元素进行输入或输出，而应该结合循环语句对元素逐个进行处理，如输入语句为：

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

而输出语句为：

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

顺序查找及二分查找

5.1.4 顺序查找与二分查找

查找是指根据某个给定的条件，在一组数据中搜索是否存在满足该条件的数据的过程。如果存在，则表示查找成功，给出成功标志；否则表示查找不成功，给出失败标志。

常用的查找方法有顺序查找和二分查找。

1. 顺序查找

顺序查找的基本思想是：把要查找的数据依次与待查的数比较，如果能够找到，则查找成功，否则查找失败。见例 5-6。

【例 5-6】 无序数组的查找。(nbuoj1151)

输入 10 个整数存入一维数组，假设这 10 个元素各不相同，再输入一个待查找的数据 key，查找数组中是否存在值为 key 的元素。如果有，则输出相应的下标，否则输出 not found。已知数组无序排列。

本题需要将待查找数据 key 依次与数组中的元素比较，一旦发现有相同数值说明查找成功，则可提前结束循环并输出所在位置的下标，而如果 10 个元素都比较完了依然没有相同元素出现，则查找失败。

```
#include<stdio.h>
int main()
{
  int key,num[10],i;
  int flag=0;                    //flag 标记查找是否成功,初值 0
  for(i=0;i<10;i++)              //输入 10 个数据存入数组
    scanf("%d",&num[i]);
  scanf("%d",&key);              //输入一个待查找的值存入 key 变量
  for(i=0;i<10;i++)              //顺序查找
  {
    if(key==num[i])              //如找到相同值
    {
      printf("%d\n",i);          //输出该数组元素下标
      flag=1;                    //将标记 flag 置 1,说明查找成功
      break;                     //可提前退出循环
    }
  }
  if(flag==0)                    //若查找失败,则输出对应提示信息
    printf("not found\n");
```

```
  return 0;
}
```

运行结果：

```
6 70 -9 80 83 54 3 88 10 2↵
80↵
3
```

【例 5-7】 最大数和最小数。(nbuoj1874)

输入任意 10 个整数，从中找出最大数值和最小数值并输出。

本题要求在 10 个数据间进行比较，分别求出一个最大数值和一个最小数值并输出。首先假设数组中的第一个元素(下标为 0)为最大值(最小值)，然后从第二个元素(下标为 1)开始逐一比较，看后面是否有更大(更小)的元素值。具体代码如下。

```
#include<stdio.h>
int main()
{
  int s[10],i,min,max;          //max 变量存放最大数值,min 变量存放最小数值
  for(i=0;i<10;i++)
    scanf("%d",&s[i]);
  max=s[0];                     //假设数组首元素为当前最大值,存入 max 变量
  min=s[0];                     //假设数组首元素为当前最小值,存入 min 变量
  for(i=1;i<10;i++)             //从下标为 1 的元素开始逐一比较
  {
    if(max<s[i]) max=s[i];      //寻找最大值
    if(min>s[i]) min=s[i];      //寻找最小值
  }
  printf("%d\n%d\n",max,min);
  return 0;
}
```

运行结果：

```
1 2 5 4 7 8 3 54 13 20↵
54
1
```

顺序查找对于数据没有要求，可以是未排序的数列，也可以是已排序的数列。

2. 二分查找

二分法查找法是在已经排好序的数中查找，并不需要将每个数据都和待查数比较，而是采用以下的方法(假设待查数据为 key，并且数据已经按从小到大顺序排列)，即每次用 key 与位于查找区间中央位置的元素进行比较，比较结果将会有以下三种情况。

(1) 如果相等，说明查找成功。

(2) 如果 key<中央位置元素，则如果有解的话，解应该位于查找区间的左半部分。此时将查找区间缩小为原来的一半(即左半部分)，并在这一半的区间中继续用相同的方法查找。

(3) 如果 key>中央位置元素，则如果有解的话，解应该位于查找区间的右半部分。此

时将查找区间缩小为原来的一半(即右半部分),并在这一半的区间中继续用相同的方法查找。

可以看出,用 key 与当前查找区间的中央位置元素比较后,要么找到数据 key,要么将查找区间缩小一半。如果查找区间不存在了依然没找到 key,则说明该数据序列中不存在 key。

【例 5-8】 有序数组的查找。(nbuoj1158)

输入 10 个升序排列的整数(假设没有重复数值)存入一个数组中,然后再输入一个待查找的数据 key,查找数组中是否存在值为 key 的数组元素。如果有,则输出相应的下标,否则输出 not found。

```
#include<stdio.h>
#define N 10
int main()
{
  int sucess=0;                          //用 sucess 来标记是否查找成功
  int location;                          //标识找到的数值的下标
  int a[N],i;
  int low,high,mid,key;
  for(i=0;i<10;i++)                      //输入 10 个数
      scanf("%d",&a[i]);
  scanf("%d",&key);                      //输入待查找数
  //二分查找法
  low=0; high=N-1;
  while(low<=high)
  {
    mid=(low+high)/2;
    if(key==a[mid]) {sucess=1;location=mid;break;}
    if(key<a[mid])  high=mid-1;
    else            low=mid+1;
  }
  if(sucess==1) printf("%d\n",location); //输出对应数的下标
  else printf("not found\n");
  return 0;
}
```

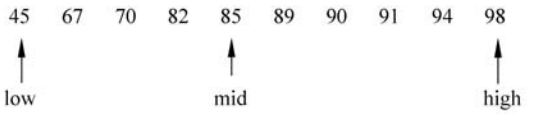
运行结果:

```
45 67 70 82 85 89 90 91 94 98↵
91↵
7
```

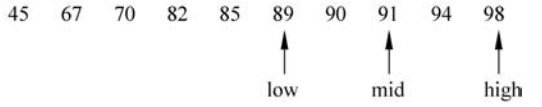
以本题为例,二分查找法首先用变量 low 和 high 来保存数组的首、尾元素的下标,low 与 high 之间的区域就是查找区域,且要求 low≤high。计算 mid=(low+high)/2,使 mid 保存中央位置元素的下标,见图 5-5(a)。接下来进行查找。

若 key=91,则 key>a[mid],因此执行"low=mid+1;",将查找区域缩小到右半区,并重新计算 mid 的值,见图 5-5(b)。

此时 key==a[mid],查找成功,该数值在数组中的下标为 7。



(a) 二分查找初始状态



(b) 一次查找后的状态

图 5-5　二分查找示意图

对于大小随机排列的数列，适合用顺序法查找，对于已经按大小排序的数列，用二分查找法效率更高。

5.1.5　一维数组的删除

要执行删除操作，首先要查找待删除的元素是否存在，若存在，则执行删除操作；否则报出错信息。

数组确定以后，其各个元素的空间也是确定的，不能“抹”去某一个元素空间，同时，元素空间中的数据也不能“撤销”，只能更新。因此，对数组元素的删除一般采用以下方法：将待删除数据后面的所有数据元素依次向前移，覆盖被删除的那个元素，这就相当于删除操作了。见例 5-9。

【例 5-9】　一维数组的删除。(nbuoj1154)

有 5 个整型数据存储在数组中，再输入一个数值 key，删除数组中第 1 个等于 key 的元素。如果 key 不是该数组中的元素，则显示 not found。

```
#include<stdio.h>
#define N 5
int main()
{
  int a[N];
  int i,j,key;
  for(i=0;i<N;i++)                    //输入原始数据
    scanf("%d",&a[i]);
  scanf("%d",&key);                   //输入待删除数据
  for(i=0;i<N&&a[i]!=key;i++) ;       //查找 key 是否存在,此处循环体为空语句
  if(i==N)                            //下标越界,说明待删除元素不存在
  {
    printf("not found\n");            //输出错误信息后就返回了
    return 0;
  }
  else
  {
    for(j=i;j<N-1;j++)                //待删除元素下标为 i
      a[j]=a[j+1];                    //后面的元素向前移动
```

```
    for(i=0;i<N-1;i++)                    //若删除成功,则输出剩下的 N-1 个元素
      printf("%d",a[i]);
      putchar('\n');
    }
    return 0;
}
```

运行结果：

```
80 65 93 100 81 ↵
93 ↵
80 65 100 81
```

本题的操作步骤如下。

(1) 查找待删除元素的位置,使用了以下语句：

```
for(i=0;i<N&&a[i]!=key;i++);
```

此处只是查找位置,不进行其他操作,因此循环体为空语句,用单独的分号表示。

(2) 查找成功,待删除位置的下标为 2,见图 5-6(a)。

(3) 将删除位置后面的元素向前移,覆盖待删除的元素(做形如 a[j]=a[j+1]的操作),见图 5-6(b)。

(4) 重复步骤(3),直到删除位置后的所有元素都前移,见图 5-6(c)。

从最终的图中可以看出,数组最后一个元素前移后其原有位置上的值并没有消失,因此最后一个元素有两份拷贝。这种元素前移的形式实现了对某个元素的“删除”效果,最后该数组的有效长度要相应减 1。

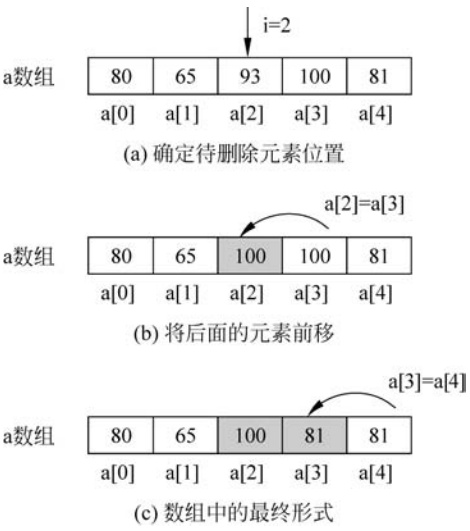


图 5-6　元素删除示意图

5.1.6　一维数组的插入

数据插入是指把一个给定的数据插到一个有序的数列中,并使数列依然保持有序。插入位置有以下两种情况。

(1) 在最后一个数组元素的后面。这是比较理想的一种插入位置,只要找到最后一个元素,在其后面添加待插入元素即可。这仅需要增加一个空间,而不涉及元素的移动。

(2) 除(1)中提到的位置以外的任何一个位置。这种情况是从数组最后一个元素至插入位置的元素,依次向后移一个位置,以“腾出”一个位置给待插入元素。

【例 5-10】 一维数组的插入。(nbuoj1153)

数组 a 中的 5 个数据按升序排列,从键盘输入一个待插入值 key,将其插入到数组中,使数组依然保持升序。

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

```
#define N 5
int main()
{
  int a[N+1]={0};                    //至少多留一个数组空间,以便插入新的元素
  int i,j,key;
  for(i=0;i<N;i++)                   //输入原始数据
    scanf("%d",&a[i]);
  scanf("%d",&key);                  //输入待插入数据
  for(i=0;i<N&&a[i]<key;i++)         //查找待插入的位置 i,循环停止时的 i 就是
  if(i==N)                           //若插入点在最后一个元素的后面
    a[N]=key;                        //则直接将待插入元素加在最后面
  else
  {
    for(j=N-1;j>=i;j--)              //元素依次后移,腾出空位
      a[j+1]=a[j];
    a[i]=key;                        //在下标为 i 的位置插入元素
  }
  for(i=0;i<N+1;i++)                 //输出插入以后的全部数组元素
      printf("%d ",a[i]);
    putchar('\n');
  return 0;
}
```

