第3章 chapter 3

数　组

学习目标与设问：

- 程序设计中为什么要使用数组
- 数组如何定义
- 如何在运行过程中扩展数组
- 特殊的数组——字符串
- 如何定义和使用平行数组

在使用程序来解决问题时，会遇到大量的数据涌现出来，对数据进行处理时需要将其进行保存、排序、比较、选择和统计。在学习完第 2 章以后，我们知道可以使用变量对数据进行保存和运算，同时也知道变量的命名是有一定的规则的。如果做一个简单的练习：输入 10 个数，求它们的平均值并输出结果。如果不用数组来解决这个问题，语句书写会很复杂，且重复语句多。

但成千上万的数据变量如何起名和管理？在程序设计实践中，最常使用的方法就是将大量相同类型的数据组成数组进行运算。

3.1　数组的概念

所谓数组，即相同数据类型的元素按一定顺序排列的集合，就是把有限个类型相同的变量用一个名字命名，然后用编号加以区分的变量的集合，这个名字称为数组名，编号称为下标。组成数组的各个变量称为数组的分量，也称为数组的元素，有时也称为下标变量。数组是在程序设计中，为了处理方便，把具有相同类型的若干变量按有序的形式组织起来的一种形式。这些按序排列的同类数据元素的集合称为数组。数组最大的好处在于用一个统一的数组名和下标(index)来唯一地确定某个数组变量中的元素。而且下标值可以参与计算，这为动态进行数组元素的遍历访问创造了条件。

数组中的各元素是有先后顺序的，它们在内存中按照这个先后顺序连续存放在一起。由于有了数组，可以用相同名字引用一系列变量，并用数字(索引)来识别它们。在许多场合，使用数组可以缩短和简化程序，因为可以利用索引值设计一个循环，高效地处理多种情况。数组有上界和下界，数组的元素在上下界内是连续的。

RAPTOR 数组的特点如下：

(1) 在 RAPTOR 中一个数组可以分解为多个数组元素，这些数组元素可以是基本数据类型。因此按数组元素的类型不同，数组又可分为数值数组、字符数组和二维数组等类别。

(2) 一般一个数组中的所有元素具有相同的数据类型。但在 RAPTOR 中，一个数组中的各个元素能够包含不同种类的数据(字符、字符串和数值等)。

(3) RAPTOR 支持可变长数组(VLA)。

(4) 数组元素用整个数组的名字和该元素在数组中的顺序位置来表示。例如，a[1]代表名字为 a 的数组中的第一个元素，a[2]代表数组 a 的第二个元素，以此类推。

(5) RAPTOR 下标要紧跟在数组名后，而且用方括号括起来(不能用其他括号)。

(6) 下标可以是常量、变量或表达式，但其值必须是整数(如果是小数将四舍五入为整数)。

(7) 下标必须为一段连续的整数，其最小值称为下界，其最大值称为上界。不加说明时下界值默认为 1。

(8) RAPTOR 数组的最大元素个数在 10 000 个左右，建议不要超过此上限。

图 3-1 显示了 RAPTOR 一维数组 scores[]的样例，该图显示的是该数组各个元素的名称而不是其中的数据，其中方括号中的数值就是下标值，以便各个元素相互区分。在 RAPTOR 中规定，下标值必须是正整数，不可以是 0 或带有小数的数值。

scores[1]	scores[2]	scores[3]	scores[4]

图 3-1 RAPTOR 一维数组的元素表示

图 3-2 则显示了 RAPTOR 一维数组的另一个视角，其中列出了 scores[]数组中各元素的样本数值和其下标值。

scores			
1	2	3	4
87	93	77	82

图 3-2 RAPTOR 一维数组的样本数据与下标

数组的命名规则与变量基本相同。但在 RAPTOR 中，已经成为数组变量的名称，不得再重复用作一个普通变量的名称。

3.2 数组的类型

3.2.1 一维数组

数组变量必须在使用之前创建。在 RAPTOR 中的数组是在输入和赋值语句中，通过给一个数组元素赋值而产生的。所创建的数组大小由赋值语句中给定的最大元素下标来决定。创建过程也可以通过一个计数循环进行，所以数组可以最初只有一个元素、

然后两个、三个，随着循环过程的运行，元素个数与下标逐渐增加。

如果程序试图访问的数组元素下标大于以前赋值语句产生过的任何数组元素的下标，则系统会发生一个运行时错误。

然而，数组中的元素可以按任何顺序赋值，这样会留下一些没有赋值的元素。在这种情况下，仍然使用最大下标定义数组的大小，但未赋值的数组元素将默认为 0（数值类型）。例如，观察以下数组的赋值过程。

第一次给 values[]数组赋值：

```
values[7]<-3
```

产生图 3-3 的结果。

1	2	3	4	5	6	7
0	0	0	0	0	0	3

图 3-3　第一次给数组 values[]赋值的结果

第二次再给该数组赋值：

```
values[9]<-6
```

则将数组进行了扩展，得到的结果如图 3-4 所示。

1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	3	0	6

图 3-4　第二次给数组 values[]赋值的结果

这种赋值方式的方便之处是，程序员可以用一个赋值语句初始化整个数组为 0。例如，100 个元素的数组初始化为 0，只需要一个语句：

```
values[100]<-0
```

就可以用来代替一个循环 100 次的数组赋值过程。

最常见的数组运算是使用下标对其进行访问。也就是说在数组名后的方括号中指定下标值，例如，values[count]或 grid[row,column]。

在 RAPTOR 中，提供了一个 Length_Of()函数来计算一维数组的长度。Length_Of()函数需要一个参数，也就是需要进行长度计算的一维数组名，不需要数组名后的方括号或任何下标形式的表示。它返回为数组定义的最大下标值（也就是元素的数量）。

例如，对如图 3-5 所示的一维数组，使用 Length_Of(scores)进行运算，得到的返回值为 4。

scores

1	2	3	4
87	93	77	82

图 3-5　一个有 4 个元素的数组样本

数组变量的好处是数组符号允许 RAPTOR 在方括号内执行数学计算。换句话说，

RAPTOR 可以计算数组的下标。因此，表达式计算所得的相同的下标，均指向相同的变量，例如：

```
a[2]
a[1+1]
a[23-21]
a[(5-14)*2+4*7-2*(7-3)]
```

在 RAPTOR 中，数组应用是有一些限制的，在方括号内的表达式可以是产生一个正整数的任何合法的表达式，在涉及数组变量时，RAPTOR 会重新计算下标的表达式。这才是数组变量和数组表示法的力量所在。

如图 3-6 所示的代码片段利用数组求一个数字集合的平均值。虽然只有 4 个数组元素，但是，这个程序如果扩展到 4000 个，也只要修改一下 howmany 变量值即可。这就是数组的力量。

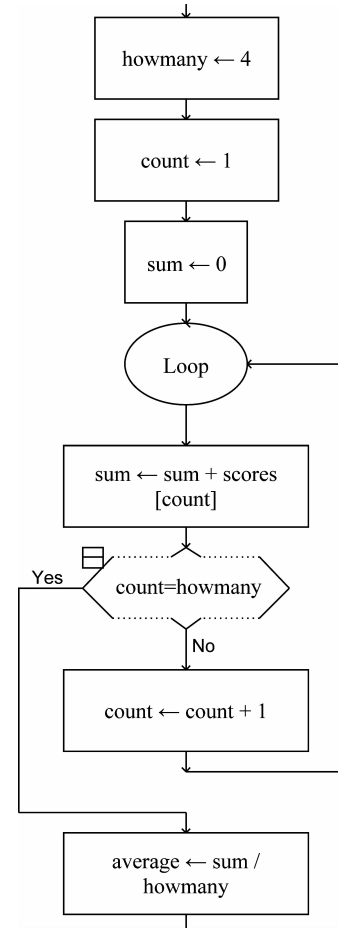


图 3-6 使用数组进行循环计算

到了这里，已经基本解决了一些简单变量程序的缺点，尽管可能不得不修改源程序，但是至少现在只需稍微修改程序，就可以处理不同数量的计算问题。目前，对于解决不同计算量的问题，已经有了一种快捷便利的方法。

