第5章

数　　组

前面章节中程序涉及的数据量都比较小，简单变量（即每个变量都使用一个独立的名称，变量间不存在联系）就可以很方便地进行存取，但是在实际问题中往往会有大量的相关数据需要存取，例如，要存储 40 个学生 5 门课程的成绩，如果用简单变量来存取这些数据，需要声明 200 个简单变量，这样做不但烦琐，而且效率低。为了在程序中能方便高效地处理大批量的数据，充分发挥循环控制结构的作用，VB 中引入了数组。本章将着重介绍数组的特点以及在实际问题中的应用。

引例：求一个班 5 个学生的某门课的平均成绩，然后统计高于平均分的人数。

程序代码如下。

```
Private Sub Form_Click()
    Dim s!, aver!, overn%, mark!
    s=0
    aver=0
    For i=1 To 5
        mark=InputBox("输入第" & i & "位学生的成绩")
        s=s+mark
    Next i
    aver=s/5
    overn=0
    For i=1 To 5
        mark=InputBox("输入第" & i & "位学生的成绩")
        If mark>=aver Then overn=overn+1
    Next i
    Print aver, overn
End Sub
```

通过阅读上述程序代码可以知道，在引例中，由于变量 mark 多次赋值后，只保留最后一次的值，所以，在求平均值时，在循环体内，给变量 mark 输入了 5 个学生的分数，输入完后，mark 只保存了最后一个学生的分数；而要统计高于平均分的人数时，又要在循环体内将这 5 个学生的分数再给变量 mark 输入一遍。也就是说，给 mark 赋了两遍值。这成倍的工作量势必降低工作效率，有没有办法只赋一遍值并且将所赋的值都保存下来供程序多次使用呢？

在数学中表示一个数列时使用 $x_1, x_2, x_3, \cdots, x_i, \cdots, x_n$ 的形式，这种形式称为数列。数列中的每一个数据项都表示为 x_i 的形式，其中 x 为数列名，i 为数据项在数列中所处的位置，称为下标，x_i 称为下标变量。这种表示形式的好处是能批量处理数据，容易找出数据之间的依赖关系(例如，Fibonacci 数列等)。在 VB 中，为了能处理一批数据或与其位置有关的数据，也采取这种类似的方法，所不同的只是将下标放入一对括号“()”中。例如，在引例中，就可以采用这种方法，可将 5 个学生的某门课的成绩分别存入到 mark(1)，mark(2)，mark(3)，mark(4)，mark(5)中。其中，mark 称为数组名，mark(i)(i=1,2,…,5)称为 mark 数组中的元素，数组元素可以存储第 i 个学生的成绩，i 称为数组元素的下标，下标指出某个数组元素在数组中的位置，所以，数组元素也叫下标变量。要使用这些下标变量 mark(i)，必须在使用之前声明一个 mark 数组。

5.1 数组的基本概念

数组是一组相同名称的下标变量的集合。这些下标变量称为数组元素，每个数组元素都有一个编号，这个编号叫做下标，放在圆括号“()”里，通过下标来区别这些元素，下标代表元素在数组中的位置。下标变量的名称称为数组名。例如，下标变量 mark(1)，mark(2)，mark(3)，mark(4)，mark(5)是数组 mark 中的元素。

数组概念

数组元素和数组是个体和整体的关系。在计算机中，数组占据一块内存区域，数组名是这个区域的名称，区域的每个单元就是数组元素。

数组必须遵循先声明后使用的原则，声明一个数组就是声明其数组名、类型、维数和数组的大小。

在 Visual Basic 中，数组元素一般都是同种类型的，而 Variant 类型(默认)数组的各元素可以是不同的数据类型，但建议不要使用。

按数组的大小(元素个数)是否可以改变，可将数组分为静态(定长)数组和动态(可变长)数组两类。

按元素的数据类型，可分为数值型数组、字符型数组、日期型数组、变体类型数组等。

按数组元素的下标数量的不同，数组可分为一维数组、二维数组、多维数组(最多 60 维)。一维数组的元素只有一个下标，二维数组的元素有两个下标，多维数组的元素有多个下标。