运行结果：

```
61 65 78 87 95 ↵
86 ↵
61 65 78 86 87 95
```

以插入元素 86 为例，本题的操作步骤如下。

(1) 查找待插入位置，使用了以下语句：

```
for(i=0;i<N-1&&a[i]<key;i++) ;
```

最后查找到插入位置的下标 i 为 3，见图 5-7(a)。

(2) 从最后一个元素到待插入位置的元素依次后移，腾出相应位置(做形如 a[j+1]=a[j]的操作，用递减循环实现)，见图 5-7(b)。注意，最后一个元素先开始后移。

(3) 重复步骤(2)，直到腾出相应的“空”位置。待插入位置的元素也有了两个拷贝，即相当于有了一个“空”位置，见图 5-7(c)。

(4) 在“空”位置插入新元素，即“a[i]=key;”，见图 5-7(d)。

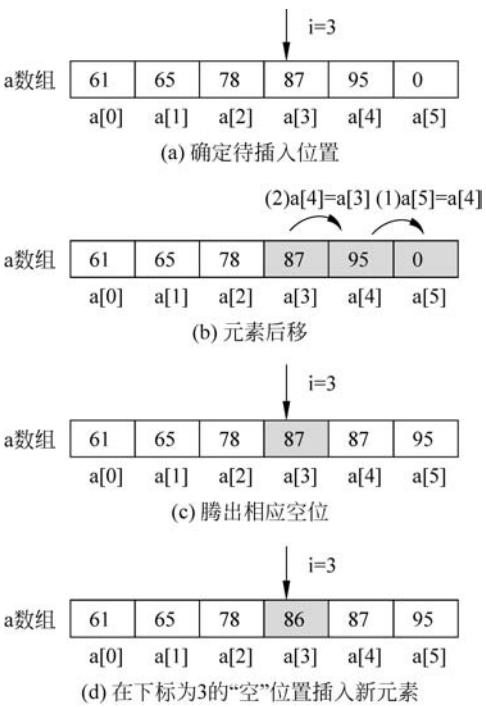


图 5-7　元素插入示意图

插入数据要有多余的空间，因此，数组长度要适当定义的长一些。

5.2 一维数组与排序

将一组无序的数列重新排列成升序(从小到大)或降序(从大到小)的形式是经常用到的操作,如从高到低显示学生成绩,或从低到高显示物品价格等,这种问题称为数的排序。排序的算法有很多种,本节主要介绍选择排序、冒泡排序。

选择排序

5.2.1 选择排序

选择排序是一种简单的排序算法,以升序为例,其基本的算法思想是:每一趟从待排序的元素中选中关键字最小的元素,顺序放在已排好序的子表的最后,直到全部元素排序完毕。

假如要对元素序列 $a_1 \sim a_n$ 按升序排序,则基本的操作思想如下。

(1) 在待排序的 n 个数 $a_1 \sim a_n$ 中找出最小数,将它与 a_1 交换。此时 a_1 成为更新后的有序部分内容,$a_2 \sim a_n$ 是新的待排序部分内容。

(2) 在剩下的待排序的 n−1 个数 $a_2 \sim a_n$ 中找出最小数,将它与 a_2 交换。此时 a_1、a_2 成为更新后的有序部分,$a_3 \sim a_n$ 是待排序部分。

(3) 用同样方法每趟从待排序的 $a_i \sim a_n$ 中选择最小的元素,将其与待排序部分的第一个元素 a_i 交换,见图 5-8(a)。交换后,a_i 成为新的有序部分的最后一个元素,见图 5-8(b)。

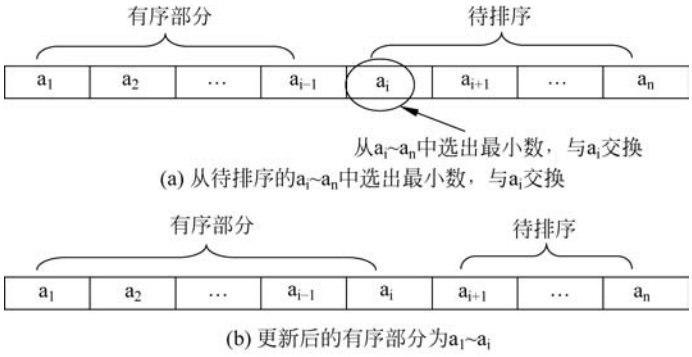


图 5-8 选择排序示意图

假设给定的数据序列为 78,64,37,21,22,77,则排序过程可如表 5-2 所示,其中,阴影表示有序部分,无阴影表示待排序部分。

表 5-2 选择排序过程

初始状态	78	64	37	21	22	77
第 1 次排序后	21	64	37	78	22	77
第 2 次排序后	21	22	37	78	64	77
第 3 次排序后	21	22	37	78	64	77
第 4 次排序后	21	22	37	64	78	77
第 5 次排序后	21	22	37	64	77	78

【例 5-11】 简单一维数组排序。(nbuoj1156)

输入 10 个学生的成绩，按从高到低的顺序显示这 10 个成绩。

本题要求按降序方式排序，在选择排序的过程中，每次要选出一个当前区域最大的数值。

```
#include<stdio.h>
#define N 10
int main()
{
  int a[N];
  int i,j,maxloc,temp;
  for(i=0;i<N;i++)
    scanf("%d",&a[i]);
  //选择排序
  for(i=0;i<N-1;i++)   //若有 N 个数,则需要选择交换 N-1 次
  {
  maxloc=i;              //先假设待排序区域的第一个数为最大值,将其下标 i 保存到 maxloc 中
  for(j=i+1;j<N;j++)   //寻找 i～N-1 中真正最大的数值,记录其下标到 maxloc 中
    if(a[j]>a[maxloc])
      maxloc=j;
    if(maxloc!=i)
    {      //若最后选出的最大值不是原来假设的待排序区域的第一个数,则对这两个数进行交换
    temp=a[i];       a[i]=a[maxloc];       a[maxloc]=temp;
    }
  }
    for(i=0;i<N;i++)
    printf("%d ",a[i]);
  printf("\n");
  return 0;
}
```

运行结果：

```
90 80 70 60 50 91 72 18 2 0↵
91 90 80 72 70 60 50 18 2 0
```

5.2.2 冒泡排序


冒泡排序

冒泡排序是较为常用的一种排序方法，它是一种具有“交换”性质的排序方法。以升序为例，其基本思想是：每次比较相邻的两个数，将小数放在前面，大数放在后面。

假设有 5 个数 9、8、7、6、5 需要按升序排序，则冒泡排序的过程可以描述如下。

第一趟：先比较第一、二个数，即 9 和 8，将小数 8 放前，大数 9 放后。然后比较第二、三个数，即 9 和 7，将小数 7 放前，大数 9 放后，如此继续，直至将第 n－1 个数与第 n 个数进行比较为止。第一趟排序结束后，将 n 个数中的最大数 9 放到了序列的尾部，即第 n 个位置上。如图 5-9 所示，经过第一趟(4 次比较和交换)后，最大数 9 已经“沉底”，而小的数都向上“浮”了一位。

第二趟：对剩下的 n－1 个数进行同样操作，一直比较到倒数第二个数(倒数第一的位置上已经是最大的 9)，第二趟排序结束，在倒数第二的位置上得到一个新的最大数 8(在整个数列中是第二大的数)。如图 5-10 所示，经过第二趟(3 次比较和交换)后，得到次大的数 8。

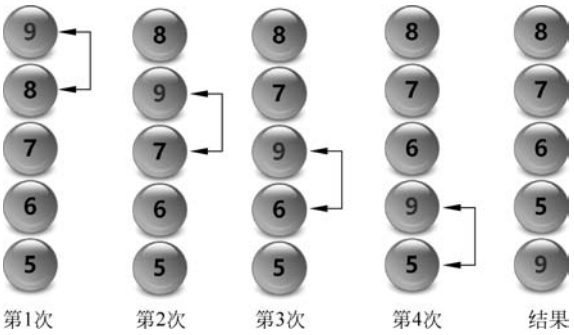


图 5-9　冒泡排序的第一趟过程示意图

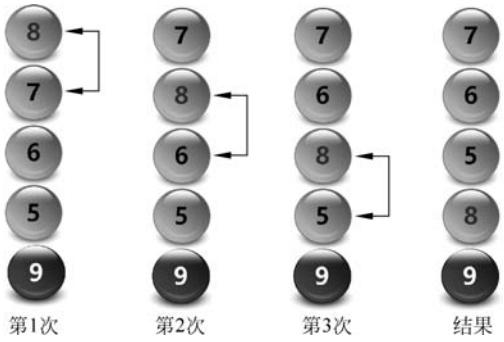


图 5-10　冒泡排序的第二趟过程示意图

如此进行下去，当执行完第 n－1 趟的冒泡排序后，就可以完成所有数的排序过程。

由于在(升序)排序过程中总是小数往前放，大数往后放，相当于气泡往上升，所以称作冒泡排序。但在实际使用中，也可以将大数往前放，小数往后放，完成降序的排序过程。

按照图 5-9 和图 5-10 的思路分析下去可知，一个包含 n 个元素的序列要进行 n－1 趟的冒泡排序。

【例 5-12】 简单一维数组排序。(nbuoj1156)

输入 10 个学生的成绩，按从高到低的顺序显示这 10 个成绩。本例用冒泡排序法完成。

本题要求按降序方式排序，在冒泡排序的过程中，两两比较时，要将大数放前面，小数放后面。

```
#include<stdio.h>
#define N 10
int main()
{
  int a[N];
  int i,j,temp;
  for(i=0;i<N;i++)
    scanf("%d",&a[i]);
  //冒泡排序
  for(i=0;i<N-1;i++)      //10 个数据需要 9 趟排序
  for(j=0;j<N-1-i;j++)  //在第 i 趟排序中,比较 9-i 次
  {
```

```
    if(a[j]<a[j+1])          //相邻元素比较
    {
      temp=a[j];
      a[j]=a[j+1];
      a[j+1]=temp;
    }
  }
  for(i=0;i<N;i++)
    printf("%d",a[i]);
  printf("\n");
  return 0;
}
```

运行结果：

```
10 80 70 60 50 91 72 18 2 0 ↵
91 80 72 70 60 50 1810 2 0
```

二维数组的定义及使用

5.3 二维数组

在很多场合需要用到二维数组，例如，有三个学生，每位学生有四门课程的成绩，要求对学生成绩用数组保存并处理，就需要用到二维数组，见图 5-11。这里有 3 行，每行代表一个学生的成绩，有 4 列，每列代表一门课程的成绩，这种按行(row)、列(column)排列的形式，有助于理解二维数组的逻辑结构。

	成绩1	成绩2	成绩3	成绩4
学生1	90	91	95	90
学生2	85	83	88	80
学生3	78	76	78	75

图 5-11 三个学生的四门课程成绩

5.3.1 二维数组的定义和引用

二维数组的定义与一维数组的定义相类似，其一般形式见表 5-3。

表 5-3 二维数组的定义形式

语　法	类型标识符 数组名[长度 1][长度 2];
示　例	int s[3][4]; //定义一个 3 行 4 列的整型数组，有 12 个元素
说　明	(1) 类型标识符号、数组名、长度 1 和长度 2 的要求同一维数组。 (2) 长度 1 表示行数，长度 2 表示列数。 (3) 长度 1 和长度 2 分别用两个方括号括起来，若写成 int s[3,4]是错误的。 (4) 引用一个二维数组的元素，必须同时给出行下标和列下标，如 s[0][0]表示 0 行 0 列的元素。 (5) 若行数为 m，列数为 n，则行下标的范围为 0～m－1，列下标的范围为 0～n－1。对于 m 行 n 列的二维数组，不存在形如 s[m][n]的元素