例 3-1 利用数组编写程序，求 1＋3＋…＋99 的值。

解：题目中要求用数组来处理，那么需要将 1～99 中的 50 个奇数放到数组中，再进行累加运算。无论是给数组赋值，还是最后累加，都可以借助循环来处理，以简化程序。

程序处理过程中，可以先统一赋值再累加，还可以边赋值边累加。

程序流程图如图 3-7(a)所示，运行结果如图 3-7(b)所示。

3.2.2 二维数组

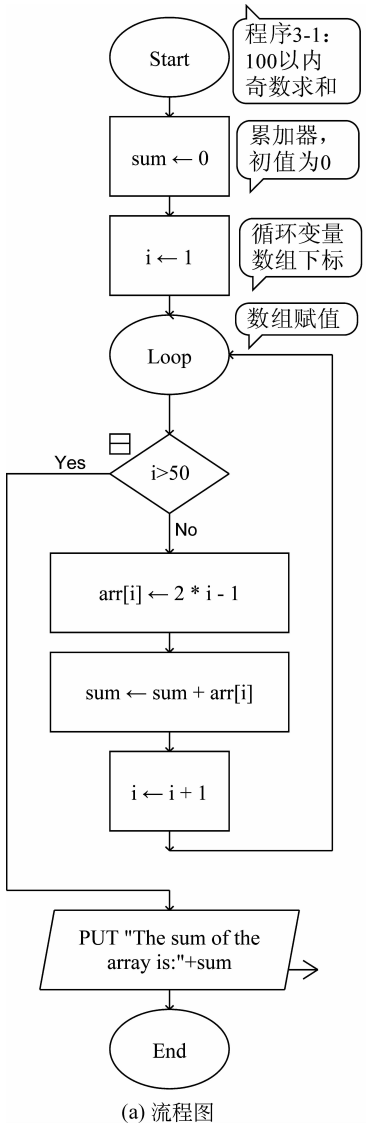
创建二维数组时，数组的两个维度的大小由最大的下标确定。同样，使用以下赋值语句：

```
numbers[3,4]<-13
```

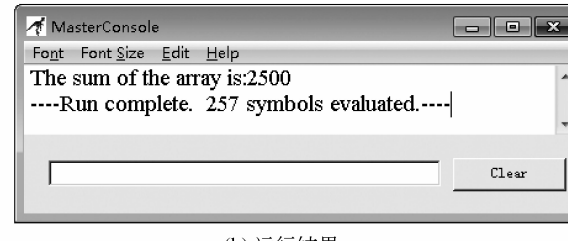
结果形成的二维数字矩阵看起来如图 3-8 所示。

数组的应用有以下注意事项：

(1) 在 RAPTOR 中，一旦某个变量名被用做数组变量，就不允许存在一个同名的非数组变量。读者可以考虑一下，为什么会有这样的规则。



(a) 流程图



(b) 运行结果

图 3-7　100 以内奇数求和的流程图和运行结果

	1	2	3	4
1	0	0	0	0
2	0	0	0	0
3	0	0	0	13

图 3-8　二维数组的创建和初始化

（2）RAPTOR 数组可以在算法运行过程中动态增加数组元素；但不可以将一个一维数组在算法运行中扩展成二维数组。

例 3-2　编写程序计算 m 行 n 列（m 和 n 小于 10）整型数组 a 周边元素之和（即第 1 行、第 m 行、第 1 列、第 n 列上元素之和，但是重复元素只参加 1 次求和）。

解：本例属于二维数组的基本应用问题。可以采用这样的策略求解该问题：按行来计算数组和值，对第 1 行和第 m 行，将整行数据全部累加，即利用循环对这些行的数据全部累加；对于非第 1 行或第 m 行，只需要累加该行第 1 列和第 n 列的值即可，即 sum＝sum＋array[i,1]＋array[i,n]。

除了上述算法外，此题还应该解决以下两个问题：

(1) 数组的输入问题。对于二维数组可以采用循环嵌套的模式来完成数组输入，其实，为了减少用户输入的工作量，还可以采用随机数来给二维数组赋值。后面例题中有对此问题的专门讲解。

(2) 数组的显示输出问题。在计算二维数组周边元素之前，应该将二维数组显示给用户，以便于用户进行最后验证。输出数组也可以采用两层循环嵌套来实现，外层循环控制行，内层循环控制列。

程序中共有 4 个子图：

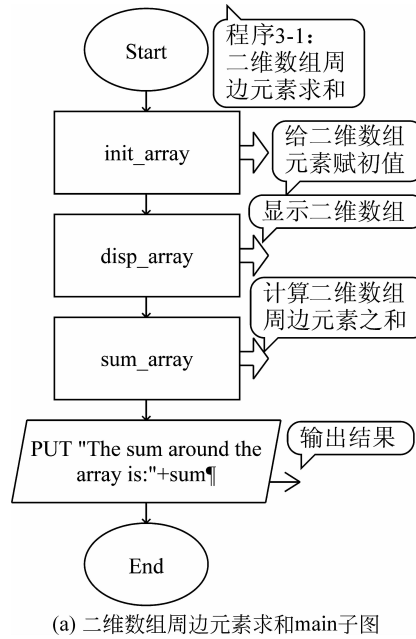
main 子图：主子图，负责程序中子图的调用和程序整体流程的控制，如图 3-9(a)所示。

init_array 子图：主要负责数组元素按照行列赋初值，如图 3-9(b)所示。

disp_array 子图：负责二维数组元素的显示，如图 3-9(c)所示。

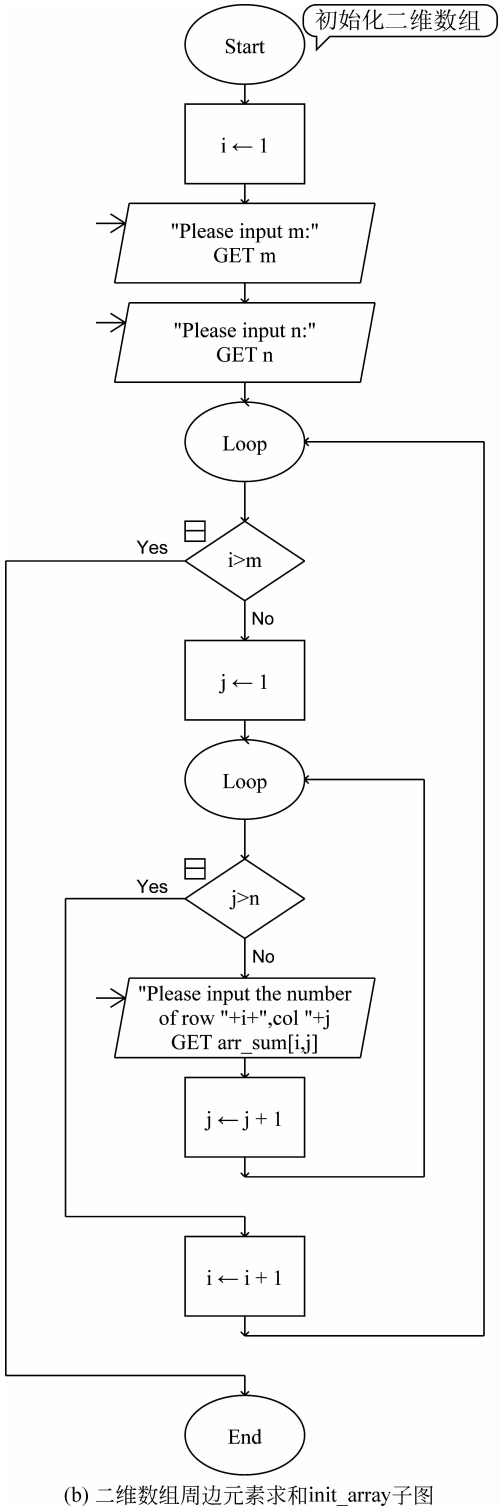
sum_array 子图：负责按照题目要求对二维数组的周边元素进行求和运算，如图 3-9(d)所示。