按数组元素的性质，可分为数据数组和控件数组。

5.1.1 静态数组及其声明

1. 一维数组

声明格式：

```
Dim|Private|Static|Public<数组名>([<下界>To]<上界>)[As<数据类型> ]
```

或

```
Dim|Private|Static|Public<数组名>[<数据类型符>] ([<下界>To]<上界>)
```

功能：声明一维数组的名称、大小、类型，并为数组分配存储空间。

说明：

(1) Dim 语句可用在窗体的通用声明段中，定义窗体数组，也可用于过程中；Private 语句用在窗体的通用声明段中，定义窗体数组；Static 语句用在过程中；Public 语句用在标准模块的通用声明段中，定义全局数组。这 4 种语句的适用范围也同样适用于各种类型的数组声明。

(2) 数组名的命名规则与变量的命名相同。在同一个过程中，数组名不能与变量名同名，否则会出错。

(3) 数组的大小就是数组元素的个数。一维数组的元素个数：上界－下界＋1(下界≤上界)。下界最小值为－32 768，上界最大值为 32 767。

(4) 如果省略＜下界＞To，其下界默认值为 0。若希望下界从 1 开始，可在窗体或模块的通用声明段使用 Option Base 1 语句将其设为 1。格式：Option Base n(n 只能取 0 或 1。该语句只能出现在窗体模块或标准模块的通用声明段，不能出现在过程内，而且必须在数组声明之前设置。如果声明的是多维数组，则使用该语句设置的默认下界值对每一维都有效)。

(5) ＜下界＞和＜上界＞必须是数值型常量或由数值型常量构成的表达式，常量可以是直接常量、符号常量，一般是整型常量。

例如：

```
Const k As Integer=10
Dim x(10) As Single                '正确
Dim a(k) As  long                  '正确
```

而

```
n=10
Dim x(n) As Single                 '错误
```

(6) 如果省略 As 子句，则数组的类型为变体类型。

(7) 数组声明，并初始化所有数组元素。数值型数组中的元素初值是 0，字符型数组中的元素初值是空字符串("")，逻辑型数组中的元素初值是 False，变体类型数组中的元素初值是空(Null)。

例如：

```
Dim  A(-1 To 5)As Integer
'声明了名称为 A 的一维数组，共有 7 个整型元素，分别是 A(-1),A(0),A(1),A(2),A(3),A(4),A(5)
Dim  B(5)As Integer
'声明了名称为 B 的一维数组，共有 6 个整型元素，分别是 B(0),B(1),B(2),B(3),B(4),B(5)
```

通过学习一维数组的有关知识，引例中的学生成绩就可以只输入一遍值并且能保存

下来供后面的程序使用了,程序修改如下。

```
Private Sub Form_Click()
    Dim s!, aver!, overn%, mark! (1 To 5)
    s=0
    aver=0
    For i=1 To 5
        mark(i)=InputBox("输入第" & i & "位学生的成绩")
        s=s+mark(i)
    Next i
    aver=s/5
    overn=0
    For i=1 To 5
        If mark(i)>=aver Then overn=overn+1
    Next i
    Print aver, overn
End Sub
```

2. 多维数组

声明格式:

```
Dim|Private|Static|Public<数组名>([<下界>To] <上界>,[<下界>To]<上界>…)
[As<数据类型>]
```

或

```
Dim<数组名>[<数据类型符>] ([<下界>To]<上界>,[<下界>To ]<上界>…)
```

功能:声明数组的名称、维数、每一维的大小、类型,并为数组分配存储空间。

说明:每一维的大小为上界-下界+1;数组的大小为各维大小的乘积。

例如:

```
Dim a(1 To 2, 1 To 3)As Integer
```

声明了名为a的二维数组,数组的元素均为整型,数组共有6个元素,分别是a(1,1),a(1,2),a(1,3),a(2,1),a(2,2),a(2,3),这6个元素将顺序存放在内存里连续的一段空间中。