例如：

```
int s[3][4];
```

定义 s 为 3 行 4 列的数组，第一个下标 3 表示行数，第二个下标 4 表示列数，该数组一共包

含 12(3×4)个元素。

二维数组在逻辑上可以看作一个由若干行组成的特殊的一维数组，例如，把 s 看作一个特殊一维数组，它有 3 个元素 s[0]、s[1]、s[2](可以把 s[0]、s[1]、s[2]看作 3 个特殊的一维数组的名字)，而每个元素又是一个包含 4 个元素的一维数组，如 s[0]包含 s[0][0]、s[0][1]、s[0][2]和 s[0][3]，见图 5-12。

图 5-12　二维数组逻辑结构示意图

虽然以表格形式显示二维数组，但是实际上二维数组元素在内存中占用一片连续的存储空间，其值是“按行顺序存放”的，即先存放行下标为 0 的各元素，接着存放行下标为 1 的各元素，以此类推。图 5-13 显示了 s 数组的存储示意图。

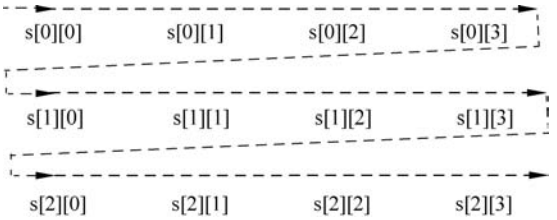


图 5-13　二维数组存储结构示意图

假设数组 s 存放在从字节编号 1000 开始的一段内存单元中，每个数据元素占 4B，则 1000～1015 的字节存放第 0 行的 4 个元素，1016～1031 的字节存放第 1 行的 4 个元素，1032～1047 的字节存放第 2 行的 4 元素，见图 5-14。

图 5-14　二维数组内存示意图

二维数组元素的表示形式为：

数组名[行下标][列下标]

例如，s[1][2]表示 s 数组中行号为 1、列号为 2 的元素。

- 表示一个二维数组元素时，行下标和列下标都要用独立的方括号括起来，如 s[1][0]是正确的表示形式，而 s[1,0]则是错误的表示形式。
- 在引用数组元素时，下标值应在已定义的数组大小的有效范围内。如在“int s[3][4];”所定义的 s 数组中，不存在 s[3][4]这样的数组元素。

5.3.2　二维数组的初始化

与一维数组一样，可以在定义二维数组的同时为其元素赋值，即初始化。见

例 5-13。

【例 5-13】 按行分段初始化。有三位同学参加了数学、英语、C 语言三门课程的考试，在程序中对各门成绩初始化，并求出每位同学的平均成绩。

```
#include <stdio.h>
#define M 3                                    //表示有三位学生
#define N 3                                    //表示有三门课程
int main()
{
  int i,j;
  float score[M][N] = {{95,68,78},{65,77,88},{94,82,73}};      //按行分段初始化
  float sum[M] = {0},ave[M] = {0};             //表示每位同学的总分和平均分
  for(i = 0;i < M;i++)
  {
    for(j = 0;j < N;j++)
      sum[i] = sum[i] + score[i][j];           //求每位同学三门课程的总分
    ave[i] = sum[i]/N;                         //求每位同学的平均分
  }
  printf("Student -- Math -- English -- C Language -- Average\n");
  for(i = 0;i < M;i++)                         //输出三位同学三门课程的成绩,以及每位同学的平均分
  {
    printf("NO %2d",i + 1);
    for(j = 0;j < N;j++)
      printf(" %8.1f",score[i][j]);
    printf(" %12.1f\n",ave[i]);
  }
  return 0;
}
```

运行结果：

```
Student--Math--English--C Language--Average
NO  1    95.0     68.0     78.0     80.3
NO  2    65.0     77.0     88.0     76.7
NO  3    94.0     82.0     73.0     83.0
```

本例采用如下语句对数组进行初始化。

```
float score[M][N] = {{95,68,78},{65,77,88},{94,82,73}};
```

这是一种“按行分段初始化”的方式，把内层第一个大括号内的数据赋给 0 行的元素，第二个大括号内的数据赋给 1 行的元素，……，即按行给二维数组元素赋初值。本例的初始化效果见图 5-15。

也可以把所有的初值写在一对大括号内，按数组元素排列的顺序对各元素赋值，称为“按行连续初始化”，例如：

```
float score[M][N] = {95,68,78,65,77,88,94,82,73};
```

在给数组中所有元素都赋值的情况下，这两种方法是等效的。但是按行分段初始化的方法更好，一行对一行，

score数组

0行	95	68	78
1行	65	77	88
2行	94	82	73

图 5-15 按行分段给二维数组赋初值

比较清晰直观。对于按行连续初始化的方法，如果数据很多的话，容易遗漏，也不容易检查。

如果对全部元素初始化，则在定义二维数组时长度1可以不指定，但长度2不能省。例如：

```
float score[3][3] = {95,68,78,65,77,88,94,82,73};
```

与下面的定义等价：

```
float score[ ][3] = {95,68,78,65,77,88,94,82,73};
```

系统会根据大括号内数据的总个数和长度2算出长度1的值。这里大括号内一共有9个元素，每行3列，则可以确定长度1的值为3。

初始化时也可以只对数组中的部分元素进行初始化，见例5-14。

【例5-14】 对部分元素初始化。题目内容同例5-13。现要求将平均成绩放在每行的最后一个位置进行处理。

按照题目要求，需要设计3行4列的二维数组score[3][4]，多出的这一列存放每位同学的平均成绩。这最后一列数据的值在初始化时尚不能确定，需要在程序运行过程中计算得到，因此初始化语句可以写成如下形式。

score数组

0行	95	68	78	0
1行	65	77	88	0
2行	94	82	73	0

图5-16 对部分元素初始化

```
float score[3][4] = {{95,68,78},{65,77,88},{94,82,73}};
```

即只对每行的前3列赋初值，其余元素自动为0，见图5-16。

```
#include <stdio.h>
#define M 3                                    //表示有三个学生
#define N 4                                    //列数增加到4,最后一列表示平均分
int main()
{
  int i,j;
  float score[M][N] = {{95,68,78},{65,77,88},{94,82,73}};
                                               //只对每行前3列元素初始化,其余自动为0
  float sum[M] = {0};                          //每位同学的总分
  for(i = 0;i < M;i++)
  {
    for(j = 0;j < N - 1;j++)
      sum[i] = sum[i] + score[i][j];           //求每位同学三门课程的总分
    score[i][N - 1] = sum[i]/(N - 1);          //求每位同学平均分,保存到该行最后一列的位置上
  }
  printf("Student --- Math -- English -- C Language -- Average\n");
  for(i = 0;i < M;i++)
  {
    printf("NO %2d",i + 1);
    for(j = 0;j < N;j++)
      printf(" %9.1f",score[i][j]);            //输出三门课程的成绩及平均分
    printf("\n");
  }
```

```
  return 0;
}
```

运行结果：

```
Student---Math--English--C Language--Average
NO  1    95.0    68.0    78.0    80.3
NO  2    65.0    77.0    88.0    76.7
NO  3    94.0    82.0    73.0    83.0
```

与一维数组一样，引用二维数组元素时，注意行下标和列下标都不要越界。

5.3.3 用循环结构存取二维数组

for 循环和一维数组的使用紧密结合，而嵌套的 for 循环则是处理多维数组的理想选择。当然用 while 和 do…while 来实现也是可以的。

【例 5-15】 单位矩阵初始化。(nbuoj1140)

输入一个整数 n，输出 n×n 的单位矩阵。单位矩阵在主对角线上的值为 1，而其他位置的值为 0，并且主对角线上的行、列下标是一样的。假设数组维数不超过 100。

```
#include <stdio.h>
#define max 100                          //最大维数
int main()
{
  int i,j;
  int a[max][max];
  int n;                                 //n 表示实际维数
  scanf("%d",&n);                        //输入一个整数表示当前需要的实际维数
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      if(i==j) a[i][j]=1;                //生成主对角线上元素
      else     a[i][j]=0;                //其他元素为 0
  for(i=0;i<n;i++)                       //输出单位矩阵
  {
    for(j=0;j<n;j++)                     //输出第 i 行的所有元素
      printf("%d ",a[i][j]);
    printf("\n");                        //一行输出结束后需换行
  }
  return 0;
}
```

运行结果：

```
4↵
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

【例 5-16】 杨辉三角形。(nbuoj1165)

输入一个整数 n(1≤n≤15)，输出如下形式的 n 行的杨辉三角形，此处显示的是 n 为 5

的杨辉三角形。

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

已知杨辉三角形有如下的规律。

(1) 每行首尾的元素都为1,即对角线上的元素和第0列元素都是1。

(2) 其余元素的值为上一行的同列元素与前一列元素之和。如第5行第3个数为6,它是第4行第3个数3和第4行第2个数3的和。因此,可以使用以下公式计算元素的值。

```
a[i][j] = a[i - 1][j] + a[i - 1][j - 1]
```

具体代码如下。

```
#include < stdio.h >
#define N 15                              //最大行数
int main()
{
  int a[N][N],n;
  int i,j;
  scanf(" % d",&n);                       //实际行数
  for(i = 0;i < n;i++)                    //设置每行的首尾为1
  {
    a[i][0] = 1;
      a[i][i] = 1;
  }
  for(i = 2;i < n;i++)                    //其余元素
    for(j = 1;j < i;j++)
    a[i][j] = a[i - 1][j] + a[i - 1][j - 1];
  for(i = 0;i < n;i++)
  {
    for(j = 0;j < = i;j++)
    printf(" % d ",a[i][j]);
    printf("\n");
  }
  return 0;
}
```

运行结果:

```
5↵
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

字符数组和字符串

5.4 字符数组和字符串

实际应用中会涉及大量的文本，文本处理的对象是字符串，例如，人的姓名(Frank)、单位的名称(Ningbo University)、住址信息(Zhongshan road)等。C 语言中只有字符类型，没有字符串类型，用字符类型定义的字符变量只能存储单个字符而不能存储多个字符，因此字符串的存取需要用字符数组来实现。

5.4.1 字符数组定义

字符数组的数据类型为 char，其中存放的元素都是字符，一个元素对应一个字符。字符数组跟普通的数组一样，可以是一维的，也可以是多维的。

一维字符数组的定义方式见表 5-4。

表 5-4 一维字符数组的定义形式

语　法	类型标识符 字符数组名[数组长度];
示　例	char str[10];
说　明	str 数组可包含 10 个字符

表 5-4 的样例中定义 str 为字符数组，包含 10 个元素，最多可以存放 10 个字符。下面的 10 个赋值语句依次给 10 个数组元素赋一个字符值。

```
str[0] = 'G';  str[1] = 'o';  str[2] = 'o';  str[3] = 'd';  str[4] = ' ';
str[5] = 'N';  str[6] = 'i';  str[7] = 'g';  str[8] = 'h';  str[9] = 't';
```

赋值后的数组状态见图 5-17。

	str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]	str[9]
str数组	G	o	o	d		N	i	g	h	t

图 5-17 赋值后的一维数组状态

字符数组也可以是二维的或者多维的。二维字符数组的定义方式见表 5-5。

表 5-5 二维字符数组的定义形式

语　法	类型标识符 字符数组名[整型常量表达式 1][整型常量表达式 2];
示　例	char name[5][10];
说　明	name 数组共 5 行 10 列。可用来存储 5 个人的姓名，每个姓名的长度不超过 10 个字符