程序运行结果如图 3-9(e)所示。

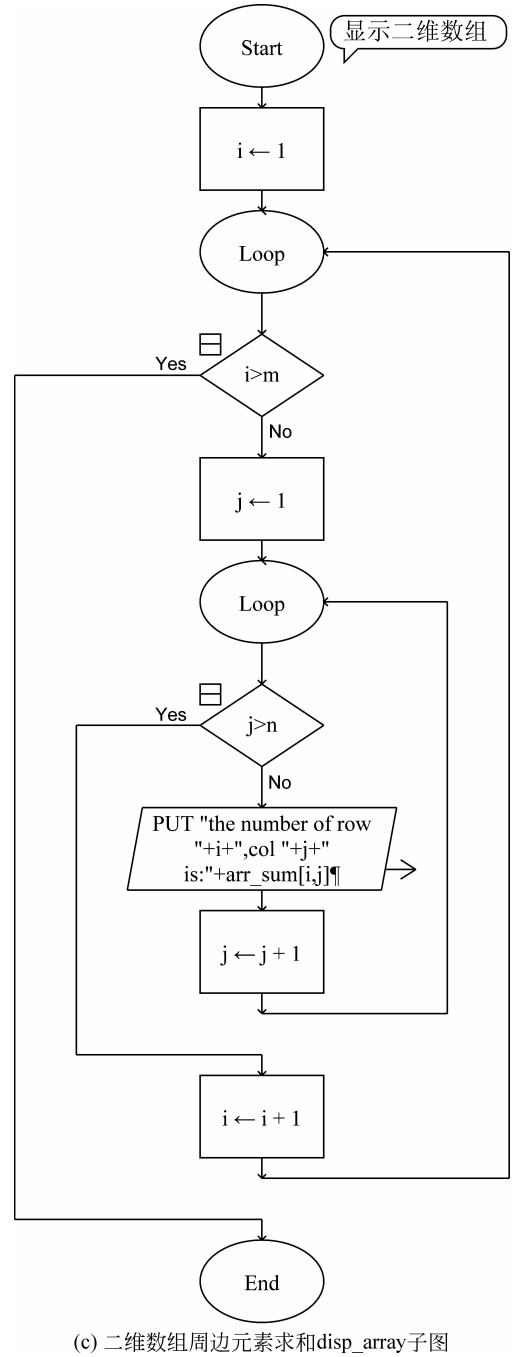


(a) 二维数组周边元素求和main子图

图 3-9　二维数组周边元素求和程序

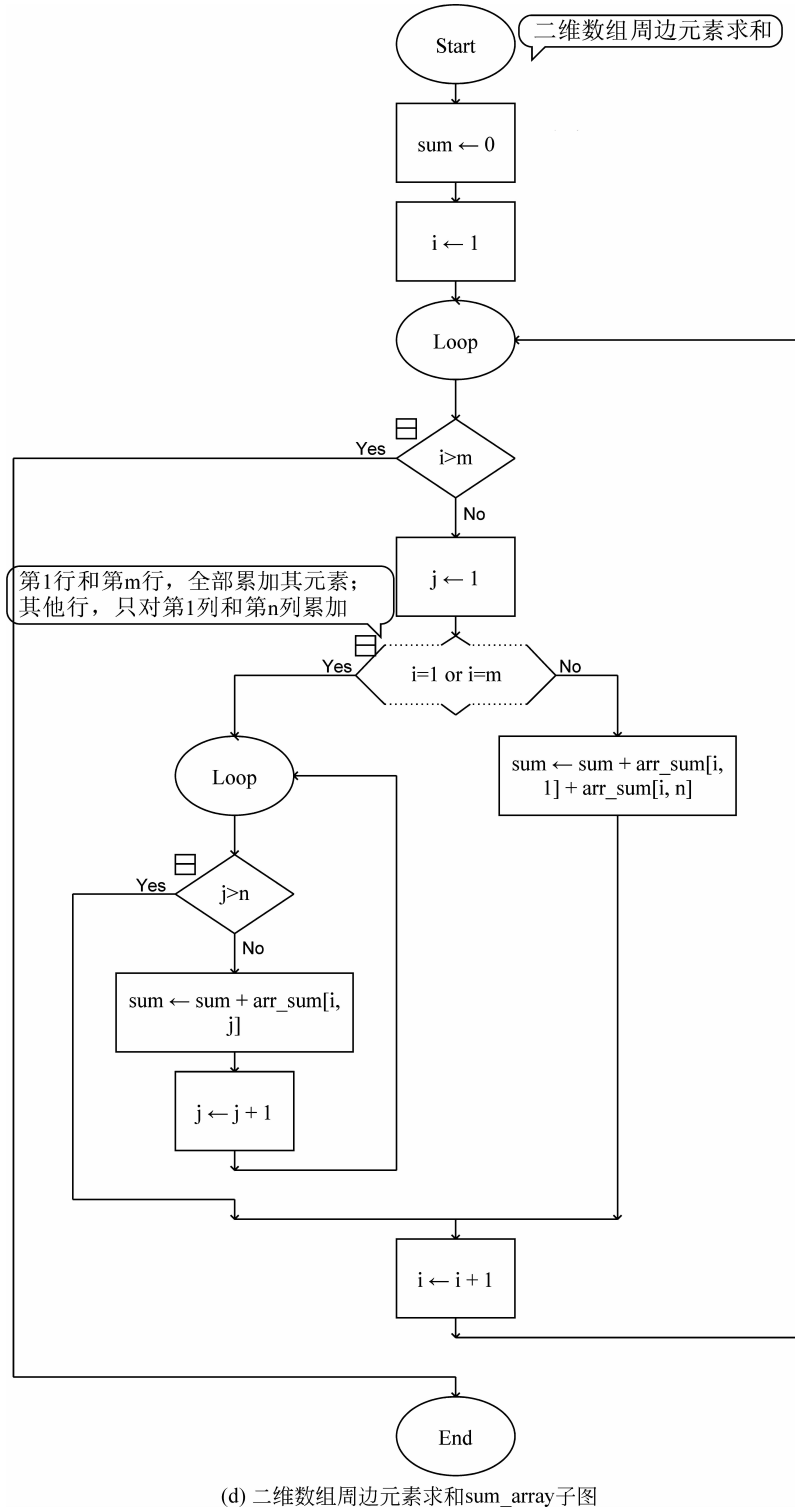


(b) 二维数组周边元素求和init_array子图



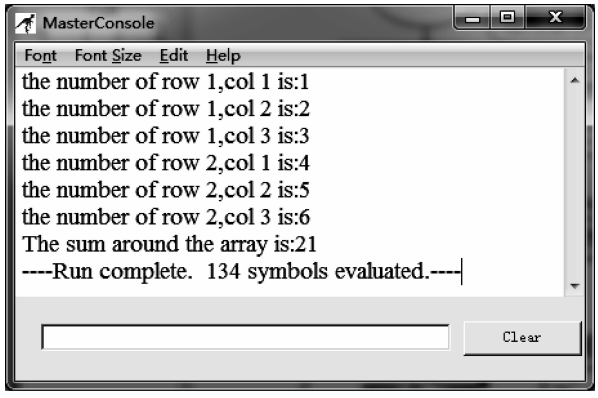
(c) 二维数组周边元素求和disp_array子图

图 3-9 （续）



(d) 二维数组周边元素求和sum_array子图

图 3-9 （续）



(e) 二维数组周边元素求和运行效果

图 3-9 （续）

3.2.3 字符串与字符数组

在 RAPTOR 环境中，字符串扮演着一种特殊的角色。它既能以一个变量的形式在程序中进行传递，又可以作为字符串数组的一个元素，本身还可以看做一个字符型数据的数组。