注意:数组声明中定义的数组名,用来说明数组的名字、维数、大小和类型。数组元素是数组中的一个成员,只能放在可执行语句中。两者虽然形式相同但意义不同。

例如:

```
Dim  b(1,2)  As Single          '声明2行3列的二维数组
b(1,2)=3.2                      '给数组元素b(1,2)赋值
```

5.1.2 动态数组及其声明

在利用数组进行程序设计时,可能会出现数组长度在程序中发生变化的情况。这时,最适合采用动态数组。

1. 动态数组的声明

声明格式：

```
Dim|Private|Static|Public<数组名>[数据类型符]() [As <数据类型>],…
ReDim [Preserve] <数组名>([下界 1 To] <上界 1>[, [下界 2 To] <上界 2>,…]),…
```

功能：先用 Dim 等声明动态数组名和类型，然后在过程内使用 ReDim 确定动态数组的维数和实际大小，并为其分配存储空间。

说明：

(1) Dim、Private、Public、Static 变量声明语句是说明性语句，可出现在过程内或通用声明段；而 ReDim 语句是执行语句，只能出现在过程内。

(2) 在过程中可多次使用 ReDim 来改变数组的大小，也可改变数组的维数，但不能改变其类型。

(3) 每次使用 ReDim 语句都会使原来数组中的值丢失，可以在 ReDim 语句后加 Preserve 参数以保留数组中的数据，但使用 Preserve 只能改变最后一维的大小，前面几维大小不能改变。若改变，则出现下标越界的错误提示。

(4) 在静态数组声明中的下标只能是常量，在动态数组 ReDim 语句中的下标可以是常量，也可以是有了确定值的变量。

(5) Dim、Private、Public、Static 变量声明语句只是说明声明的数组为动态数组，系统并不为其分配存储空间，使用 ReDim 语句重新声明数组后，才确定了数组的维数和大小，此时才为数组分配存储空间。

例 5.1　改进引例的程序，使其更通用。要求输入 n 个学生的成绩，计算平均分和高于平均分的人数，并将这两项放在该数组的最后。

分析：因为学生的成绩个数不确定，所以要将输入的学生成绩保存下来不能用静态数组，需要声明动态数组。又要将平均分和高于平均分的人数保存下来，并且输入的学生成绩需保留，则需重新声明数组并且在重新声明的语句中要加参数 Preserve。

程序代码如下。

```
Private Sub Form_Click()
    Dim mark%(), i%, n%, s!
    n=InputBox("输入学生的人数")
    ReDim mark(1 To n)
    s=0
    For i=1 To n
        mark(i)=InputBox("输入第"&i&"位学生的成绩")
        s=s+mark(i)
    Next i
    ReDim Preserve mark(1 To n+2)
    mark(n+1)=s/n
    mark(n+2)=0
    For i=1 To n
```

```
            If mark(i)>mark(n+1) Then
                mark(n+2)=mark(n+2)+1
            End If
        Next i
        For i=1 To n
            Print "mark("; i; ")="; mark(i)
        Next i
        Print "平均分="; mark(n+1), "高于平均分人数="; mark(n+2)
    End Sub
```