5.4.2 字符数组初始化

在定义字符数组时，可对数组元素进行初始化，有以下两种初始化方法。

1. 初始化列表

这种方法比较容易理解，相当于给出一个初始化列表，把字符常量依次放在一对大括号内，逐个赋给对应的字符数组元素。

【例 5-17】 用初始化列表对一维字符数组逐个元素初始化。

```
#include <stdio.h>
int main()
{
  char str[5] = {'F','r','a','n','k'};     //将字符常量依次赋给字符数组中的各元素
  int i;
  for(i = 0;i < 5;i++)
    printf("%c",str[i]);                   //逐个输出字符数组的元素
  printf("\n");
  return 0;
}
```

运行结果：

```
Frank
```

- 单个字符的输入输出，用格式控制符%c。
- 字符常量外面的单引号只是在书写代码时起到界定符的作用，在存储或输出时都不会显示。

本例的字符数组 str 有 5 个数组元素，大括号内也有 5 个字符常量。初始化后，str 数组的存储情况如图 5-18 所示，5 个字符常量依次赋给了 str[0]到 str[4]的 5 个数组元素。

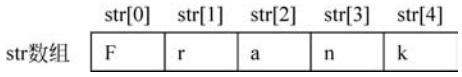


图 5-18 逐个字符赋给数组中的各元素

用初始化列表时要注意以下三种情况。

(1) 若大括号内的初值个数(即字符个数)大于数组长度，编译时将出现语法错误。

(2) 若大括号内的初值个数小于数组长度，则将这些字符依次赋给数组中前面的那些元素，其他元素自动赋空字符'\0'(空字符在字符数组中有特殊含义，作为字符串结束标志)。例如，将上例中的初始化语句改为：

```
char str[8] = {'F','r','a','n','k'};
```

则 str 存储情况如图 5-19 所示，共有 8 个数组元素，前 5 个数组元素存储了有效的字符内容，后 3 个数组元素自动为空字符。

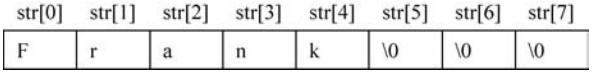


图 5-19 字符个数小于数组长度的初始化

(3) 若大括号内的初值个数与预定的数组长度相同，则在定义时可以省略数组长度，系统会自动根据初值字符的个数来决定数组大小，例如：

```
char str[ ] = {'F','r','a','n','k'};
```

此时系统将数组 str 的长度自动定义为 5。

可以定义和初始化一个二维字符数组，见例 5-18。

【例 5-18】 用初始化列表对二维字符数组逐个元素初始化。

```
#include<stdio.h>
int main()
{
  char tri[3][5]={{' ',' ','A'},
                  {' ','A','A','A'},
                  {'A','A','A','A','A'}};
  int i,j;
  for(i=0;i<3;i++)
  {
    for(j=0;j<5;j++)
      printf("%c",tri[i][j]);
    printf("\n");
  }
  return 0;
}
```

运行结果：

```
  A
 AAA
AAAAA
```

本例通过对二维字符数组的初始化，最后输出一个由大写字母 A 组成的三角形。

2. 用字符串常量初始化

用逐个字符赋给数组中的各元素初始化字符数组的形式，虽然比较清晰，但实际使用时很不方便，每个字符常量书写时都要加单引号，是一件比较烦琐的事情。C 语言中对于字符数组的初始化，除了逐个元素赋值以外，还允许用字符串常量进行初始化。

字符串常量是由双引号括起来的字符序列，如"Hello"或空字符串""。无论双引号内是否包含字符，包含多少个字符，都代表一个字符串常量。

为了便于确定字符串长度，C 编译器会自动在字符串末尾添加一个“字符串结束标志”，即一个 ASCII 码值为 0 的空字符('\0')，空字符是一个不可显示的字符，只作为字符串结束的标志。

由于字符串常量具有以'\0'结尾的特性，因此当用字符串常量初始化字符数组时，就会出现特别的地方，见例 5-19。

【例 5-19】 用字符串常量对字符数组初始化。

```
#include<stdio.h>
int main()
{
  char str[6]={"Frank"};     //用字符串常量初始化字符数组
  int i;
    for(i=0;i<5;i++)
      printf("%c",str[i]);
  printf("\n");
  return 0;
}
```

运行结果：

```
Frank
```

本例用字符串常量"Frank"初始化字符数组，书写上简便了许多，而且赋值的内容也显得更加直观。

细心的读者可能会发现这里字符数组 str 的长度变成了 6，而不是 5。请注意，这决不是一个可有可无的改变，如果用字符串常量来初始化字符数组的话，字符数组的长度一定要至少比字符串的有效长度加 1。例如，"Frank"字符串常量的有效长度为 5，则存储它的字符数组的长度至少要为 6。

原因还是前面提到的字符串常量以'\0'结尾的这一特性。当以字符串常量对字符数组初始化时，系统会自动在最后面添加'\0'作为字符串的结束标志，此时 str 数组的存储情况将如图 5-20 所示。

	str[0]	str[1]	str[2]	str[3]	str[4]	str[5]
str数组	F	r	a	n	k	\0

图 5-20　用字符串常量初始化数组

其中，str[0]到 str[4]存储的是有效字符，另外还需要 str[5]来存储字符串结束标志，因此，虽然字符串"Frank"的有效字符只有 5 个，但是存储它的字符数组的长度至少需要 6 个。

如果用以下形式来初始化：

```
char str[5] = {"Frank"};  //错误
```

这是错误的，因为字符串"Frank"至少需要 6B 的存储单元，而数组的长度为 5，无法存储字符串结束标志，从而导致系统无法处理该字符串。

用字符串常量初始化字符数组的话，还有更简洁的书写形式，即可以省略大括号，直接写成如下形式。

```
char str[6] = "Frank";
```

- C 语言中，字符串以空字符'\0'作为结束标志，空字符在输出时不会被显示出来。
- C 语言并不要求所有的字符数组的最后一个字符都必须是'\0'，如“char str[5]={'F','r','a','n','k'};”也是正确的。但如果用字符串常量的形式初始化字符数组，系统必将在字符串后面自动加'\0'。
- 对于有效字符个数为 n 的字符串，其占用内存为 n+1 个字符所占空间大小。

【例 5-20】 计算字符串的有效长度，并输出该字符串。

```
#include <stdio.h>
int main()
{
  int i,len = 0;
  char str[20] = "Programming C";      //用字符串常量初始化字符数组
  for(i = 0;str[i]!= '\0';i++)         //若 str[i]不等于'\0'则继续循环
    len++;                             //计算字符串有效长度的计数器增 1
  printf("String is:");
  for(i = 0;i < len;i++)
```

```
    printf("%c",str[i]);
  printf("\nLength = %d\n",len);
  return 0;
}
```

运行结果：

```
String is:Programming C
Length = 13
```

由于字符串常量以'\0'结尾，因此在计算字符串有效长度时，先按照下标递增的顺序逐个处理元素，一旦遇到某个元素是'\0'，则说明字符串已经结束了。

字符串的有效长度和字符数组的长度是两回事，本例中字符数组的长度是 20，而字符串“Programming C”的有效长度只有 13。

在字符串的有效长度 len 已经确定的情况下，对字符串处理时可通过比较数组下标与 len 的大小来判断字符串是否结束，如本例采用的形式：

```
for(i = 0;i < len;i + + )
```

当然也可以继续用'\0'来控制循环，例如：

```
for(i = 0;str[i]!= '\0';i + + )
```

- 空字符'\0'不计入字符串的长度。
- 当遍历一个字符数组时，如果遇到空字符'\0'，就认为字符串结束。

5.4.3 字符数组的输入/输出

如果在定义字符数组的时候，没有进行初始化工作，可以在程序运行后通过输入语句实现为字符数组赋值的工作。

字符数组的输入/输出主要有三种方式。

1. 单个字符输入/输出

如果已知字符个数，可以用 scanf()/printf()函数按%c 格式控制单个字符的输入/输出，结合循环语句，就可以处理多个字符。

【例 5-21】 从键盘输入 5 个字符，存入数组中，然后再输出这 5 个字符。

```
#include <stdio.h>
int main()
{
  char s[5];
  int i;
  for(i = 0;i < 5;i++)                    //输入 5 个字符到 s 数组
    scanf("%c",&s[i]);
  for(i = 0;i < 5;i++)                    //输出 s 数组中的 5 个字符
      printf("%c",s[i]);
    printf("\n");
  return 0;
}
```

运行结果：

```
hello↵
hello
```


用 getchar()和 putchar()来处理单个字符也是可以的，如本例的输入/输出部分可改写为：

```
for(i=0;i<5;i++)          //输入 5 个字符到 s 数组
  s[i]=getchar();
for(i=0;i<5;i++)          //输出 s 数组中的 5 个字符
  putchar(s[i]);
```

通过控制个数来输入/输出单个字符的方法在实际使用中很不方便，容易出错，一般不建议采用这种方式。

2. 整个字符串输入/输出

可以用 scanf()/printf()函数按%s 格式控制整个字符串的输入/输出，但是要注意，scanf()函数接收的字符串中不可以包含空格。

【例 5-22】 从键盘输入一个字符串(不带空格)，统计其中的数字字符有多少个。

```
#include<stdio.h>
int main()
{
  char str[20];
  int i=0,count=0;
  scanf("%s",str);        //输入一行字符,不带空格,以回车结束
  while(str[i]!='\0')
  {
    if(str[i]>='0'&&str[i]<='9')
      count++;            //统计数字字符个数的计数器
    i++;
  }
  printf("String is:");
  printf("%s",str);       //整个字符串输出,到结束标志'\0'则认为字符串结束
  printf("\nDigit=%d\n",count);
  return 0;
}
```

运行结果：

```
Hello007↵
String is:Hello007
Digit=3
```

在语句“scanf("%s",str);”中，地址表部分是字符数组名，不需要再加取地址符号“&”，因为在 C 语言中数组名代表该数组的起始地址。

在语句“printf("%s",str);”中，输出项也是字符数组名，而不是某个数组元素的名字。

scanf()函数按%s 格式输入字符串时，如果以空格、回车、制表符(Tab)作为间隔符，则

不能得到完整的输入内容。见例 5-23。

【例 5-23】 字符串输入/输出。(nbuoj1088)

输入任意长度的字符串(小于 100 个字符),以换行结束,并输出。

```
#include<stdio.h>
int main()
{
  char str[100];
  scanf("%s",str);
  printf("%s",str);
  putchar('\n');
  return 0;
}
```

运行结果:

```
Hello Boy↵
Hello
```

程序运行时从键盘输入"Hello Boy",但实际上只接收空格前的"Hello"存入 str 数组,并在后面增加一个'\0',而把空格后面的内容"Boy"丢弃了,见图 5-21。

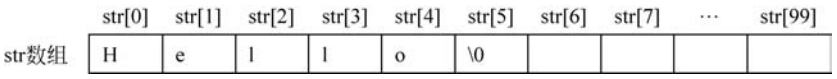


图 5-21 str 数组的存储情况

用 scanf()函数按%s 格式输入字符串时,遇到空格、回车、制表符,系统认为字符串输入结束。

本题用 scanf()按%s 格式来输入字符串,只能接收不带空格的字符串,如果要接收带空格的字符串则需要用到下面介绍的 gets()函数。

3. 用字符串处理函数 gets()和 puts()进行输入/输出

C 语言提供的字符串处理函数 gets()可以接收带空格的字符串。gets()函数的作用是从终端输入一个字符串到字符数组,其调用形式见表 5-6。

表 5-6 gets()函数的调用形式

语 法	示 例	说 明
gets(字符数组);	char str[100]; gets(str);	(1) 从终端输入一个字符串到字符数组,输入正确时,返回值为字符数组的起始地址;输入失败时,返回 NULL 指针。 (2) 输入的字符串以换行符'\n'为结束标记。在向字符数组赋值时,自动将'\n'转换成'\0',作为字符串的结束标记

gets()函数一次只能输入一个字符串,不能写成 gets(str1,str2)。

gets()函数接收的字符串中可以包含空格。

因此,例 5-23 的代码改写为:

```
char str[100];
gets(str);   //用 gets()函数可读取带空格的字符串
```

```
printf("%s",str);
putchar('\n');
```