RAPTOR 中可以直接对整个字符串赋值，而不用像数组一样使用循环语句一个一个地输入；当然也可以使用与数组一样的方式来调取它的第 i 个元素，即第 i 个字符。

字符串的声明与运算在第 2 章中已经提及，这里不再赘述。下面主要介绍字符串作为数组如何应用。

字符串可以使用数组的函数 length_of()来获取字符串的长度，即字符串中的字符个数。若要在字符数组 s 中调取字符串的第 i 个元素，直接使用 s[i]即可。但是需要注意的是，字符串作为数组，它的元素都是字符而不是字符串。例如，图 3-10 中程序运行后，c 的值为'v'而不是"v"（注意单引号与双引号的区别）。

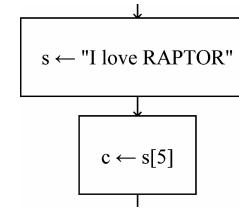


图 3-10 利用数组取字符串中某一元素

利用字符串的数组特性，可以对字符串按每一位来进行操作。

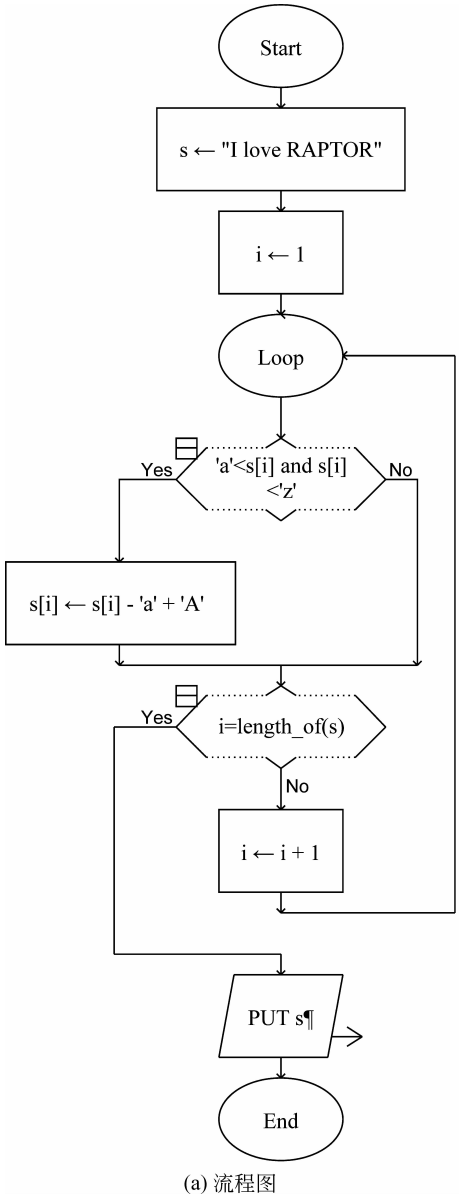
例 3-3 编写程序，将字符串 s="I love RAPTOR"转化为全大写的形式。

程序流程图如图 3-11(a)所示，运行结果如图 3-11(b)所示。

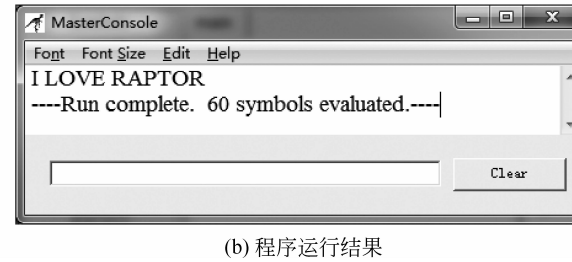
值得注意的是，由于在 RAPTOR 中最多只支持二维数组，利用字符串其实可以进行维数的扩充。原因很简单，字符串自己也是数组。但是存储的数据必须是字符型或者可以转化为字符型的。例如，图 3-12(a)中的程序调用了 getArr 子程序（见图 3-12(b)），实现了数组的维数的扩充。

运行结果为 c='l'。实际上这构造了一个三维数组。c 是(2,2,3)号元素。

若存取的是 0～255 的整数，可以使用 to_ascii()函数与 to_character()函数进行转

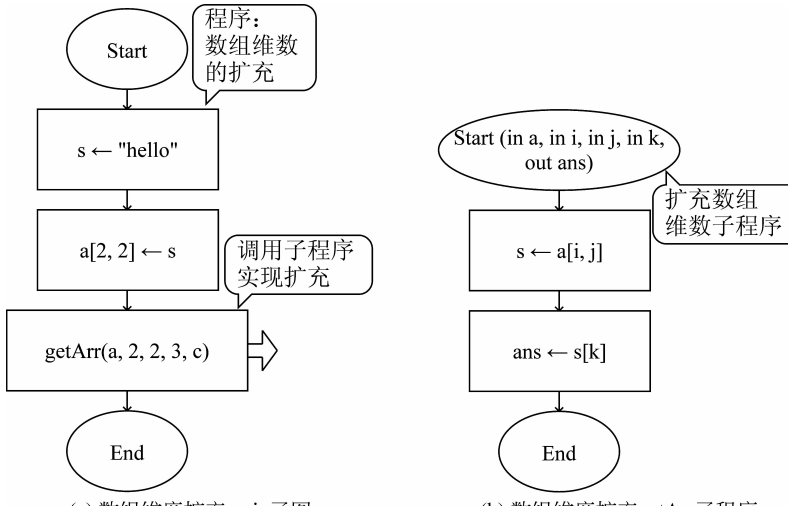


(a) 流程图



(b) 程序运行结果

图 3-11 利用数组将字符串转换为大写的流程图和运行结果



(a) 数组维度扩充main子图　　(b) 数组维度扩充getArr子程序

图 3-12　数组维数扩充

化。若需要存取更大的数，可以采取每两个字节存一个数等方式扩充数的范围。当然，也可以尝试使用一些特殊字符来对字符串进行分隔，以此达到扩充维数的效果，不过需要自己编写转换函数，有兴趣的读者可以试一试。

3.3　数组的其他应用方式

3.3.1　平行数组

如果要计算某个班级学生的一个学期 3 门课程的总成绩和平均成绩，应该如何保存参与计算的课程成绩？当然采用数组，可以分别使用 english[]、computer[]和 math[]，而每个同学的编号可以用作数组的下标。

然而，这是 3 个不同的数组，数组之间并不存在任何必然的联系。但是，我们可以选择用这样的方式使它们产生关联。具体来说，将同一个对象的相关数据存储在各数组相同的下标元素中。采用这种方式存储数据的数组称为“平行数组”。这个简单的发明是一种最自然的、而且能够解决问题的手段。

在程序设计中，有许多情况需要大量的相关数据值，仅使用简单的变量，将会使程序变得非常烦琐或不切实际。数组允许用户使用一个单一的基本名称，结合一个下标，来访问许多不同的变量。每次遇到一个数组变量的引用时，数组下标会进行表达式计算，在大量的循环过程中，每次都会用到下标变量，每次循环都会针对各个不同的下标变量所指向的数组元素进行操作。

