每隔 1 米就任意放 1 件物品。请找出这些物品中有没有苹果。

诸位每前进 1 米就要捡起地上的物品，检查是否拿到了苹果，同时还要检查有没有到达悬崖的边缘（不检查的话就有可能掉到海里）。也就是说要对这两种检查反复若干次。

使用了哨兵以后，就要先把起点挪到距悬崖边缘 101 米的地方，再在悬崖的边缘放置一个苹果（如图 5.8 所示）。这个苹果就是哨兵。通过放置哨兵，诸位就一定能找到苹果了。每前进 1 米时只需检查捡到的物品是不是苹果就可以了。发现是苹果以后，只需站在原地再检查一步开外的情况。如果还没有到达悬崖边缘，就意味着找到了真正要找的苹果。已经达到了悬崖边缘，则说明现在手中的苹果是哨兵，而没有找到真正要找的苹果。

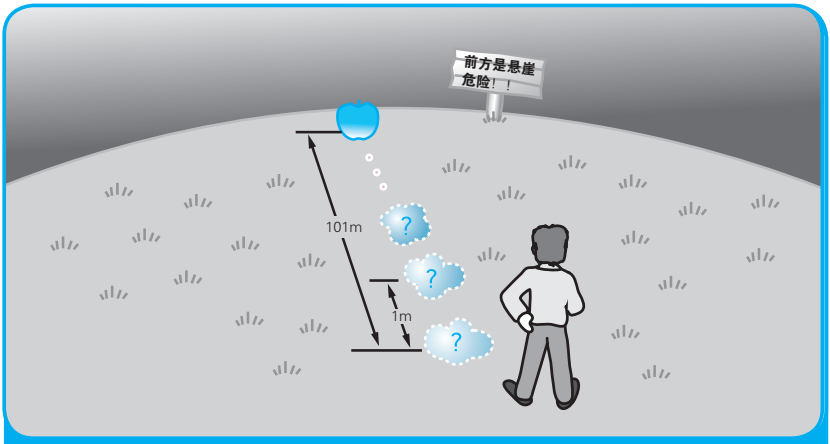

图 5.8　使用了哨兵的游戏

5.7 要点 6：找出数字间的规律

所有的信息都可以用数字表示——这是计算机的特性之一。因此为了构造算法，经常会利用到存在于数字间的规律。例如，请思考一下判定石头剪刀布游戏胜负的算法。如果把石头、剪刀、布分别用数字 0、1、2 表示，把玩家 A 做出的手势用变量 A 表示，玩家 B 做出的手势用变量 B 表示，那么变量 A 和 B 中所存储的值就是这三个数中的某一个。请以此判断玩家 A 和 B 的输赢。

如果算法没有使用任何技巧，也许就会通过枚举表 5.2 中所列出的 3×3 = 9 种组合来判断输赢吧。把这个表格转换成程序后就得到了代码清单 5.4 中的代码。可以看出这是一种冗长而又枯燥的判断方法（代码清单 5.4 和 5.5 列出的都只是程序的一部分，因此不能直接运行）。

表 5.2　判定石头剪刀布输赢的表

变量 A 的值	变量 B 的值	判定结果
0（石头）	0（石头）	平局
0（石头）	1（剪刀）	玩家 A 获胜
0（石头）	2（布）	玩家 B 获胜
1（剪刀）	0（石头）	玩家 B 获胜
1（剪刀）	1（剪刀）	平局
1（剪刀）	2（布）	玩家 A 获胜
2（布）	0（石头）	玩家 A 获胜
2（布）	1（剪刀）	玩家 B 获胜
2（布）	2（布）	平局

代码清单 5.4　判断石头剪刀布输赢的程序（方法一）

```
If (A = 0) And (B = 0) Then
    MsgBox "平局"
ElseIf (A = 0) And (B = 1) Then
    MsgBox "玩家A获胜"
ElseIf (A = 0) And (B = 2) Then
```

```
    MsgBox "玩家B获胜"
ElseIf (A = 1) And (B = 0) Then
    MsgBox "玩家B获胜"
ElseIf (A = 1) And (B = 1) Then
    MsgBox "平局"
ElseIf (A = 1) And (B = 2) Then
    MsgBox "玩家A获胜"
ElseIf (A = 2) And (B = 0) Then
    MsgBox "玩家A获胜"
ElseIf (A = 2) And (B = 1) Then
    MsgBox "玩家B获胜"
ElseIf (A = 2) And (B = 2) Then
    MsgBox "平局"
End If
```