就可以接收带空格的字符串。

puts()函数的作用是将一个字符串(以'\0'结束的字符序列)输出到终端,其调用形式见表5-7。

表5-7 puts()函数的调用形式

语 法	示 例	说 明
puts(字符数组);	char str[100]; … puts(str);	(1) 将一个字符串(以'\0'结束的字符序列)输出到终端。 (2) 在输出时将字符串结束标记'\0'转换成'\n',即输出字符串后自动换行

用puts()函数输出的字符串中可以包含转义字符。例如:

```
char str[30] = "Hello\nNice to meet you";
```

输出为:

```
Hello
Nice to meet you
```

【例5-24】 字符变换。(nbuoj1057)

输入任意一个字符串(长度小于等于1000),将字符串中的大写英文字母转换成对应的小写英文字母,而将小写英文字母转换成对应的大写英文字母,其余字符不变。输出转换后的字符串。

```
#include<stdio.h>
#include<string.h>
int main()
{
  char str[1001];
  int i=0;
  gets(str);        //输入一个字符串,以换行结束
  while(str[i]!='\0')
  {
    if(str[i]>='a'&&str[i]<='z')
      str[i]=str[i]-32;
    else if(str[i]>='A'&&str[i]<='Z')
      str[i]=str[i]+32;
    i++;
  }
  puts(str);        //输出字符串
  return 0;
}
```

运行结果:

```
hELLO bOY! ↵
Hello Boy!
```

puts()函数的作用等同于 printf("%s\n",字符数组名)。不过 puts()函数一次只能输出一个字符串,而 printf()函数可以一次输出多个字符串,如 printf("%s %s\n",字符数组名 1,字符数组名 2)。

gets()函数在读取字符串时将删除结尾的换行符'\n',而 puts()函数在写字符串时将在结尾添加一个换行符。

5.4.4 字符数组输入/输出的异常情况

与数值型数组一样,如果在定义字符数组后没有给数组元素赋值,数组元素的值是一个不确定的数据。

【例 5-25】 数组元素没有正确赋值时的输出异常。

```
#include <stdio.h>
int main()
{
  char s[10];                          //定义数组时没有初始化
  int i;
  for(i=0;i<5;i++)                     //仅对前 5 个数组元素赋值
    scanf("%c",&s[i]);
  puts(s); //或 printf("%s",s);        //试图输出整个字符串
  return 0;
}
```

运行结果:

```
Frank ↵
Frank 烫烫烫蘘 □
```

本例在定义 s 数组时没有对其初始化,因此 10 个数组元素的值都是不确定的。在循环语句中只对前 5 个数组元素进行了读入,数组元素的存储示意图见图 5-22,由于后 5 个数组元素没有读入任何数据,其值是不确定的。

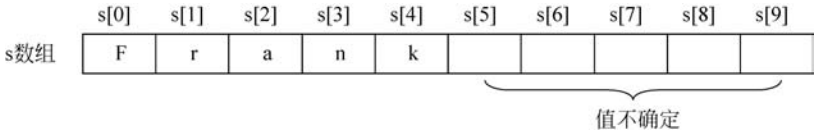


图 5-22 对字符数组不正确赋值时的存储示意图

如果本例的输入/输出采用如下形式:

```
for(i=0;i<5;i++)                       //仅对前 5 个数组元素赋值
  scanf("%c",&s[i]);
for(i=0;i<5;i++)                       //仅输出前 5 个数组元素
  printf("%c",s[i]);
```

那么在输入 Frank 后能正确输出 Frank,即仅对前 5 个数组元素操作。但是本例在输出时用了 put(s)(或 printf("%s",s)),由于 puts()函数(或 printf 按%s 格式)在输出时以'\0'作为字符串结束标志,而通过图 5-22 可知,后 5 个数组元素的值不确定,没有字符串结束标志'\0',因此输出时会出现乱码。

字符串往往在整体出现时才有意义，如人的名字“Frank”，打招呼的语句“Hello Boy”，课程名字“Programming C”等，因此建议在输入/输出时采用整体输入/输出的方式，而不要采用%c 单个字符处理的方式。以下两种方式都可对字符串进行整体操作。

```
char s[10];
scanf("%s",s);          //整体输入字符串,不可接收带空格字符串
printf("%s",s);         //整体输出字符串
```

或者

```
char s[10];
gets(s);                //整体输入字符串,可接收带空格字符串
puts(s);                //整体输出字符串
```

用 scanf("%s",s)输入字符串，按空格键或回车键后，系统会将空格或回车之前的内容赋给数组元素，然后自动在末尾加字符串结束标志'\0'。同样，用 gets(s)输入字符串，按回车键后，系统会将回车之前的内容赋给数组元素，然后自动在末尾加字符串结束标志'\0'。因此在有效字符的后面出现了字符串结束标志'\0'，见图 5-23，此时用 puts 或%s 整体输出时就不会出现乱码。

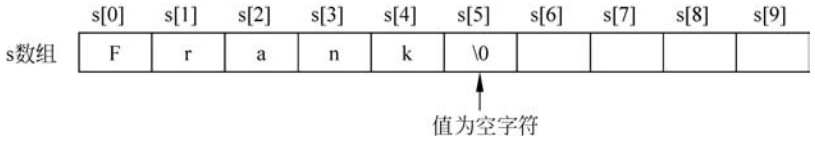


图 5-23　对字符数组正确赋值后的存储示意图

常用字符串处理函数

5.4.5　字符串处理函数

C 语言的库函数中提供了丰富的字符串处理函数，除了上面已介绍的 gets()和 puts()外，还有一些其他的字符串处理函数，这些函数在使用时需要加上头文件 string.h，即：

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

1. 字符串长度函数 strlen()

前面的例子中实现了测试字符串的有效长度，C 语言库函数也提供了测字符串长度的函数，即 strlen()函数，其调用形式见表 5-8。

表 5-8　strlen()函数的调用形式

语　法	示　例	说　明

strlen(字符数组名)	char st[20]="hello boy";
	int len;
	len=strlen(st);
	printf("%d\n",len);
		strlen()函数的返回值是字符串的实际长度，不包括'\0'。如示例中 len 获得的值为 9，不是 10，也不是 20

2. 字符串连接函数 strcat()

strcat()函数的作用是将两个字符数组中的字符串连接起来，组成一个新的字符串。其

调用形式见表 5-9。

表 5-9　strcat()函数的调用形式

语　　法	示　　例	说　　明
strcat(字符数组 1,字符数组 2);	strcat(str1,str2); (假设 str1,str2 是数组名)	(1) 将字符串 2 连接到字符串 1 的后面，结果放到字符数组 1 中，函数调用后返回字符数组 1 的地址。 (2) 字符数组 1 必须足够大，以便容纳连接后的新字符串

【例 5-26】 输入两个字符串，然后把它们连接起来。

```
#include<stdio.h>
#include<string.h>
int main()
{
  char str1[80],str2[30];
  int i=0;
  printf("Enter the first string:");
  gets(str1);
  printf("Enter the second string:");
  gets(str2);
  strcat(str1,str2);   //用 strcat 函数实现两个字符串连接,结果放在第一个字符数组
  printf("New string:");
  puts(str1);          //输出第一个字符数组内容,即连接后的新字符串
  return 0;
}
```

运行结果：

```
Enter the first string:Hello↵
Enter the second string:Boy↵
New string:HelloBoy
```

- 连接前两个字符串的后面都有一个'\0'，连接后原来的第一个字符串末尾的'\0'被覆盖，只在新形成的字符串末尾加一个'\0'。
- 连接两个字符串时不会自动添加其他字符，比如输入第一个串 Hello 后面没有空格，则连接后内容 HelloBoy 在 Hello 后面也不会有空格。

可以自己编码来实现 strcat()函数的功能。见例 5-27。

【例 5-27】 编程实现 strcat()函数的功能。

```
#include<stdio.h>
int main()
{
  char s[15],t[10];
  int i=0,j=0;
  printf("Enter the first string:");
  gets(s);                    //输入第一个字符串存入 s 数组
  printf("Enter the second string:");
```

```
    gets(t);                    //输入第二个字符串存入 t 数组
    while(s[i]!= '\0') i++;     //搜索到字符串 1 的末尾
    while(t[j]!= '\0')          //若字符串 2 未遇到'\0',则执行连接操作
    {
      s[i] = t[j];              //将字符串 2 的内容逐一复制到字符数组 1
        i++;
      j++;
    }
    s[i] = '\0';                //在新的字符串末尾添加'\0'作为结束标记
    printf("New String is:");
    puts(s);                    //输出连接后形成的新的字符串
    return 0;
}
```

运行结果：

```
Enter the First String:Good.↵
Enter the second string:Thanks!↵
New String: Good.Thanks!
```

连接前，s 数组中存放“Good.”，t 数组中存放“Thanks!”，见图 5-24(a)。代码中第一个 while 循环对 s 数组搜索，遇到'\0'停下，此时已到字符串“Good.”的末尾，i 的值为 5。第二个 while 循环实施连接操作，不断执行“s[i]＝t[j]”，直到第二个字符串遇到'\0'停下，见图 5-24(b)。

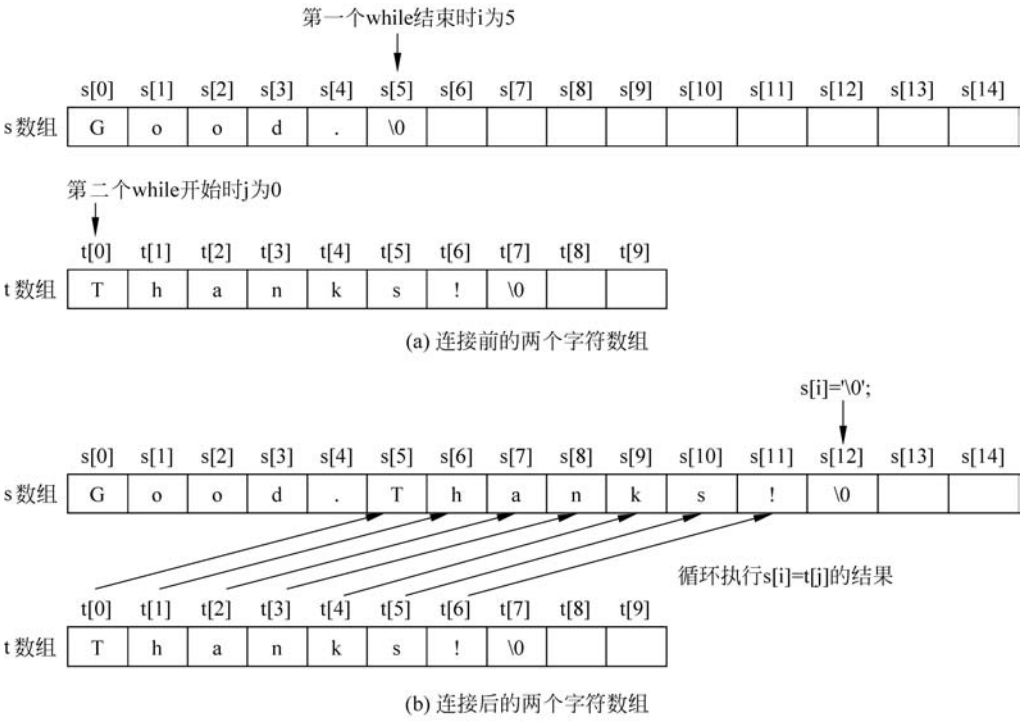


图 5-24　连接前后两个字符数组的存储示意图

- 程序最后的“s[i]='\0';”不能省略，否则 s 数组中的字符串没有结束标志，会造成输出错误。
- 连接的是两个字符串，而不是两个字符数组。

3. 字符串复制函数 strcpy()