3.3.2　多种数据类型元素共存的数组

RAPTOR 的数组有一个非常灵活之处，就是并不强制每个数组的元素必须具备相

同的数据类型，有了这个特点，程序员可以将数组，尤其是二维数组，设计成类似数据库那样的一种记录或字段式结构。参见图 3-13，可以将二维数组的每一行的 3 个元素设计成不同的数据类型。在该程序段中，a[]数组的 3 个元素分别为数值、字符和字符串类型。

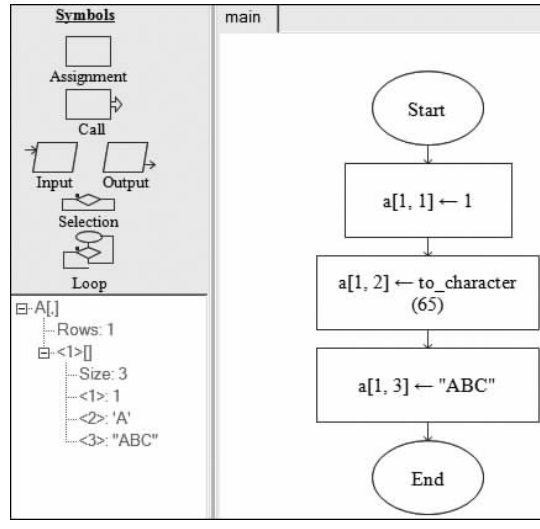


图 3-13　将数组元素设计成为数据库的“字段”形式

数组是用来处理数据集合的有力工具，同时 RAPTOR 在定义数组和表示数组元素时采用 a[]这种方括弧的方式。数组初始化十分方便，它的使用方式也非常灵活，为后续的算法设计提供了基础条件。

例 3-4　从键盘输入 3 个学生的信息，这些信息包括学号、姓名、英语、计算机、数学、总分和平均分，其中总分和平均分不需要输入，通过程序自动计算。最后按总分从高到低依次输入 3 个学生的相关信息。

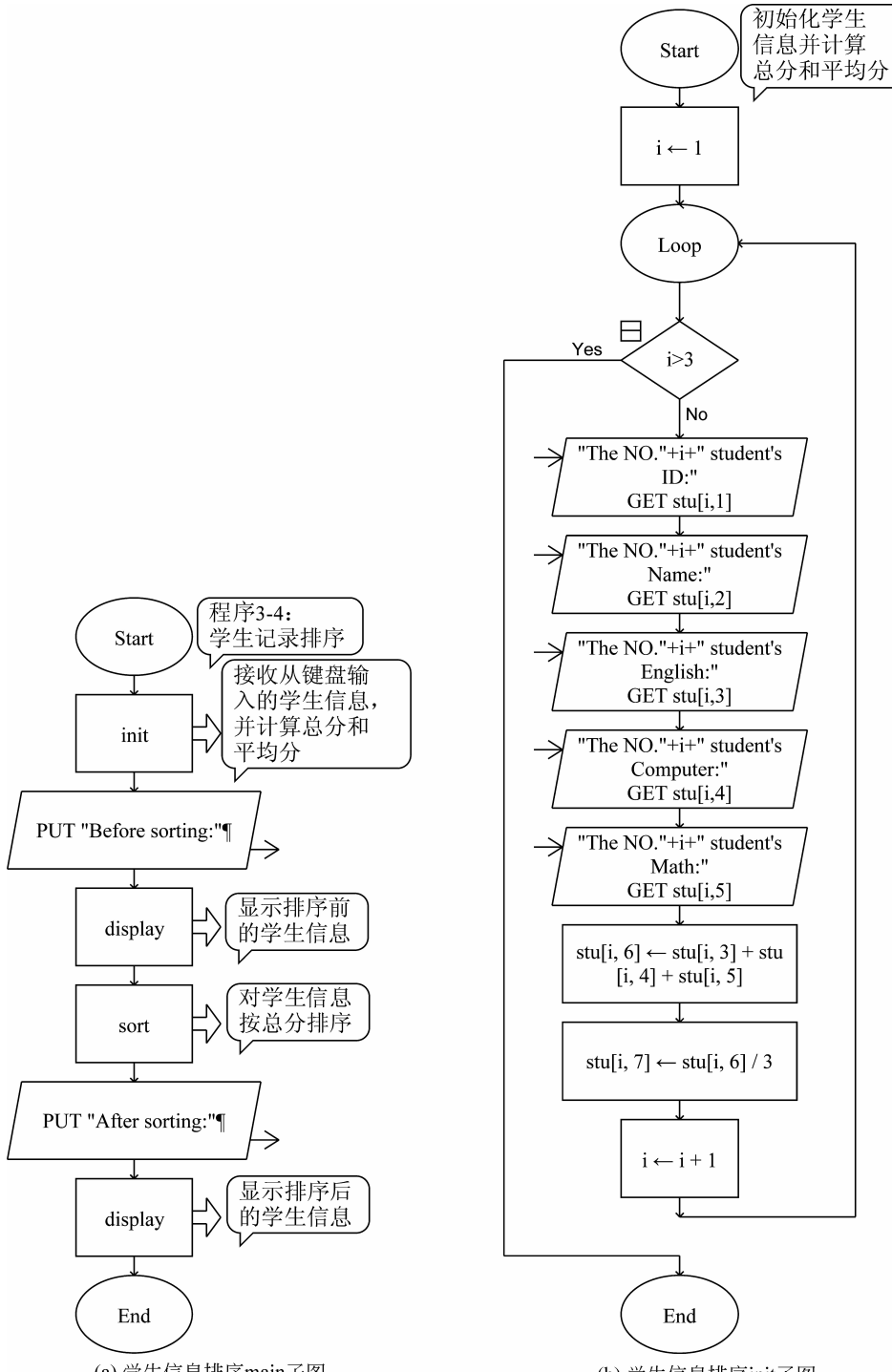
解：程序中因为每个学生的基本信息都需要从键盘输入，为了能够把问题说清楚，又不至于带来大量的输入工作，所以仅选用 3 个学生作为对象，实际应用中，学生的数量也可由用户从键盘指定。要进行相关输入和计算，先必须确定存储学生信息的数据结构，由于同一个 RAPTOR 数组中可以存放不同类型的数据，因此这里选择二维数组存放学生的相关信息。数组中第 1 个下标表示学生，第 2 个下标表示该学生的信息，例如：stu[3,5]表示第 3 个学生的数学成绩，stu[5,2]表示第 5 个学生的姓名。

学生的总分可以用 3 门课的成绩相加求得，平均分可以用总分除以 3 求得。

可以采用两种方式完成总分的排序输出，一种方式是排序后在数组中交换参与排序对象的信息，然后直接按数组下标输出结果，采用这种将会改变学生信息在数组中的存放位置，本例中采用这种方法完成按总分排序输出；另一种方式是不实际交换数组相关元素，只需记录总分由高到低的学生编号，并按此编号输出，这种方式不会改变学生信息在数组中的存放位置。

