

用存储在栈中。编译器可以优化尾递归以减小栈空间。

通常,可以使用辅助参数将非尾递归方法转换为尾递归方法。这些参数被用于保存结果。思路是将后续的操作以一种方式结合到辅助参数中,这样递归调用中将不再有后续操作。可以定义一个带辅助参数的新的辅助递归方法,这个方法可以重载原始方法,具有相同的名字而签名不同。例如,程序清单 18-1 中的 `factorial` 方法可以写成尾递归形式,如代码清单 18-10 所示。

程序清单 18-10 `ComputeFactorialTailRecursion.java`

```
1 public class ComputeFactorialTailRecursion {
2     /** Return the factorial for a specified number */
3     public static long factorial(int n) {
4         return factorial(n, 1); // Call auxiliary method
5     }
6
7     /** Auxiliary tail-recursive method for factorial */
8     private static long factorial(int n, int result) {
9         if (n == 0)
10            return result;
11        else
12            return factorial(n - 1, n * result); // Recursive call
13    }
14 }
```

第一个 `factorial` 方法(第 3 行)只是简单调用了第二个辅助方法(第 4 行)。第二个方法包括了一个辅助参数 `result`,它存储了 `n` 的阶乘的结果。这个方法在第 12 行被递归地调用。在调用返回之后,就没有了后续的操作。最终的结果在第 10 行返回,它也是在第 4 行调用 `factorial(n,1)` 的返回值。

复习题

18.30 指出本章中的尾递归方法。

18.31 使用尾递归重写程序清单 18-2 中的 `fib` 方法。

关键术语

base case (基础情况)

direct recursion(直接递归)

indirect recursion(间接递归)

infinite recursion(无限递归)

recursive helper method(递归辅助方法)

recursive method(递归方法)

stopping condition(终止条件)

tail recursion(尾递归)

本章小结

1. 递归方法是一个直接或间接调用自己的方法。要终止一个递归方法,必须有一个或多个基础情况。
2. 递归是程序控制的另外一种形式。本质上它是没有循环控制的重复。对于用其他方法很难解决而本质上是递归的问题,使用递归可以给出简单、清楚的解决方案。
3. 为了进行递归调用,有时候需要修改原始方法使其接收附加的参数。为达到这个目的,可以定义递归辅助方法。
4. 递归需要相当大的系统开销。程序每调用一个方法一次,系统必须给方法中所有的局部变量和参数分配空间。这就要消耗大量的内存,并且需要额外的时间来管理这些内存。
5. 如果从递归调用返回时没有后续的操作要完成,这个递归的方法就称为尾递归(tail recursive)。某些编译器会优化尾递归以减少栈空间。

测试题

回答本章位于 www.cs.armstrong.edu/liang/intro10e/quiz.html 的测试题。

编程练习题

18.2 ~ 18.3 节

*18.1 (计算阶乘) 使用 10.9 节介绍的 `BigInteger` 类, 求得大数字的阶乘 (例如, $100!$)。使用递归实现 `factorial` 方法。编写一个程序, 提示用户输入一个整数, 然后显示它的阶乘。

*18.2 (斐波那契数) 使用迭代改写程序清单 18-2 中的 `fib` 方法。

提示: 不使用递归来计算 `fib(n)`, 首先要得到 `fib(n-2)` 和 `fib(n-1)`。设 `f0` 和 `f1` 表示前面的两个斐波那契数, 那么当前的斐波那契数就是 `f0+f1`。这个算法可以描述为如下所示:

```
f0 = 0; // For fib(0)
f1 = 1; // For fib(1)

for (int i = 1; i <= n; i++) {
    currentFib = f0 + f1;
    f0 = f1;
    f1 = currentFib;
}
// After the loop, currentFib is fib(n)
```

编写一个测试程序, 提示用户输入一个索引, 然后显示它的斐波那契数。

*18.3 (使用递归求最大公约数) 求最大公约数的 `gcd(m,n)` 方法也可以如下递归地定义:

- 如果 $m \% n$ 为 0, 那么 `gcd(m,n)` 的值为 `n`。
- 否则, `gcd(m,n)` 就是 `gcd(n, m%n)`。

编写一个递归的方法来求最大公约数。编写一个测试程序, 提示用户输入两个整数, 显示它们的最大公约数。

18.4 (对数列表求和) 编写一个递归方法来计算下面的级数:

$$m(i) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{i}$$

编写一个测试程序, 为 $i=1, 2, \dots, 10$ 显示 `m(i)`。

18.5 (对数列表求和) 编写一个递归的方法来计算下面的级数:

$$m(i) = \frac{1}{3} + \frac{2}{5} + \frac{3}{7} + \frac{4}{9} + \frac{5}{11} + \frac{6}{13} + \cdots + \frac{i}{2i+1}$$

编写一个测试程序, 为 $i=1, 2, \dots, 10$ 显示 `m(i)`。

*18.6 (对数列表求和) 编写一个递归的方法来计算下面的级数:

$$m(i) = \frac{1}{2} + \frac{2}{3} + \cdots + \frac{i}{i+1}$$

编写一个测试程序, 为 $i=1, 2, \dots, 10$ 显示 `m(i)`。

*18.7 (斐波那契数列) 修改程序清单 18-2, 使程序可以找出调用 `fib` 方法的次数。

提示: 使用一个静态变量, 每当调用这个方法时, 该变量就加 1。

18.4 节

*18.8 (以逆序输出一个整数中的数字) 编写一个递归方法, 使用下面的方法头在控制台上以逆序显示一个 `int` 型的值:

```
public static void reverseDisplay(int value)
```

例如, `reverseDisplay(12345)` 显示的是 54321。编写一个测试程序, 提示用户输入一个

整数，然后显示它的逆序数字。

- *18.9 (以逆序输出一个字符串中的字符) 编写一个递归方法，使用下面的方法头在控制台上以逆序显示一个字符串：

```
public static void reverseDisplay(String value)
```

例如，`reverseDisplay("abcd")` 显示的是 `dcba`。编写一个测试程序，提示用户输入一个字符串，然后显示它的逆序字符串。

- *18.10 (字符串中某个指定字符出现的次数) 编写一个递归方法，使用下面的方法头给出一个指定字符在字符串中出现的次数。

```
public static int count(String str, char a)
```

例如，`count("Welcome", 'e')` 会返回 2。编写一个测试程序，提示用户输入一个字符串和一个字符，显示该字符在字符串中出现的次数。

- *18.11 (使用递归求一个整数各位数之和) 编写一个递归方法，使用下面的方法头计算一个整数中各位数之和：

```
public static int sumDigits(long n)
```

例如，`sumDigits(234)` 返回的是 $2+3+4=9$ 。编写一个测试程序，提示用户输入一个整数，然后显示各位数字之和。

18.5 节

- **18.12 (以逆序打印字符串中的字符) 使用辅助方法改写编程练习题 18.9，将子串的 `high` 下标传递给这个方法。辅助方法头为：

```
public static void reverseDisplay(String value, int high)
```

- *18.13 (找出数组中的最大数) 编写一个递归方法，返回一个数组中的最大整数。编写一个测试程序，提示用户输入一个包含 8 个整数的列表，然后显示最大的元素。
- *18.14 (求字符串中大写字母的个数) 编写一个递归方法，返回一个字符串中大写字母的个数。编写一个测试程序，提示用户输入一个字符串，然后显示该字符串中大写字母的数目。
- *18.15 (字符串中某个指定字符出现的次数) 使用辅助方法改写编程练习题 18.10，将子串的 `high` 下标传递给这个方法。辅助方法头为：

```
public static int count(String str, char a, int high)
```

- *18.16 (求数组中大写字母的个数) 编写一个递归的方法，返回一个字符数组中大写字母的个数。需要定义下面两个方法。第二个方法是一个递归辅助方法。

```
public static int count(char[] chars)
```

```
public static int count(char[] chars, int high)
```

编写一个测试程序，提示用户在一行中输入一个字符列表，然后显示该列表中大写字母的个数。

- *18.17 (数组中某个指定字符出现的次数) 编写一个递归的方法，求出数组中一个指定字符出现的次数。需要定义下面两个方法，第二个方法是一个递归的辅助方法。


```
public static int count(char[] chars, char ch)
```

```
public static int count(char[] chars, char ch, int high)
```