C 语言中，不能像基本变量一样直接用赋值号进行字符串之间的复制，需要使用循环逐个对字符进行复制，并在最后加上字符串结束标志'\0'。C 语言提供的 strcpy()函数可以实现这个功能，将字符串 2 的内容复制到字符数组 1 中，其调用形式见表 5-10。

表 5-10　strcpy()函数的调用形式

语　法	示　例	说　明
strcpy(字符数组 1,字符串 2);	strcpy(str1,str2); 或 strcpy(str1,字符串常量) (假设 str1,str2 是数组名)	(1) 字符数组 1 必须足够大，以便容纳被复制的内容，至少不能小于字符串 2 的长度。 (2) 字符数组 1 必须写成数组名形式，字符串 2 可以是字符数组名，也可以是字符串常量。 (3) 字符串 2 后面的'\0'也一同被复制到字符数组 1

例如：

```
char str1[80],str2[20] = "Good luck! "
strcpy(str1,str2);
```

则 str1 数组中存放的内容也将是"Good luck! "。

注意：C 语言中不能用赋值号将一个字符串或字符数组直接复制给另一个字符数组，例如，下面的赋值语句是错误的。

```
char str1[20],str2[10] = "Bye Bye! "
str1 = "Bye Bye! "   //错误
str1 = str2;         //错误
```

因为数组名是数组的首地址，是一个常量，不可以给常量赋值。

可以自己编码来实现 strcpy()函数的功能，见例 5-28。

【例 5-28】 编程实现 strcpy()函数的功能。

```
#include<stdio.h>
int main()
{
  char s[10],t[10];
  int i=0,j=0;
  printf("T String is:");
  gets(t);              //输入第二个字符串
  while(t[i]!='\0')     //若字符串 2 未遇到'\0',则执行复制操作
  {
    s[i]=t[i];          //将 t 数组的内容逐一复制到 s 数组
    i++;
  }
  s[i]='\0';            //在新的字符串末尾添加'\0'作为结束标记
```

```
    printf("S String is:");
    puts(s);              //输出 s 数组的内容
    return 0;
}
```

运行结果：

```
T String is:Hello Boy↵
S String is:Hello Boy
```

4. 字符串比较函数 strcmp()

strcmp()函数的作用是对两个字符串进行比较，其用法见表 5-11。

表 5-11　strcmp()函数的调用形式

语　法	示　例	说　明
strcmp(字符串 1,字符串 2);	strcmp(str1,str2); (假设 str1,str2 是数组名)	对两个字符串从左到右逐个字符比较(按 ASCII 码值大小比较)，直到出现不同字符或遇到'\0'。比较结果返回一个函数值。 (1) 若全部字符相同，则认为两个字符串相等，函数返回 0。 (2) 若出现不相同字符，则以第 1 对不相同的字符的比较结果为准，若字符串 1 大于字符串 2，函数返回一个正整数；若字符串 1 小于字符串 2，函数返回一个负整数

例如：

```
int k;
k = strcmp("Hi","Hello");
```

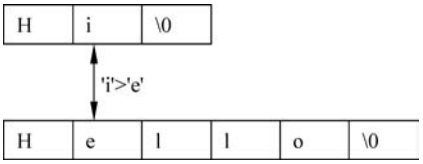


图 5-25　字符串比大小的示意图

本例中的 k 是一个正整数，说明字符串"Hi"大于字符串"Hello"，因为"Hi"中的第 2 个字符是'i'，"Hello"中的第 2 个字符是'e'，显然字母'i'的 ASCII 码值大于字母'e'的 ASCII 码值，见图 5-25。

字符串比大小，比的是对应字符 ASCII 码值的大小，而不是比两个字符串的长度。

字符串比大小的功能在类似字符串排序的过程中非常有用，见例 5-29。

【例 5-29】 输入两个人的姓名，按字典顺序进行输出。

```
#include<stdio.h>
#include<string.h>
int main()
{
  char s1[20],s2[20];
  printf("Enter first name:");
  gets(s1);
  printf("Enter second Name:");
```

```
  gets(s2);
  if(strcmp(s1,s2)< 0||strcmp(s1,s2) == 0)   //比较两个字符串大小
    printf(" % s  % s\n",s1,s2);              //按字典序输出
  else
    printf(" % s  % s\n",s2,s1);
  return 0;
}
```

运行结果：

```
Input first name:Susan↵
Input second name:Anney↵
Anney Susan
```

不能用关系运算符比较两个字符串，假设 str1 和 str2 是两个字符串，则以下写法是错误的。

```
if(str1 = = str2) printf("Equal\n");                    //写法错误
```

应该写成：

```
if(strcmp(str1,str2) = = 0) printf("Equal\n");          //写法正确
```

5.5 高精度加法

高精度计算，是指参与运算的数大大超出标准数据类型所能表示的范围的运算。例如，两个 1000 位的数相加，或者求斐波那契数列的前 1000 位等，用已知的数据类型无法正确实现这些运算。

在 C 语言中，可以用数组模拟大数的运算，数组的每个元素代表大数的某一位，然后按位处理进位、借位问题，处理完以后再将该数组用规定的格式输出即可。基本步骤如下。

(1) 读入数据：建议用字符串方式读入，优点是字符串输入较方便。

(2) 存储数据：建议转换为整型数组存储，这样计算的时候比较方便。

(3) 实施运算：模拟数学计算，从个位开始向高位逐位计算。

(4) 输出结果：用规定格式输出计算结果。

【例 5-30】 大数的输入/输出。(nbuoj1909)

输入一个不超过 200 位的正整数，并输出该数。

```
#include < stdio.h >
#include < string.h >
int main()
{
  int i,len,a[201];
  char s[201];
  gets(s);                                          //用字符串形式读入大数
  len = strlen(s) - 1;
  for (i = 0; i < = len; i++)a[i] = s[i] - '0';     //将字符数组 s 的每一位转换成整数
  printf("The Number:");
```

```
    for (i = 0; i <= len; i++) printf("%d",a[i]);      //用整数形式输出数组 a 的每一位
  printf("\n");
  return 0;
}
```

运行结果：

```
111222333444555666777888999 ↵
The Number: 111222333444555666777888999
```

本例如果用 int 型处理会溢出，用 double 型处理则会失真。

用数组可以实现大数的输入和输出，那么如何实现两个大整数的加法呢？首先以 9768＋523＝10291 为例，来看一个加法运算的实现过程。将这个过程写成竖式，见图 5-26。

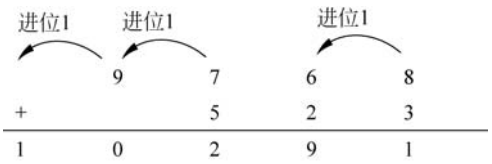


图 5-26　加法的竖式

加法需要尾对齐，从个位开始模拟。

个位：8＋3＝11，个位取 1，向高位进位 1。

十位：6＋2＋1(低位进位)＝9，十位取 9，进位为 0。

百位：7＋5＝12，百位取 2，向高位进位 1。

千位：9＋1(低位进位)＝10，千位取 0，向高位进位 1。

万位：万位为 1，来自低位的进位。

在加法的每个步骤都要考虑是否有来自低位的进位，以及是否需要向高位进位。

由于加法需要个位对齐，如果两个数直接从左到右存入数组的话，会错位，给计算带来麻烦，因此考虑将数据逆序存放到数组中，即从左到右依次存放个位、十位、百位、……，见图 5-27。计算时，将两个数组中的元素逐位相加即可。输出时，要注意再次实施逆序操作，按数据的正常格式输出计算结果。

图 5-27　大数相加时数据在数组中的存放形式

假设两个大数已分别存放在数组 a 和 b 中，相加的结果存回数组 a，则两个大数加法的核心部分的参考程序如下。

```
c = 0;                          //进位初值 0
for(i = 0;i < len;i++)
{
  a[i] = a[i] + b[i] + c;       //逐位相加，并加上进位，存入 a[i]
```

```
  if(a[i]>=10)                //若大于 10 则进位
    {
      a[i]=a[i]%10;           //保留小于 10 的数值
      c=1;                    //进位 1
    }
    else c=0;                 //若小于 10 则无进位
}
if(c>0) {a[len]=c;len++;}  //最后一次计算还有进位
```

【例 5-31】 大数相加。(nbuoj2830)

输入任意长度(不超过 1000 位)的两个正整数,计算两数相加的结果并输出。

```
#include<stdio.h>
#include<string.h>
int main()
{
  char sa[1002],sb[1002];
  int a[1002]={0},b[1002]={0};
  int i,j,lensa,lensb,len,c;
  gets(sa); gets(sb);                    //用字符串形式读入两个大数
  lensa=strlen(sa);lensb=strlen(sb);//测出每个字符串的长度
  j=0;
  for(i=lensa-1;i>=0;i--)          //将第一个数逆序转换为整型数组形式,即低位在前
          a[j++]=sa[i]-'0';
  j=0;
  for(i=lensb-1;i>=0;i--)          //将第二个数逆序转换为整型数组形式,即低位在前
          b[j++]=sb[i]-'0';
  if(lensa>=lensb) len=lensa;      //len 取两数中最大的位数
      else len=lensb;
  //进行两个大数的加法
  c=0;                                   //进位初值 0
  for(i=0;i<len;i++)
  {
    a[i]=a[i]+b[i]+c;                    //逐位相加,并加上进位,存入 a[i]
    if(a[i]>=10)                         //大于 10 则进位
        {a[i]=a[i]%10;c=1;}
    else c=0;
  }
  if(c>0) {a[len]=c;len++;}              //最后一次计算还有进位
  //逆序输出
  for(i=len-1;i>=0;i--)
    printf("%d",a[i]);
  printf("\n");
  return 0;
}
```

运行结果:

```
111222333444555666777888999↵
1↵
111222333444555666777889000
```

两个数都是以字符串形式输入的，分别存入字符数组 sa 和 sb。接着测出每个字符串的长度(也就是该数的位数)，因为字符串不能直接进行运算，所以将字符串形式的数据逆序转换为整型数组的形式(例如输入两个数为 1234 和 5678，则转换为整型数组形式存储为 4321 和 8765，让低位在前，使运算更方便)，然后进行运算。加法计算完成后，在数组 a 中低位在前，高位在后，因此在输出时，需要再做一次逆序操作，使高位在前，低位在后。

5.6 实例研究

5.6.1 统计单词数

【例 5-32】 统计单词数。(nbuoj1176)

输入一行由英文字母、数字和空格组成的字符串，遇到换行符时表示输入结束。单词间以空格分隔，可能有多个空格。文章最多由 1000 个字符组成。试统计其中的单词个数并输出。

本题的关键是如何判断出现了新的单词。判断是否有新单词，可以由是否有空格来决定(一行开头的空格不算，连续出现的多个空格记为 1 次)。如果当前字符为空格，说明没有新单词出现。如果当前字符非空格，而它前面的字符为空格，说明“新的单词开始了”。如果当前字符非空格，而它前面的字符也是非空格，说明仍然是前面那个单词的延续，未出现新单词。一个基本的流程图见图 5-28。