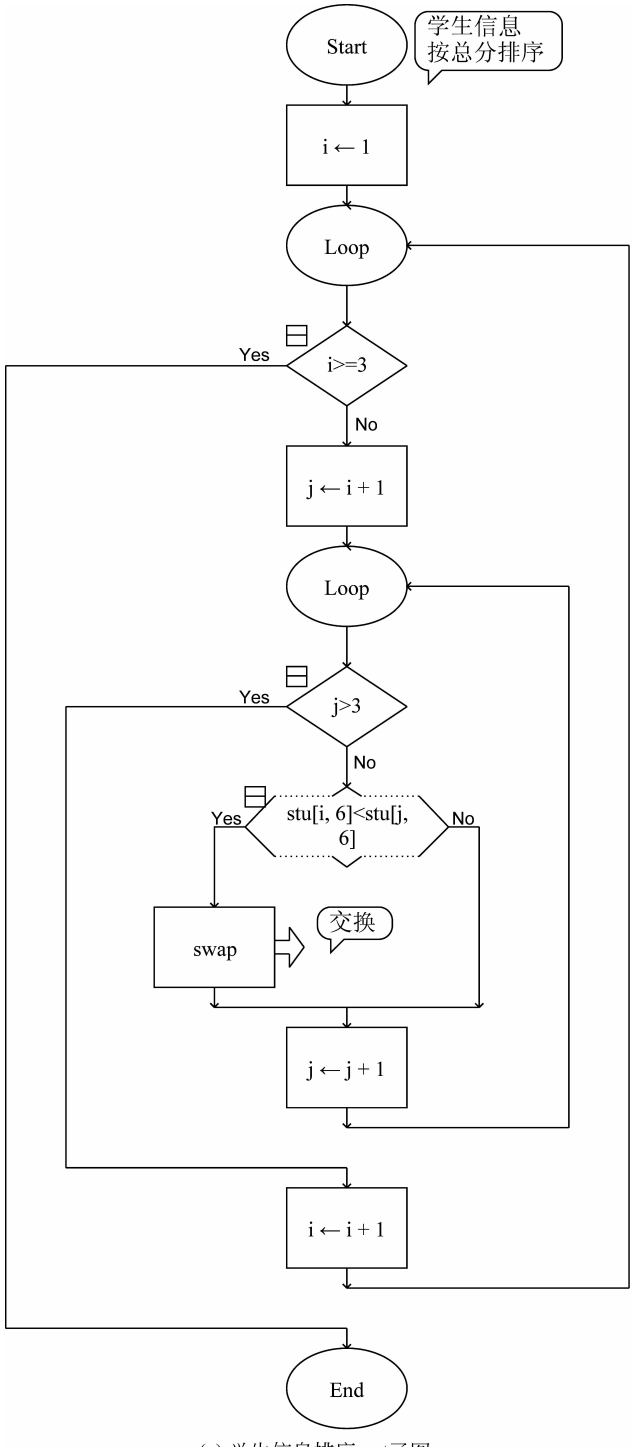
程序中共有 5 个子图。

main 子图：主子图，负责程序中子图的调用和程序整体流程的控制，如图 3-14(a)所示。

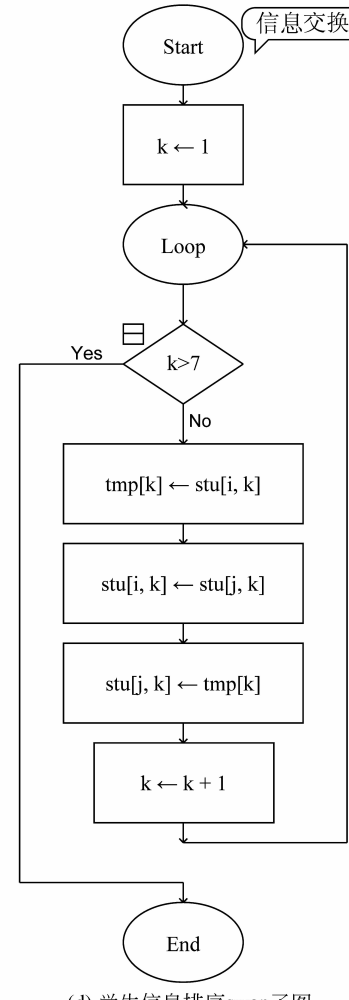


(a) 学生信息排序main子图　　(b) 学生信息排序init子图

图 3-14　例 3-4 各子图和程序运行结果

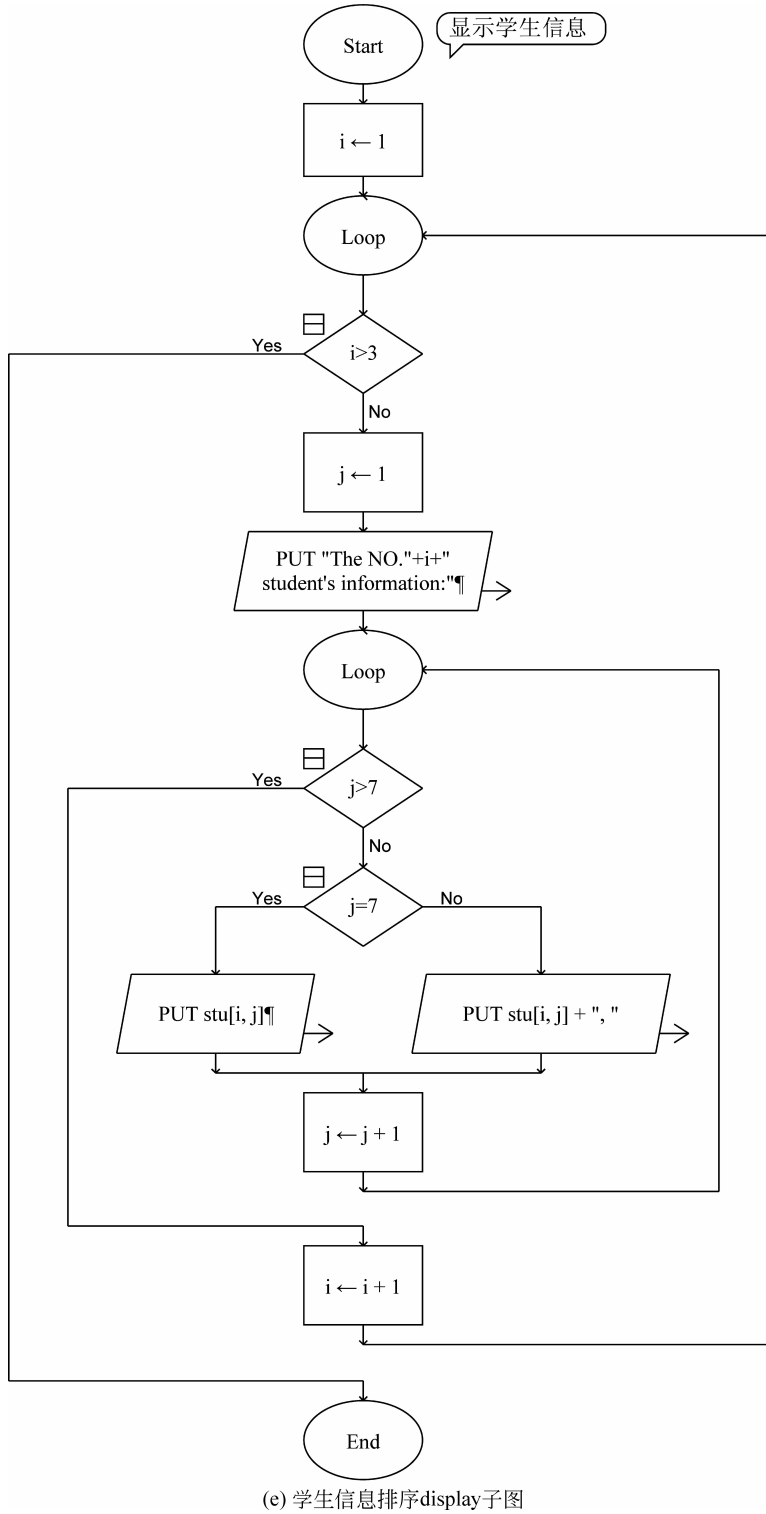


(c) 学生信息排序sort子图



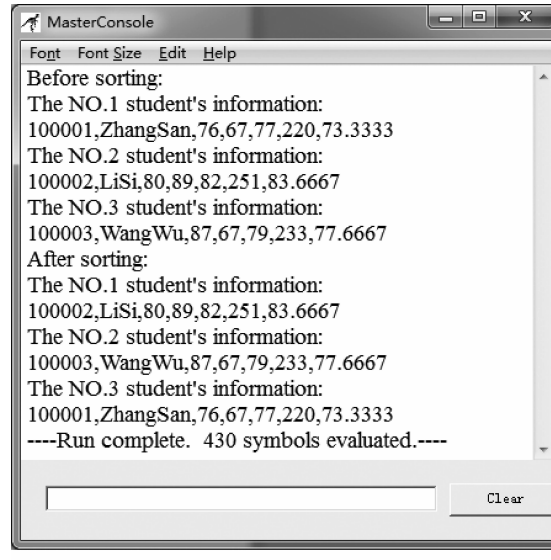
(d) 学生信息排序swap子图

图 3-14 （续）



(e) 学生信息排序display子图

图 3-14 （续）



(f) 学生信息排序运行结果

图 3-14 （续）

init 子图：主要负责学生信息数组信息的录入，并根据录入的信息计算每个学生的总分和平均分，如图 3-14 (b)所示。

sort 子图：实现学生总分由高到低排序，如图 3-14(c)所示。

swap 子图：在排序过程中，负责两个比较数据的交换，如图 3-14(d)所示。

display 子图：用于显示学生的信息，如图 3-14(e)所示。

程序运行结果如图 3-14(f)所示。

3.4 数组的应用案例

3.4.1 使用随机数产生数组的元素并输出

在计算机上用数学方法产生随机数列是目前产生算法所需数据的常用方法，它的特点是占用的内存少，速度快。

由于数列是用算法产生的，因而本质上是决定性的，再加上计算机字长有限，所以无论用什么算法产生的数列，在统计特征上都不可能与从均匀分布中抽样所得的子样完全相同，因而只能要求尽可能地近似。