程序运行结果如图 5.1 所示。

使用动态数组的优点是,可根据用户需要,有效地利用存储空间,它是在程序执行到 ReDim 语句时分配存储空间,而静态数组是在程序编译时分配存储空间的。

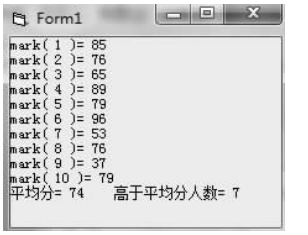

图 5.1 程序运行结果

2. 数组函数

1) LBound()函数

格式:

```
LBound(数组名,[维数])
```

功能:返回数组某维的下界值。

例如,LBound(A,1),返回值为数组 A 的第 1 维的下界的值。

2) UBound()函数

格式:

```
UBound(数组名,[维数])
```

功能:返回数组某维的上界值。

例如,UBound(B,2),返回值为数组 B 的第 2 维的上界的值。

5.2 数组的基本操作

数组是程序设计中最常用的结构类型,将数组元素的下标和循环语句结合使用,能解决大量的实际问题。数组在定义时用数组名表示该数组的整体,但在具体操作时是针对每个数组元素进行的。

数组操作

5.2.1 数组元素的输入

1. 利用循环结构对数组元素逐一赋值

下面一段程序利用单循环结构对一维数组 A 赋 1～100 的随机整数。

```
Private Sub Form_Click()
    Dim A(5) As Integer
    Dim  I As Integer
    For I=1 To 5
        A(I)=Int(100 * Rnd)+1
        Print A(I);
    Next I
End Sub
```

下面这段程序利用双循环结构通过 InputBox 函数在程序执行时给二维数组 sc 赋值。

```
Dim  sc(1 To 4,1 To 5) As String
    For i=1 To 4
        For j=1 To 5
            sc(i, j)=InputBox("输入 sc("  &  i  & ", " & j &") 的值")
        Next j
Next I
```

2. 利用 Array()函数赋值

利用 Array()函数把一组数据赋值给一个一维数组。

格式：

```
<变体变量名>=Array([数据列表])
```

说明：

(1) 利用 Array()函数对数组各元素赋值，声明的数组是动态数组或连圆括号都可省略的数组，并且其类型只能是 Variant。

(2) “数据列表”以逗号间隔，如果不提供参数，则创建一个长度为 0 的数组。

(3) 数组的下界默认为零，也可通过 Option Base 语句决定，上界由 Array()函数括号内的参数个数决定，也可通过 UBound()函数获得。

例如，要将 1,2,3,4,5,6,7 这些值赋值给数组 a，可使用下面的方法赋值。

```
Dim a()
a=Array(1,2,3,4,5,6,7)
```

或者使用下面的方法：

```
Dim a
a=Array(1,2,3,4,5,6,7)
```

注意：一维数组既可以用单循环结构赋值，也可以用 Array()函数赋值；而二维数组的赋值只能利用双循环结构。一般几维数组就用几重循环来赋值。

5.2.2　数组元素的输出

例 5.2　形成 5×5 的方阵，在窗体中分别输出方阵中的各元素，上三角和下三角

元素。

分析：对于二维矩阵的输出，一般使用二重循环。在这3种输出方式中，只是对循环控制变量的起始值或者结束值做了相应的修改，读者可以对照输出结果查看程序代码，比较容易理解。

程序的代码如下。

```
Private Sub Form_Click()
    Dim sc%(4, 4)
    Print  "产生方阵元素"
    For i=0 To 4
        For j=0 To 4
            sc(i, j)=i * 5+j
            Print Tab(j * 5); sc(i, j);
        Next j
        Print
    Next i
    Print  "显示上三角元素"
    For i=0 To 4
        For j=i To 4
            sc(i, j)=i * 5+j
            Print Tab(j * 5); sc(i, j);
        Next j
        Print
    Next i
    Print  "显示下三角元素"
    For i=0 To 4
        For j=0 To i
            sc(i, j)=i * 5+j
            Print Tab(j * 5); sc(i, j);
        Next j
        Print
    Next i
End Sub
```

输出的结果如图5.2所示。

图5.2 程序运行结果

5.2.3 复制整个数组

在VB 6.0以前的版本中，若要将一个一维数组的各个元素的值赋值给另一个一维数组元素，这要通过一个单循环来实现。在VB 6.0中，只要通过一个简单的赋值语句即可。