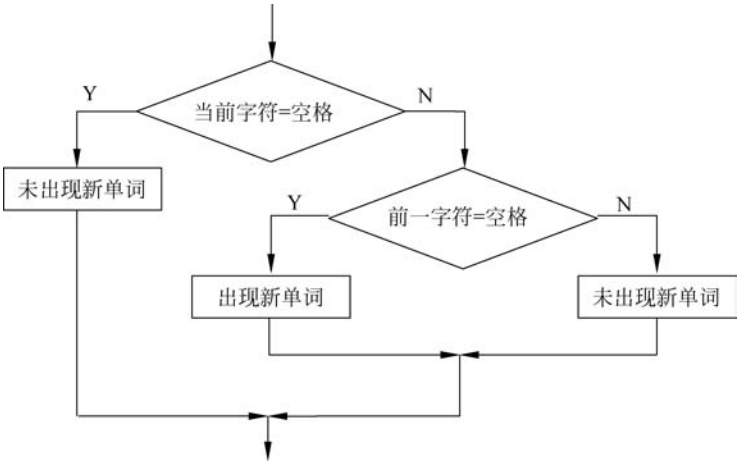


图 5-28 “统计单词数”的流程图

假设：用变量 is_word=0 表示前一字符为空格，is_word 初值为 0，is_word=1 表示前一字符不是空格。用变量 num 记录单词数，初值为 0。

则具体操作如下：如果当前字符是空格，说明未出现新单词，此时使 is_word 为 0，num 保持原值不变。如果当前字符不是空格，而前一个字符是空格(is_word=0)，说明出现新单词，此时使 is_word=1，num 加 1；如果当前字符不是空格，前一字符也不是空格(is_word=1)，说明未出现新单词，维持 is_word=1 不变，并且 num 保持不变。

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

```
int main()
{
  char str[1001];
  int i,num = 0,is_word = 0;
  char ch;
  gets(str);                    //读入一行字符
  for(i = 0;str[i]!= '\0';i++)
  {
    if(str[i] == ' ')           //若当前字符是空格,说明未出现新单词
      is_word = 0;              //使 is_word = 0,num 不变
    else                        //若当前字符不是空格
      if(is_word == 0)          //若前一字符是空格
      {                         //说明出现新单词
        is_word = 1;
        num++;                  //num 加 1,说明出现新单词
      }
  //若当前字符不是空格,前一字符也不是空格,说明未出现新词,is_word 和 num 都不变
  }
  printf(" % d\n",num);
  return 0;
}
```

运行结果：

```
He comes from Thailand↵
4
```

5.6.2 成绩管理

【例 5-33】 输入 n 名学生的学号、姓名以及 c 门课的成绩,并输出这些信息。其中,n 和 c 也是从键盘输入的。

本题定义三个二维数组：二维字符数组 id 存储学生的学号,二维字符数组 name 存储学生的姓名,二维浮点型数组 score 存储每位同学的若干门课的成绩。

```
#include < stdio.h >
#define S_NUM 100                       //假设学生最大数 100
#define C_NUM 10                        //假设课程最大数 10
int main()
{
  int i,j;
  char id[S_NUM][10];                   //学号
  char name[S_NUM][20];                 //姓名
  double score[S_NUM][C_NUM];           //课程成绩
  int n,c;                              //实际学生数及课程门数
  printf("Enter student number:\n");
  scanf(" % d",&n);                     //输入学生实际人数
  printf("Enter course number:\n");
  scanf(" % d",&c);                     //输入课程实际数量
  for(i = 0;i < n;i++)
  {
```

```
        printf("-------------------------\n");
        printf("Student %d's ID:",i+1);
        scanf("%s",id[i]);                    //输入学号
        getchar();                            //抵消上一个输入的回车符
        printf("Student %d's Name:",i+1);
        gets(name[i]);                        //输入姓名
        printf("Student %d's %d Scores:",i+1,c);
        for(j=0;j<c;j++)
          scanf("%lf",&score[i][j]);          //输入成绩
    }
    printf("------------ Information -------------\n");
    for(i=0;i<n;i++)
    {
      printf("%-10s%-20s",id[i],name[i]);
      for(j=0;j<c;j++)
        printf("%5.1f",score[i][j]);
      printf("\n");
    }
    return 0;
}
```

运行结果：

```
Enter student number:
2↵
Enter course numbet
3↵
-------------------------
Student 1's ID:1963001↵
Student 1's Name:Chen lin↵
Student 1's 3 Scores:90.5 90 90↵
-------------------------
Student 2's ID:1963002↵
Student 2's Name:Li ning↵
Student 2's 3 Scores:80 80 80.5↵
------------ Information -------------
1963001   Chen lin     90.5 90.0 90.0
1963002   Li ning      80.0 80.0 80.5
```

本题有一个二维字符数组name，存储每个学生的姓名。图5-29展示了name数组的存储情况，数组的大小为100×20，即有100行20列，可存储100个学生的姓名，每个姓名的最大长度为19列（name数组每行可容纳20个字符，但字符串结束标志'\0'要占用一个位置，因此姓名最大长度为19列）。实际使用时，不一定每次都要存储100个学生的信息，因此本例输入一个实际学生的数目保存到变量n中，即只使用了二维数组name的部分元素。

如前所述，可以把name[0]，name[1]，…，name[i]，…看作若干个一维字符数组（它们各有20个元素），可以把它们如同一维字符数组那样处理，在循环体中用“gets(name[i]);”读入每个学生的姓名。

name[0]	C	h	e	n		l	i	n	\0	…
name[1]	L	i		n	i	n	g	\0		…
…										…
name[i]										…
…										…
name[n-1]										…
…										…
name[99]										…

图 5-29　二维字符数组 name

5.6.3　城市名排序

【例 5-34】 城市名排序。(nbuoj1185)

输入 n 个城市的名称,进行升序排序并输出。先输入一个整数 n,表示有 n 个城市,n 不超过 100。接着输入 n 个字符串,每个字符串代表一个城市名,一个字符串内部不包含空格,字符串长度不超过 100。

```
#include<stdio.h>
#include<string.h>
int main()
{
  int n,i,j,min;
  char city[101][101],temp[101]; //temp 数组可临时保存一个字符串
  scanf("%d",&n);getchar();
  for(i=0;i<n;i++)
    gets(city[i]);                    //输入 n 个城市名称
  for(i=0;i<n-1;i++)                  //选择排序
  {  min=i;
     for(j=i+1;j<n;j++)
        if(strcmp(city[min],city[j])>0) min=j;
     if(min!=i)
        {strcpy(temp,city[i]);strcpy(city[i],city[min]);strcpy(city[min],temp);}
  }
  for(i=0;i<n;i++)
    puts(city[i]);                    //输出排序后的城市名称
  return 0;
}
```

运行结果:

```
3↵
Ningbo↵
Hangzhou↵
Shanghai↵
Hangzhou
Ningbo
Shanghai
```

本题用二维字符数组 city 来存储若干个城市名称，存储示意图见图 5-30。可以用 city[i]来表示某个城市(整体处理表示城市名的字符串)，在选择排序中用形如“strcmp(city[min],city[j])”的语句对两个字符串比较大小，用形如“strcpy(city[i],city[min])”的语句实现字符串的复制。

city[0]	N	i	n	g	b	o	\0			…
city[1]	H	a	n	g	z	h	o	u	\0	…
…	S	h	a	n	g	h	a	i	\0	…
city[i]										…
…										…
city[n-1]										…
…										…
city[100]										…

图 5-30　二维字符数组 city 表示城市名

5.6.4　扑克游戏

【例 5-35】 扑克游戏。模拟扑克游戏中的发牌过程，当用户输入需要的牌的张数时，就随机发给用户对应数目的牌。显示用户手上牌的花色和牌点。

本题可定义两个数组分别表示牌的花色和点数，其中表示花色的数组中，用字母'c'表示 clubs(梅花)，用字母'd'表示 diamonds(方块)，字母'h'表示 hearts(红桃)，字母's'表示 spades(黑桃)。

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define SUITS 4                              //牌的花色有 4 种
#define RANKS 13                             //每种花色有 13 张牌
int main()
{
  int in_hand[SUITS][RANKS]={0};             //记录每张纸牌是否被发过,初值为 0
  char rank_code[]={'2','3','4','5','6','7','8','9','T','J','Q','K','A'};   //13 张牌的牌点
  char suit_code[]={'c','d','h','s'};        //牌的 4 种花色
  int num;                                   //玩家手头牌的张数
  int suit,rank;                             //每次抽取的牌的花色和牌点
  int count=0;                               //控制每行输出几组牌
  srand((unsigned)time(NULL));
  printf("Enter number of cards:\n");
  scanf("%d",&num);                          //输入需要发牌的张数
  printf("----------------Your Cards----------------\n");
  while(num>0)
  {
    suit=rand()%SUITS;                       //随机产生一种花色
    rank=rand()%RANKS;                       //随机产生一个牌点
    if(!in_hand[suit][rank])                 //判断该牌是否被发过,没被发过的牌本次可以发出
    {
      in_hand[suit][rank]=1;
```

```
            num--;
            printf("(%c,%c) ",suit_code[suit],rank_code[rank]);
            count++;
            if(count%5==0) putchar('\n');
        }
    }
    putchar('\n');
    return 0;
}
```

运行结果：

```
Enter number of cards:
12↵
----------------Your Cards------------------
(s,2) (d,Q) (h,J) (c,5) (d,7)
(h,2) (h,4) (d,5) (c,8) (d,J)
(c,7) (s,K)
```

为了避免两次发同样的牌，需要记录已经发过的牌。本例定义了二维数组 in_hand 来执行这个操作，该数组有 4 行 13 列，每行表示一种花色，每列表示一种牌点。在程序开始时，该数组的所有元素都置 0，表示都没有被发出去过。每次随机抽取一张纸牌后，对应的元素值就变为 1，则下次就不会被再抽取了。

5.7 习　　题

5.7.1 选择题

1. C 语言中，以下关于数组的描述正确的是(　　)。
   A. 数组大小固定，但是可以有不同类型的数组元素
   B. 数组大小可变，但是所有数组元素的类型必须相同
   C. 数组大小固定，所有元素的类型必须相同
   D. 数组大小可变，可以有不同类型的数组元素
2. 若有定义“int score[10];”，则对 score 数组中的元素的正确引用是(　　)。
   A. score(1)　　B. score[10]　　C. score[6.0]　　D. score[0]
3. 若有定义“int a[]={5,4,3,2,1},i=4;”，则以下对 a 数组元素的引用错误的是(　　)。
   A. a[--i]　　B. a[a[0]]　　C. a[a[1*2]]　　D. a[a[i]]
4. 以下能正确定义一维数组的选项是(　　)。
   A. int num[ ];
   B.
   ```
   #define N 100
   int num[N];
   ```
   C. int num[0..100];
   D. int N=100,num[N];
5. C 语言中，下面能正确定义一维数组并初始化的选项是(　　)。
   A. int a[5]={0,1,2,3,4,5};　　B. int a[5]={3};

C. int a[5]=5;　　D. int a[N]={1,2,3};

6. 设有定义“int a[3][4];”,则对数组元素引用正确的是(　　)。

A. a[2][4]　　B. a[2][0]　　C. a[3][3]　　D. a[0][4]

7. 若有说明:int a[][3]={{1,2,3},{4,5},{6,7}};则数组a的第一维的大小为(　　)。

A. 2　　B. 3　　C. 4　　D. 无确定值

8. 若有初始化语句“int a[3][4]={0};”,则下面正确的叙述是(　　)。

A. 只有元素a[0][0]可得到初值0

B. 此初始化语句不正确

C. 数组a中每个元素都可得到初值0

D. 数组a中各元素都可得到初值,a[0][0]初值0,其他元素初值为随机数

9. 下述对C语言字符数组的描述错误的是(　　)。

A. 字符数组中的内容不一定是字符串

B. 可以用输入语句把字符串整体输入给字符数组

C. 可以在赋值语句中通过赋值运算符"="对字符数组整体赋值

D. 字符数组中没有'\0'也可以正确输出其中的内容

10. 下面是对数组s的初始化,其中错误的语句是(　　)。

A. char s[5]={"boy"};　　B. char s[5]={ 'b','o','y'};

C. char s[5]=" ";　　D. char s[5]="Frank";

11. 已有定义“char a[ ]="boy",b[ ]={'b','o','y'};”,则以下叙述中正确的是(　　)。

A. 数组a和数组b长度相同　　B. 数组a的长度小于数组b的长度

C. 数组a的长度大于数组b的长度　　D. 上述说法都不对

12. 函数调用strcat(strcpy(s1,s2),s3)的功能是(　　)。

A. 将串s1复制到串s2中,然后再连接到s3之后

B. 将串s1连接到串s2之后,再复制到s2之中

C. 将串s2复制到s1中,再将串s3连接到串s1之后

D. 将串s2复制到s1中,再将s1连接到串s3之后

13. 下列叙述错误的(　　)。

A. 空字符串也占用内存,其内存空间大小是1B

B. 两个连续的单引号可表示合法的字符常量

C. 两个连续的双引号可表示合法的字符串常量

D. 不能用关系运算符比较字符串的大小

14. 设有定义“int a[2][3];”,则以下的叙述错误的是(　　)。

A. a[0]可看作是由3个整型元素组成的一维数组

B. a[0]和a[1]数组名,分别代表不同的地址常量

C. 可以用语句“a[0]=0;”为数组所有元素赋初值

D. 数组a包含6个元素

15. 以下程序运行后的输出结果为(　　)。