这种在计算机上用算法得到的统计性质上近似于[0,1)上均匀分布的数，一般就称为**伪随机数**，以区别于真正抽样所得到的[0,1)上均匀分布的随机数。

例 3-5 请设计一个随机数产生程序，并将结果保存在一维数组中。

解：图 3-15 显示了求 8 个随机数并保存到数组中的流程图和运行结果。

在算法设计中，随机数的主要用途如下：

(1) 产生算法所需要的数据，例如，排序、查找算法需要大量基础数据进行算法检验，而随机数符合算法应用的大部分场合。

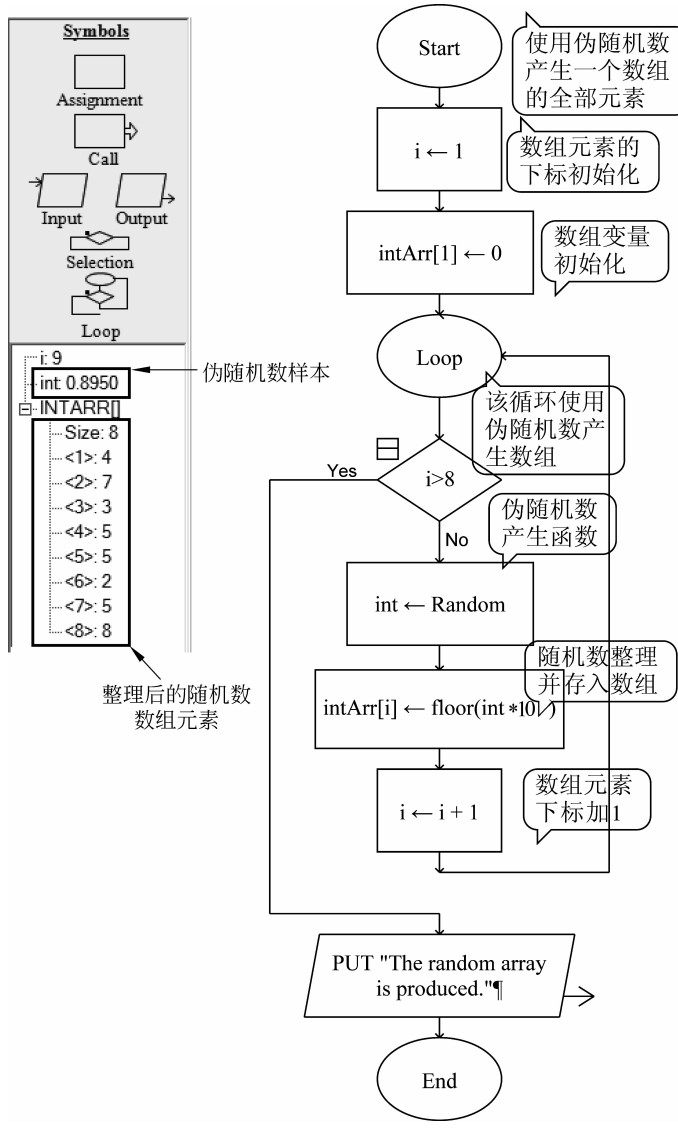


图 3-15　随机数的产生、处理和存储

（2）一些随机模拟算法需要的基础数据，例如随机漫步模型的模拟算法。

（3）减少不必要的人机交互，例如，要求用户输入 10 个数据，进行最大、最小值的查找等。

随机数使用的注意事项如下：

（1）由于随机数只有[0,1)区间的小数，所以需要对计算机产生的随机数进行加工，才能获得算法所需要的整数。在 RAPTOR 中，可以采用将随机数函数 Random 乘以 10 的倍数，再使用向下取整函数 floor()和向上取整函数 ceiling()来获取相应范围内的随机整数。

（2）需要获取 ASCII 码表中的数值，可以使用模除运算，如（Random * 1000 mod

128)可能得到全部的标准 ASCII 码值(0～127)。

(3) 由于 RAPTOR 的数值默认精度只有 4 位小数,所以,部分随机数结果可能为 0.0000,经过处理得到的整数就是 0,所以,在不希望出现 0 的场合,必须对随机函数得出的结果进行检验,去除不希望得到的值。