接下来就试着在此之上稍微加入些技巧吧。请仔细观察表 5.2 并找出数字间的一种规律，这个规律可以简单地判定出是玩家 A 获胜，玩家 B 获胜，还是平局这三种结果。可能需要习惯一下思维上的转变，但最终应该都可以发现如下的规律。

- 如果变量 A 和 B 相等就是“平局”
- 如果用 B + 1 除以 3 得到的余数与变量 A 相等就是“玩家 B 获胜”
- 其余的情况都是“玩家 A 获胜”

用程序来表示这个规律就得到了如代码清单 5.5 所示的代码。与没有使用任何技巧的代码清单 5.4 中的代码相比，可以发现处理过程简单并且代码短小精悍。当然程序的执行速度也会随之提升。

代码清单 5.5　判断石头剪刀布输赢的程序（方法二）

```
If A = B Then
    MsgBox "平局"
ElseIf A = (B + 1) Mod 3 Then
    MsgBox "玩家B获胜"
Else
    MsgBox "玩家A获胜"
End If
```

构造算法时需要找出数字间的规律不仅适用于数学游戏，编写用于计算工资的应用程序时，计算工资的规则也可以说是一种数字上的规律。如果能够发现“工资 = 底薪 + 加班补贴 + 交通补贴 – 预扣税款”这样的规律，那么解决问题的步骤就是明确的，步骤数也是有限的，因此构造出的算法也就是优秀的了。

5.8 要点 7：先在纸上考虑算法

最后介绍最为重要的一点，那就是思考算法的时候，要先在纸上用文字或图表描述出解决问题的步骤，而不要立刻开始编写代码。

画流程图就可以方便地把算法用图表示出来，因此请诸位大量地、灵活地运用它。如果不想画流程图，也可以用语言把算法描述出来，写成文书。总之先写到纸上这一点很重要。

在纸上画完或写完流程以后，再把具体的数据代入以跟踪流程的处理，确认是否能得到预期的结果。在验算的时候，建议使用简单的数据，这样即使是用心算也能得出正确的结果。例如，要确认辗转相除法的流程，就可以使用数值较小的数做验算，这样就算是用中学所学的求解步骤也能求出最大公约数。如果使用的是数值较大的数，比如 123456789 和 987654321（最大公约数是 9），那么就难跟踪流程的处理了。

☆　　☆　　☆

曾经有一本被誉为凡是立志成为程序员的人都应该去读的名著，那就是 Niklaus Wirth 的 *Algorithms + Data Structures = Programs*。

要在网上搜索这本书的话，会查到一本又一本地以“算法和数据结

构”为主题的书，总共有数十本。看这些书名就可知道，如果只了解算法，实际上关于编程的知识是不完整的，因此还必须要考虑和算法相辅相成的数据结构。在接下来的第 6 章中，笔者将会讲解数据结构。敬请期待！

第6章 与数据结构成为好朋友的七个要点

热身问答

在阅读本章内容前，让我们先回答下面的几个问题来热热身吧。

初级问题

程序中的变量是指什么？

中级问题

把若干个数据沿直线排列起来的数据结构叫作什么？

高级问题

栈和队列的区别是什么？

怎么样？被这么一问，是不是发现有一些问题无法简单地解释清楚呢？下面，笔者就公布答案并解释。

初级问题：变量是数据的容器。

中级问题：叫作“数组”。

高级问题：栈中数据的存取形式是 LIFO；队列中数据的存取形式是 FIFO。

解释

初级问题：变量中所存储的数据是可以改变的。变量的实质是按照变量所存储数据的大小被分配到的一块内存空间。

中级问题：使用了数组就可以高效地处理大量的数据。数组的实质是连续分配的一块特定大小的内存空间。

高级问题：LIFO（Last In First Out，后进先出）表示优先读取后存入的数据；FIFO（First In First Out，先进先出）表示优先读取先存入的数据。本章将会详细地讲解栈和队列的结构。

本章重点

在第 5 章中笔者曾经这样介绍过算法：程序是用来在计算机上实现现实世界中的业务和娱乐活动的，为了达到这个目的，程序员们需要结合计算机的特性，用程序来表示现实世界中对问题的处理步骤，即处理流程。本章的主题是数据结构，也就是如何结合计算机的特性，用程序来表示现实世界中的数据结构。