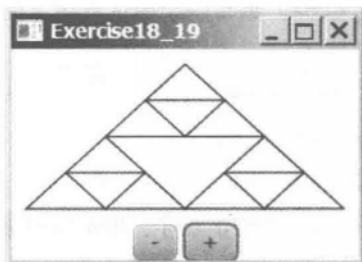
编写一个测试程序，提示用户在一行中输入一个字符列表以及一个字符，然后显示该字符在列表中出现的次数。

18.6 ~ 18.10 节

- *18.18 (汉诺塔) 修改程序清单 18-8，使程序可以计算将 n 个盘子从塔 A 移到塔 B 所需的移动次数。

 提示：使用一个静态变量，每当调用方法一次，该变量就加 1。

- *18.19 (思瑞平斯基三角形) 修改程序清单 18-9，开发一个程序，让用户使用“+”和“-”按钮将当前阶数增 1 或减 1，如图 18-12a 所示。初始阶数为 0。如果当前阶数为 0，就忽略“-”按钮。



a) 编程练习题 18.19 使用“+”和“-”按钮将当前阶数增加 1 或减小 1



b) 练习题 18.20 使用递归方法绘制多个圆

图 18-12

- *18.20 (显示多个圆) 编写一个 Java 程序显示多个圆，如图 18-12b 所示。这些圆都处于面板的中心位置。两个相邻圆之间相距 10 像素，面板和最大圆之间也相距 10 像素。

- *18.21 (将十进制数转换为二进制数) 编写一个递归方法，将一个十进制数转换为一个二进制数的字符串。方法头如下：

```
public static String dec2Bin(int value)
```

编写一个测试程序，提示用户输入一个十进制数，然后显示等价的二进制数。

- *18.22 (将十进制数转换为十六进制数) 编写一个递归方法，将一个十进制数转换为一个十六进制数的字符串。方法头如下：

```
public static String dec2Hex(int value)
```

编写一个测试程序，提示用户输入一个十进制数，然后显示等价的十六进制数。

- *18.23 (将二进制数转换为十进制数) 编写一个递归方法，将一个字符串形式的二进制数转换为一个十进制数。方法头如下：

```
public static int bin2Dec(String binaryString)
```

编写一个测试程序，提示用户输入一个二进制字符串，然后显示等价的十进制数。


- *18.24 (将十六进制数转换为十进制数) 编写一个递归方法，将一个字符串形式的十六进制数转换为一个十进制数。方法头如下：

```
public static int hex2Dec(String hexString)
```

编写一个测试程序，提示用户输入一个十六进制字符串，然后显示等价的十进制数。

- **18.25 (字符串排列) 编写一个递归方法，输出一个字符串的所有排列。例如，对于字符串 abc，输出为：

```
abc
acb
bac
bca
cab
cba
```

 提示：定义下面两个方法，第二个方法是一个辅助方法。

```
public static void displayPermutation(String s)
public static void displayPermutation(String s1, String s2)
```

第一个方法简单地调用 `displayPermutation("",s)`。第二个方法使用循环，将一个字符从 `s2` 移到 `s1`，并使用新的 `s1` 和 `s2` 递归地调用该方法。基础情况是 `s2` 为空，将 `s1` 打印到控制台。