3.4.2 模拟掷骰子

本例模拟一个骰子投掷 n 次所得的结果并分别统计输出。

例 3-6 使用随机数函数可以模拟许多自然现象和社会行为,掷骰子就是一个典型例子。请设计一个掷骰子的程序,并将掷出的结果分点数保存在一个有 6 个元素的数组中。

解: 在图 3-16 中,使用 i 作为循环计数器,intarr 数组直接利用数组的下标(1～6)进

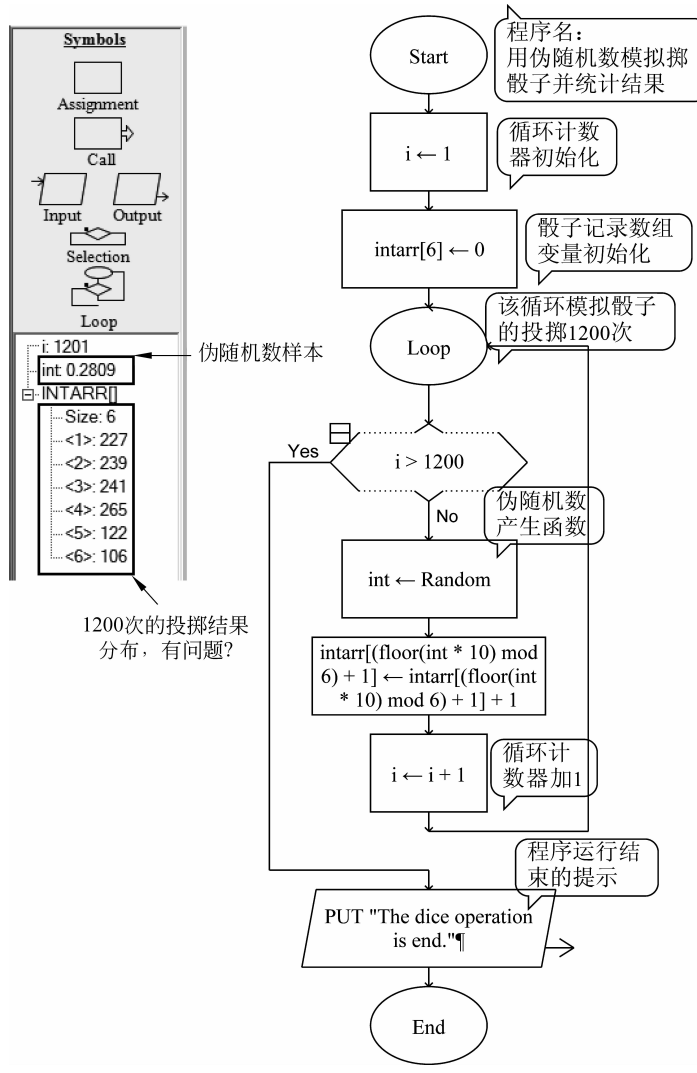


图 3-16 骰子投掷 1200 次所得的结果

行 1200 次掷骰子的模拟，随机数的整理公式为：(floor(int * 10) mod 6)+1，这是由于随机函数会输出 0 的结果，这在本例中是不可取的。

图 3-16 左侧列出了 1200 次投掷的结果，读者可以将 i 计数器的循环控制扩展到更大的次数。但是，得出的结果似乎与我们的常识大相径庭，这个问题出在哪？

实际上，这个程序的问题出在随机数整理公式上，(floor(int * 10) mod 6)+1 在运算中会发生重大偏差，读者只需将该公式改为(floor(int * 100) mod 6)+1 或(floor(int * 1000) mod 6)+1 就会大大改善程序的运算结果。

这给我们一个提示，就是伪随机数的输出结果是相对“随机的”，如果算法设计的处理方式不对，会使伪随机函数的输出结果的“随机性”大打折扣。

3.4.3 使用随机数模拟井字棋

井字棋，英文名为 Tic-Tac-Toe，是一种在 3×3 格子上进行的连珠游戏，和五子棋类似，因棋盘一般不画边框，格线排成井字而得名。游戏需要的工具仅为纸和笔，然后由两个游戏者轮流在格子里留下○和×标记(一般来说，先手者为×)。由最先在任意一条直线上(横向、纵向或对角线方向)成功连接 3 个标记的一方获胜。例如，图 3-17 所示的游戏中×方获胜。

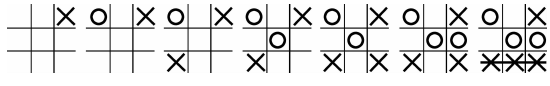


图 3-17 井字棋的游戏规则示意

玩过这个游戏的人会发现，如果两个玩家都做出最好的选择，这个游戏一定会平局。所以，井字棋通常作为儿童游戏。

例 3-7 请设计一个程序，使用随机数的方法向井字棋盘各格中填入×或○。

解：由于井字棋游戏在一个 3×3 的网格中填入×和○，因此可以使用一个 3×3 的二维数组来保存相应网格中的数据。本例使用随机数来确定向井字棋盘各格中填入×还是○，并将其填入到 a[,]数组中，其流程图和运行结果参见图 3-18。

尽管没有博弈过程，还是可以将这样的填充结果看成博弈产生的结果。读者可以考虑设计一个程序来判断某次填充的结果哪一方为胜方，或者是一个和局。

3.4.4 凯撒密码与字符串运算

在程序设计中涉及的地方较多，尤其在网络信息传播的过程中，字符和字符串运算扮演了重要的角色。而字符串运算与数组运算具有许多共性，例如，在 RAPTOR 中，length_of()函数既可以统计一维数组元素的个数，也可以统计字符串的长度。

归纳起来，字符和字符串运算在算法中的用途如下：

(1) 在输入输出界面中的应用。例如，在输入过程中提示用户输入变量的值，在输出过程中将计算结果与单位结合在一起向用户展示。

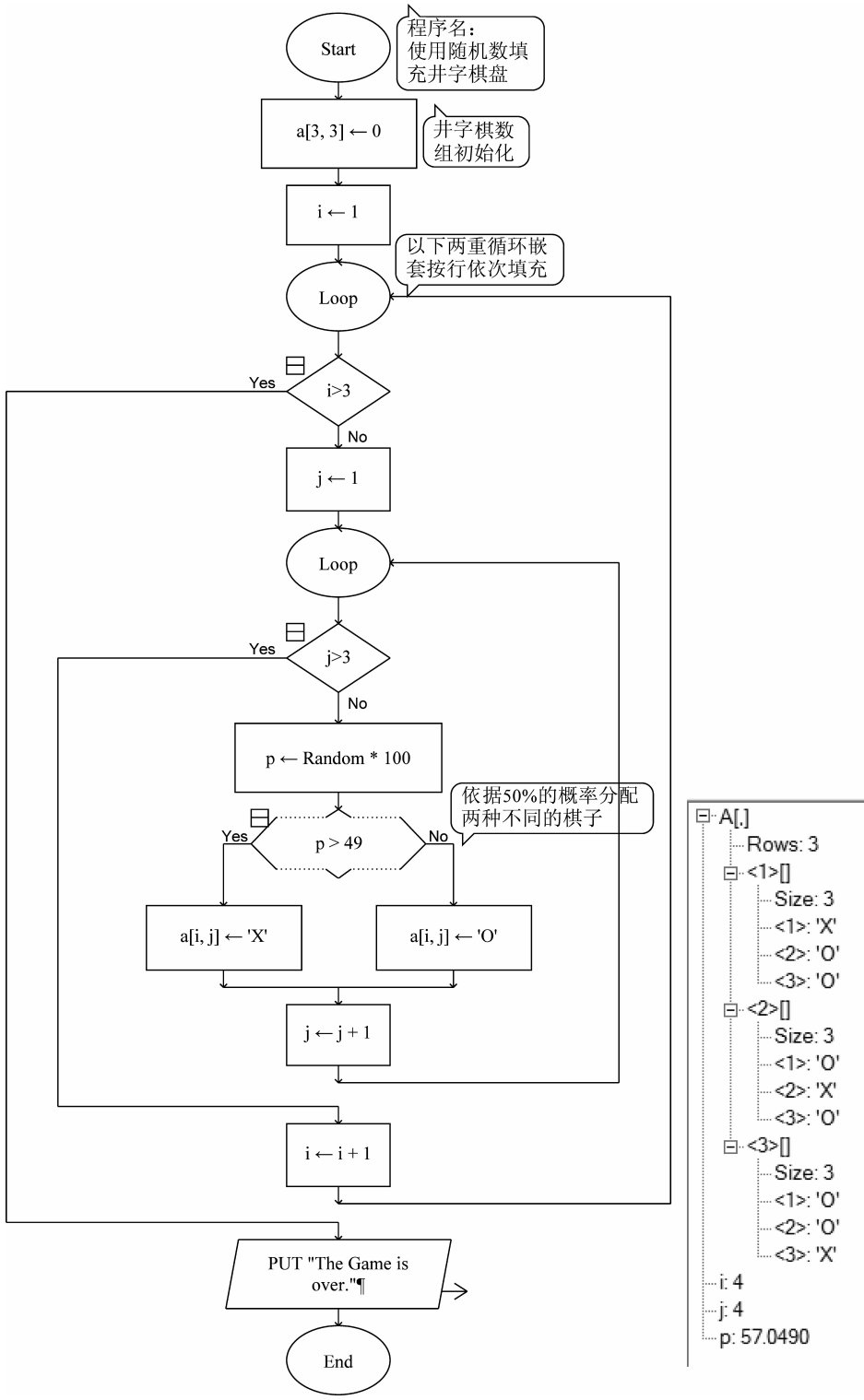


图 3-18　使用随机数填充井子棋盘的一个流程图与结果样例