程序员有必要把算法（处理问题的步骤）和数据结构（作为处理对象的数据的排列方式）两者放到一起考虑。选用的算法和数据结构两者要相互匹配这一点很重要。本章会依次讲解以下 3 点：数据结构的基础、最好先记忆下来的典型数据结构以及如何用程序实现典型的数据结构。范例代码全部由适合于学习算法和数据结构的 C 语言编写。为了让即便不懂 C 语言的读者也能读懂，笔者会采取简单易懂的说明，所以请诸位不要担心。另外，为了易于理解，文中只展示了程序中的核心片段，省略了错误处理等环节，这一点还要请诸位谅解。

6.1 要点 1：了解内存和变量的关系

计算机所处理的数据都存储在了被称为内存的 IC（Integrated Circuit，集成电路）中。在一般的个人计算机中，内存内部被分割成了若干个数据存储单元，每个单元可以存储 8 比特的数据（8 比特 = 1 字节）。为了区分各个单元，每个单元都被分配了一个编号，这个编号被称为“地址”或是“门牌号码”。如果一台个人计算机装配有 64M 字节的内存，那么就会有从 0 到 64M（M = 100 万）这么多个地址。

因为依靠指定地址的方式编写程序很麻烦，所以在 C 语言、Java、BASIC 等几乎所有的编程语言中，都是使用变量把数据存储进内存，

或从内存中把数据读出来的。在代码清单 6.1 中列出了一段用 C 语言写的程序，用于把数据“123”存入变量 a 中。其中用“/*”和“*/”括起来的内容是 C 语言的注释。

代码清单 6.1　把 123 存入变量 a

```
char a;    /* 定义变量 */
a = 123;   /* 把数据存入变量 */
```

首先请看后面注释有“定义变量”的这一行代码“char a;”。“char”代表一种 C 语言的数据类型，该类型可用于存储 1 字节的整数。通过这一行代码就在内存中预留了一块空间，并为这块空间起了个名字叫作 a。

对于程序员来说，他们并不需要知道变量 a 被存储到内存空间中的哪个地址上了。因为当程序运行时是由操作系统为我们从尚未使用的内存空间中划分出一部分分配给变量 a 的。如图 6.1 所示，变量是程序中数据存储的最小单位，每个变量都对应着一块物理上的内存空间。

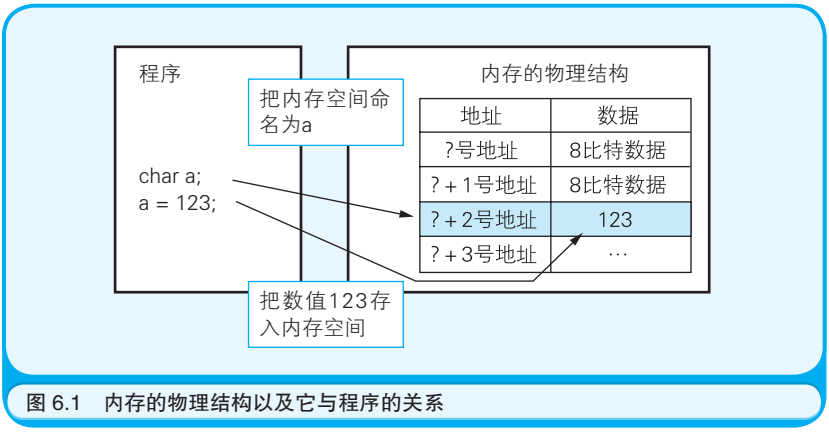

图 6.1　内存的物理结构以及它与程序的关系

如果是完全不了解数据结构的程序员，说不定会通过一个挨一个

地定义出若干个离散的变量来编写程序吧。要是程序可以按照预期运行，那么以这种方式编程倒也可以。但是若还要用这种方式实现对多个数据排序的算法，那就有些困难了。

代码清单 6.2 中列出了一段程序，把三个数据分别存入 a、b、c 三个变量中，再将 a、b、c 中的数据的值按照降序（从大到小的顺序）排列。在排序时为了交换两个变量的值还需要用到 tmp 变量。程序使用 if 语句一对儿一对儿地比较变量的大小，并根据比较的结果交换变量的值。

代码清单 6.2　把存入到三个变量中的数值按照降序排列

```
/* 定义变量 */
char a, b, c, tmp;

/* 把数据存入变量 */
a = 123;
b = 124;
c = 125;

/* 按降序排列 */
if (b > a){
    tmp = b;
    b = a;
    a = tmp;
}

if (c > a){
    tmp = c;
    c = a;
    a = tmp;
}

if (c > b){
    tmp = c;
    c = b;
    b = tmp;
}
```