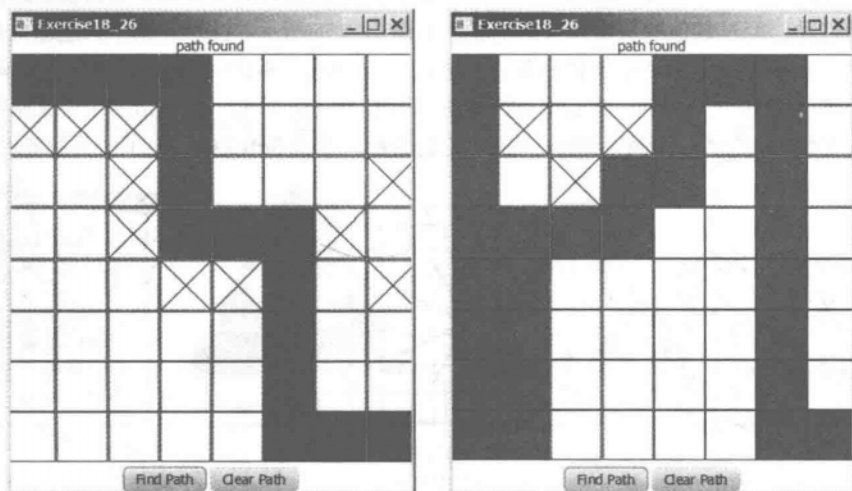
编写一个测试程序，提示用户输入一个字符串，然后显示其所有排列。

****18.26 (创建一个迷宫)** 编写一个程序，在迷宫中寻找一条路径，如图 18-13a 所示。该迷宫由一个 8×8 的棋盘表示。路径必须满足下列条件：

路径在迷宫的左上角单元和右下角单元之间。

程序允许用户在一个单元格中放入或移走一个标志。路径由相邻的未放标志的单元格组成。如果两个单元格在水平方向或垂直方向相邻，但在对角线方向上不相邻，那么就称它们是相邻的。

路径不包含能形成一个正方形的单元格。例如，在图 18-13b 中的路径就不满足这个条件。（这个条件使得面板上的路径很容易识别。）



a) 合法路径

b) 非法路径

图 18-13 程序求出从左上角到右下角的路径

****18.27 (科赫雪花分形)** 本章给出了思瑞平斯基三角形分形。本练习要编写一个程序，显示另一个称为科赫雪花 (Koch snowflake) 的分形，这是根据一位著名的瑞典数学家的名字命名的。科赫雪花按如下方式产生：

- 1) 从一个等边三角形开始，将其作为 0 阶 (或 0 级) 科赫分形，如图 18-14a 所示。
- 2) 三角形中的每条边分成三个相等的线段，以中间的线段作为底边向外画一个等边三角形，产生 1 阶科赫分形，如图 18-14b 所示。
- 3) 重复步骤 2 产生 2 阶科赫分形，3 阶科赫分形，...，如图 18-14c ~ d 所示。

***18.28 (非递归目录大小)** 不使用递归改写程序清单 18-7。

***18.29 (某个目录下的文件数目)** 编写一个程序，提示用户输入一个目录，然后显示该目录下的文件数。

****18.30 (找出单词)** 编写一个程序，递归地找出某个目录下的所有文件中某个单词出现的次数。从命令行如下传递参数：

```
java Exercise18_30 dirName word
```

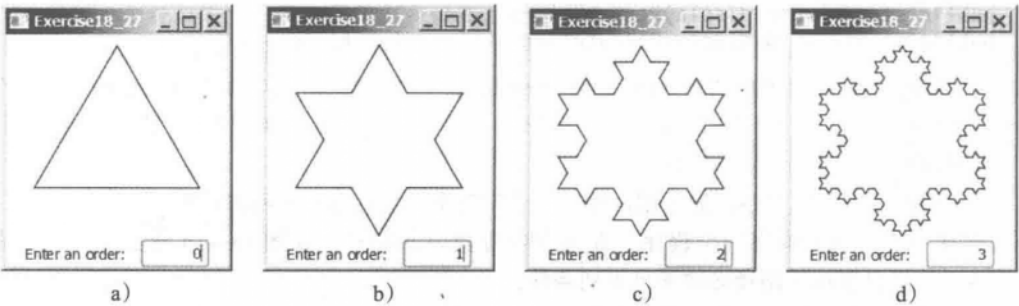
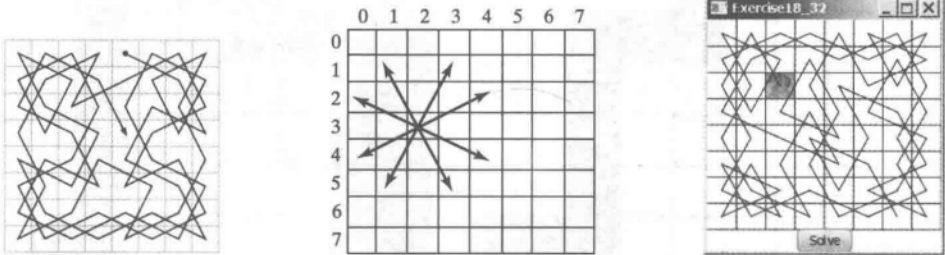