例如：

```
Private Sub Form_Click()
```

```
    Dim a(), b()
    DimI As Integer
    a()=Array(1, 2, 3, 4, 5)
    b=a
    For I=LBound(a) To UBound(a)
        Print a(I); b(I)
    Next
End Sub
```

上述程序中输出 a 数组各元素的值和 b 数组各元素的值是相同的。其中 b＝a 语句相当于执行了：

```
ReDim b(UBound(a))
For i=0 To UBound(a)
    b(i)=a(i)
Next i
```

注意：

(1) 赋值号两边的数据类型必须一致。

(2) 如果赋值号左边的是一个动态数组，则赋值时系统自动将动态数组 ReDim 成右边同样大小的数组。

(3) 如果赋值号左边是一个静态数组，则数组赋值出错。

5.2.4　For Each… Next 语句

通过 For Each…Next 语句，可以十分方便地访问数组中的任一元素。For Each…Next 语句类似于 For …Next 语句，两者都用来执行指定重复次数的一组操作，但 For Each…Next 语句专门用于数组或对象“集合”(本书不涉及集合)，其一般格式为：

```
For Each <成员>In <数组名>
    循环体
    [Exit  For]
    …
Next[成员]
```

功能：用于指定数组中的数组元素个数作为循环次数，重复执行循环体中的语句。

说明：

(1) 语句中的“成员”是循环变量，实际上是数组中的每个数组元素。Next 语句中的“成员”与 For Each 语句中的成员是同一个变量，可以省略。该变量必须是变体类型。

(2) 数组名是已经声明过的一维数组或多维数组名，其后可以带一对圆括号，但不能带下标。

(3) 语句的执行过程是：首先从指定的数组中按顺序依次取出一个数组元素值赋给变量，然后执行循环体，直到数组中的数组元素值取完为止。

(4) 如果循环体中包含 Exit For 语句，则执行该语句后退出循环，接着执行 Next 后

面的语句。

例如：

```
Private Sub Form_Load()
Dim a(9)
    For I=0 To 9
    a(I)=I
    Next
    For Each e In a
        f=f+e
    Next
    MsgBox "数组 a 中所有数据的和是" & f, vbInformation, "计算结果"
End Sub
```

上述程序中的e类似于For…Next循环中的循环控制变量，但不需要为其提供初值和终值，而是根据数组元素的个数确定执行循环体的次数。此外，e的值就是a数组中的各元素的值，开始执行时，e是数组第1个元素的值，执行完一次循环体后，e变为数组第2个元素的值……，当e为最后一个元素的值时，执行最后一次循环。e是一个变体变量，它可以代表任何类型的数组元素。

可以看出，在数组操作中，For Each…Next语句比For…Next语句更方便，因为它不需要指明结束循环的条件，不必根据初值、终值和步长值来确定循环次数，而是根据数组中元素的个数确定循环次数，当按顺序依次处理数组的全部元素时比较方便。缺点是难以处理数组中的部分元素，也难以控制数组元素的处理次序。

注意：不能在For Each…Next语句中使用用户自定义类型数组，因为Variant不能包含用户自定义类型。

5.3 列表框和组合框控件

列表框和组合框都是通过列表的形式显示多个项目，供用户选择，达到与用户对话的目的。并且，当列表项目过多时，还会自动出现垂直滚动条。列表框和组合框控件实质就是一维字符数组，以可视化形式直观显示其项目列表。

列表框与组合框

5.3.1 列表框

列表框控件可以同时显示多个项目的列表，供用户选择，但不能编辑。列表框中可供选择的项目称为表项。表项加入列表框是按照一定顺序号进行的，这个顺序号称为索引。当表项较多而超出设计列表框所能显示的范围时，系统会在列表框上自动添加垂直滚动条，方便用户滚动选择表项。在窗体上添加的列表框的默认名称为List1，List2，…。