```
#include <stdio.h>
int main()
```

```
{
  char str[3][5] = {"AAAA","BBB","CC"};
  printf("%s\n",str[1]);
  return 0;
}
```

A. AAAA　　B. BBB　　C. CC　　D. BBBCC

16. 有以下程序，在运行时输入 how do you do<回车>，则输出结果为(　　)。

```
#include <stdio.h>
int main()
{
  char a[20],b[20],c[20];
  scanf("%s%s",a,b);
  gets(c);
  printf("%s%s%s\n",a,b,c);
  return 0;
}
```

A. Howdoyoudo　　B. Howdo you do

C. How do you　　D. Howdoyou do

17. 以下程序运行后的输出结果为(　　)。

```
#include <stdio.h>
int main()
{  int A[10] = {9,3,2,15,12};
  int i,t,n = 5;
  A[n] = 10;
  i = n - 1;
  while((i >= 0)&&(A[i]> A[i + 1]))
  {
    t = A[i];A[i] = A[i + 1];A[i + 1] = t;
    i -- ;
  }
  n++;
  for(i = 0;i < n;i++)
    printf("%d ",A[i]);
  return 0;
}
```

A. 2,3,9,10,12,15　　B. 9,3,2,10,15,12

C. 15,12,10,9,3,2　　D. 9,3,2,15,12,10

18. 以下程序运行后的输出结果为(　　)。

```
#include <stdio.h>
int main()
{
  int a[3][4] = {5,4,6,7,3,9,0,2,8,1,3,5 };
  int t[3],i,j;
  for (i = 0; i < 3; i++)
  {
```

```
    t[i] = a[i][0];
    for(j = 1; j < 4; j++)
      if(a[i][j]> t[i]) t[i] = a[i][j];
  }
  for (i = 0; i < 3; i++)
    printf(" % d ", t[i]);
  return 0;
}
```

A. 4 0 1　　B. 7 4 9　　C. 7 9 8　　D. 5 3 8

19. 以下程序运行后的输出结果为(　　)。

```
#include < stdio.h >
#include < string.h >
int main()
{
  char str[100] = "How do you do";
  strcpy(str + strlen(str)/2,"es she");
  printf(" % s\n",str);
  return 0;
}
```

A. How do you do　　B. es she

C. How are you　　D. How does she

20. 以下程序运行后的输出结果是(　　)。

```
#include < stdio.h >
int main()
{
  int a[6] = {2,4,6,8,10,12};
  int b[6] = {6,8,10,7,5,1};
  int i,j;
  for (i = 0; i < 6; i++)
  {
    for (j = 0; j < 6; j++)
      if (a[i] == b[j]) break;
    if (j < 6) printf(" % d ", a[i]);
  }
  printf("\n");
  return 0;
}
```

A. 2 4 6 8 10 12　　B. 10 7 5　　C. 2 4 12　　D. 6 8 10

5.7.2 在线编程题

1. 简单评委打分。(nbuoj1147)

学生参加项目结题汇报,假设有 8 位老师作为评委。计算学生最终得分的方法如下:去掉一个最高分和一个最低分,计算剩余 6 个分数的平均值,所得结果就是该学生的最后得分。本题先输入 8 个分数,去掉一个最高分和一个最低分后计算平均得分。输出保留两位

小数。

2. 求年月日。(nbuoj1075)

输入两个整数分别代表某一年和这一年的第几天(假设数据都在有效范围内),要求输出具体的年、月、日的信息。如输入 2011 20,则输出 2-11-1-20。

3. 最高分和最低分。(nbuoj1157)

已知有 10 个同学的成绩,求最高分和最低分以及相应分数所在的位置。本题输入 10 个整数,假设这 10 个数互不相同,且无序排列。请找出其中最大数及它在数组中的下标,以及最小数和下标。

4. 十进制转换成八进制。(nbuoj1172)

输入一个十进制整数,把这个数转换为八进制的数输出。

5. 百灯判熄。(nbuoj1122)

有 M 盏灯,编号为 1～M,分别由相应的 M 个开关控制。开始时全部开关朝上(朝上为开,灯亮),然后进行以下操作:编号凡是 1 的倍数的灯反方向拨一次开关;是 2 的倍数的灯再反方向拨一次开关;是 3 的倍数的灯又反方向拨一次开关,……,直到是 M 的倍数的灯又反方向拨一次开关。本题输入一个整数 m(1≤m≤100)代表灯的数量,要求输出最后为熄灭状态的灯(不亮)的数量以及编号。

6. Susan 的货币兑换。(nbuoj1167)

Susan 到中国观光旅游,她不太熟悉人民币,因此分别将 1 角,2 角,5 角,1 元,2 元,5 元,10 元,20 元,50 元,100 元的人民币依次排序号(从 1 开始排序号),她每天将自己手中不同面值人民币的张数输入 iPAD,以计算手头的人民币数额。请帮她编写一个程序,可以根据她手中的不同面值人民币的张数,计算出对应的人民币数额。本题输入人民币序号及张数,每种面值占据一行,如 5 20 表示序号为 5 的人民币有 20 张,当输入序号或张数为负数时结束,要求输出对应的人民币数值。输出保留两位小数。

7. 对角线元素和。(nbuoj1164)

输入一个整数 n,然后输入 n×n 个数据建立一个方阵,计算并输出方阵主对角线元素的和。

8. 上三角置零。(nbuoj1298)

输入一个 5 行 5 列的二维矩阵,输出上三角置零后的二维矩阵。

9. 二维数组最大值。(nbuoj1161)

输入 12 个整数构成一个 3×4 的二维数组,求出该数组的最大元素并输出。

10. 二维数组每行最大值。(nbuoj1191)

输入 12 个整数构成一个 3×4 的二维数组,求出每行的最大元素并输出。

11. 内部和。(nbuoj1299)

输入两个整数 m 和 n(范围为 1～9),接着输入 m×n 个数据建立一个二维矩阵,计算并输出矩阵内部元素(不包括最上下两行及最左右两列)的和。

12. 特定字符出现次数。(nbuoj1056)

输入一个字符串(长度小于等于 1000),再输入一个待查找的特定字符 key,统计 key 在字符串中的出现次数

13. 相邻字符判相等。(nbuoj1054)

输入一行字符(长度小于等于1000),判断其中是否存在相邻两个字符相同的情形,若有,则输出该相同的字符并结束程序(只需输出第一种相等的字符即可),否则输出No。

14. 单词译码。(nbuoj1139)

输入一个单词,长度不超过9(假设输入内容全部都是英文字母,不存在其他字符)。对该单词进行译码并输出结果。译码规律是:用原来字母后面的第4个字母代替原来的字母,并能循环译码。例如,字母A后面第4个字母是E,用E代替A;同理,字母y用c代替。如单词China应译为Glmre,单词Today应译为Xshec。

15. 回文数字。(nbuoj1144)

给定一个数字字符串,长度不超过100,判断它是否是回文数字。例如,121, 1221是回文数字, 123不是回文数字。若是回文输出Yes, 否则输出No。

16. 回文字符串。(nbuoj1145)

给定一个字符串,长度不超过100,判断它是否是回文串。例如,aba, abcba是回文,abc, xyy不是回文。若是回文输出Yes, 否则输出No。

17. 数组字符数出现频率。(nbuoj1148)

输入一行文本,统计其中数字字符0～9出现的频率并输出。没有出现的数字字符不要显示。

18. 字母出现频率。(nbuoj1159)

输入一行文本(小于1000字符),统计其中每个英文字母出现的频率,输出出现过的英文字母及其次数,未出现过的字母不需要显示。为了简化问题的复杂度,假设在统计过程中不区分字母的大小写,即'A'与'a'被认为是一种字母。

19. C语言合法标识符。(nbuoj1190)

输入一个长度不超过50的字符串,判断其是否为C语言合法的标识符。

20. 输出最短字符串。(nbuoj1201)

输入五个字符串,输出其中最短的字符串,若长度相同则输出出现较早的那一个。每个字符串长度不大于1000。

21. 判断字符串类型。(nbuoj1199)

输入一个字符串(长度不超过1000),其中只包括数字或字母。对应输入的字符串,输出它的类型。如果仅由数字构成的则输出digit,如果仅由字母构成的则输出character,如果是由数字和字母一起构成的则输出mixed。

22. 你能找出多少个整数?(nbuoj1315)

输入一个字符串,由空格、英文字母、数字组成,长度小于1000。输出字符串中的整数的个数(不是数字字符)。

23. 查找最大字符串。(nbuoj1175)

输入一行长度不超过100的字符串,字符串仅由大小写字母构成,查找其中最大字母(按ASCII码大小排),在该字母后面插入字符串"(max)",不包括引号。如果存在多个最大的字母,就在每一个最大字母后面都插入"(max)"。

24. 去过的城市。(nbuoj1352)

CoCo喜欢旅游,每次都会去一个地方,并且每去过一个地方都会记录一下地名,当然有些地方去过多次也都会一一记录下来的。现在列出了CoCo去过的n个城市的名称(会

有重复的),然后再输入一个城市的名称,请你帮忙计算一下这个城市 CoCo 去过几次了。本题先输入一个正整数 n(n≤1000),接下来 n 行依次输入 n 个字符串表示 CoCo 去过的城市名,每个字符串的长度小于等于 100 字符,并且字符串中无空格。然后再输入一个城市名表示待查找字符串,要求输出该城市 CoCo 已经去过几次了。

25. 加法的进位。(nbuoj1451)

多位数的加法通常会有进位,如 555+555 有三次进位。从键盘输入两个正整数,计算它们在进行加法运算时有几次进位。(每个数不超过 20 位。)