图 18-14 科赫雪花是一个从三角形开始的分形

****18.31 (替换单词)** 编写一个程序，递归地用一个新单词替换某个目录下的所有文件中出现的某个单词。从命令行如下传递参数：

```
java Exercisel8_31 dirName oldWord newWord
```

*****18.32 (游戏：骑士的旅途)** 骑士的旅途是一个古老的谜题。它的目的是使骑士从棋盘上的任意一个正方形开始移动，经过其他的每个正方形一次，如图 18-15a 所示。注意，骑士只能做 L 形的移动（两个空格在一个方向上而一个空格在垂直的方向上）。如图 18-15b 所示，骑士可以移动到八个正方形的位置。编写一个程序，显示骑士的移动，如图 18-15c 所示。当单击一个单元格的时候，骑士被放置在该单元格中。该单元格作为骑士的起始点。单击 Solve 按钮显示作为解答的路径。



a) 骑士遍历所有的正方形一次 b) 骑士作 L 形的移动 c) 程序显示骑士的旅途路径

图 18-15

提示：这个问题的穷举方法是将骑士从一个正方形随意地移动到另一个可用的正方形。使用这样的方法，程序将需要很多时间来完成。比较好的方法是采用一些启发式方法。依据骑士目前的位置，它可以有两个、三个、四个、六个或八个可能的移动线路。直觉上讲，应该首先尝试将骑士移动到可访问性最小的正方形，将那些更多的可访问的正方形保留为开放的，这样，在查找的结尾就会有更好的成功机会。

*****18.33 (游戏：骑士旅途的动画)** 为骑士旅途的问题编写一个程序，该程序应该允许用户将骑士放到任何一个起始正方形，并单击 Solve 按钮，用动画展示骑士沿着路径的移动，如图 18-16 所示。

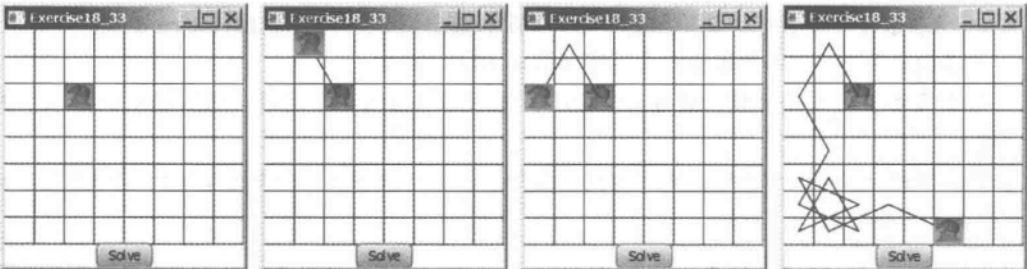


图 18-16 骑士沿着路径遍历

****18.34 (游戏：八皇后问题)** 八皇后问题是要找到一个解决方案，将一个皇后棋子放到棋盘上的每行中，并且两个皇后棋子之间不能相互攻击。编写一个程序，使用递归来解决八皇后问题，并如图 18-17 显示结果。



图 18-17 程序显示八皇后问题的求解

****18.35 (H- 树分形)** 一个 H- 树分形 (本章开始部分介绍过，如图 18-1) 如下定义：

1) 从字母 H 开始。H 的三条线长度一样，如图 18-1a 所示。

2) 字母 H (以它的 sans-serif 字体形式，H) 有四个端点。以这四个端点为中心位置绘制一个 1 阶 H 树，如图 18-1b 所示。这些 H 的大小是包括这四个端点的 H 的一半。

3) 重复步骤 2 来创建 2 阶，3 阶， \dots ， n 阶的 H 树，如图 18-1c-d 所示。

编写程序，绘制如图 18-1 所示的 H- 树。

18.36 (思瑞平斯基三角形) 编写一个程序，让用户输入一个阶数，然后显示填充的思瑞平斯基三角形，如图 18-18 所示。

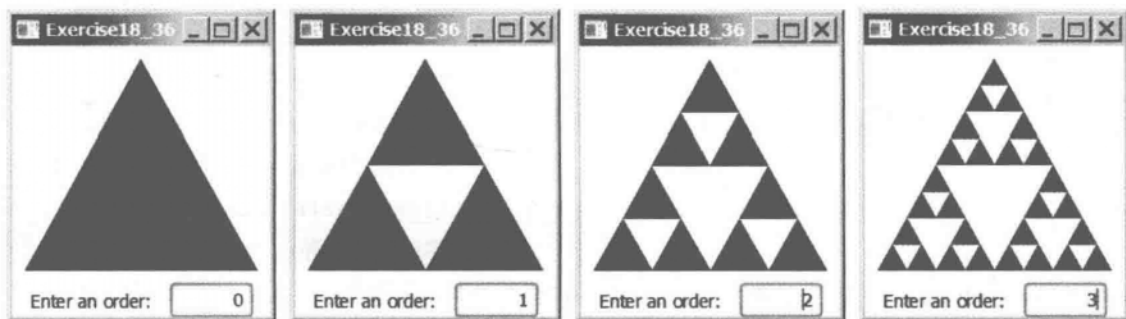


图 18-18 显示一个填充的思瑞平斯基三角形

****18.37 (希尔伯特曲线)** 希尔伯特曲线，由德国数学家希尔伯特于 1891 年第一个给出描述，是一种空间填充曲线，以 2×2 ， 4×4 ， 8×8 ， 16×16 ，或者任何其他 2 的幂的大小来访问一个方格网的每个点。编写程序，以给定的阶数显示希尔伯特曲线，如图 18-19 所示。

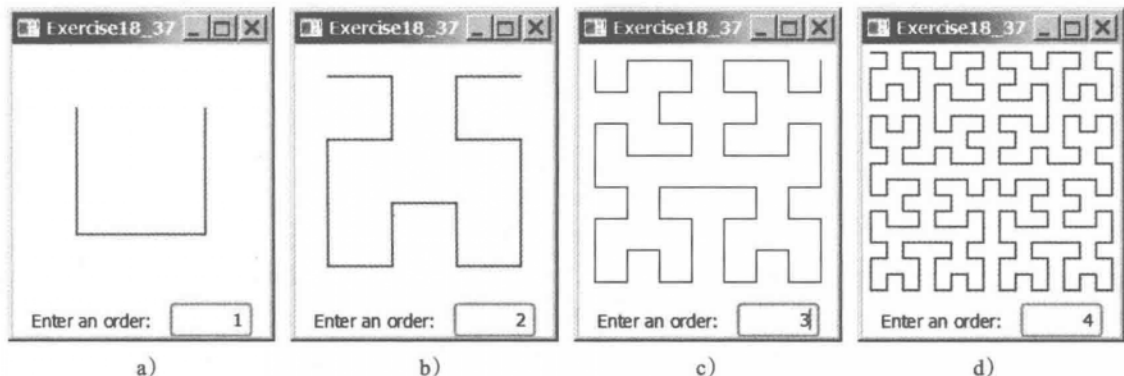


图 18-19 绘制给定阶数的希尔伯特曲线

****18.38 (递归树)** 编写一个程序来显示一个递归树，如图 18-20 所示。

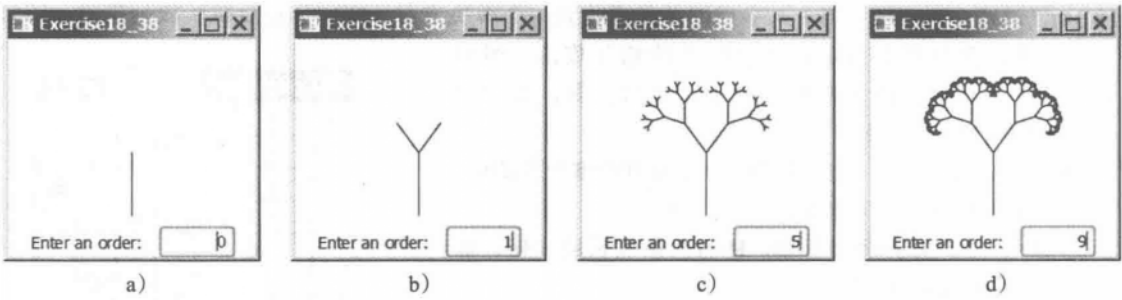


图 18-20 绘制一个带特定深度的递归树

****18.39** (拖动树) 修改编程练习题 18.38, 将树移动到鼠标所拖动到的位置。