第八讲

Recursion and Similarity

薛浩

2023 年 4 月 13 日

www.stickmind.com

今日话题

话题 1：编程基础 初学编程的新手，一般应该熟练使用函数和库处理字符串相关的编程任务。

话题 2：抽象数据类型的使用 在尝试实现抽象数据类型之前，应该先熟练使用这些工具解决问题。

话题 3：递归和算法分析 递归是一种强有力的思想，一旦掌握就可以解决很多看起来非常难的问题。

话题 4：类和内存管理 使用 C++ 实现数据抽象之前，应先学习 C++ 的内存机制。

话题 5：常见数据结构和算法 在熟练使用抽象数据类型解决常见问题之后，学习如何实现它们是一件很自然的事情。

话题 3：递归和算法分析

递归是一种强有力的思想，一旦掌握就可以解决很多看起来非常难的问题。

- 递归过程
- 算法分析
- 递归回溯
- 排序算法

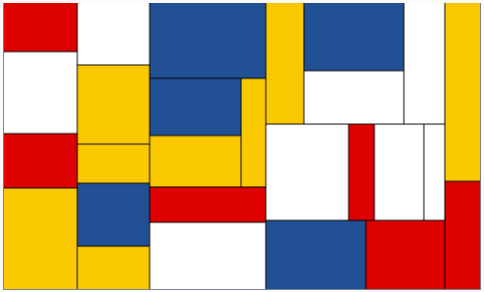


Figure 1: 递归艺术

如何使用递归思维解决问题?

目录

递归入门

递归

本课程最重要的“伟大思想”之一是**递归**（Recursion）的概念，递归是通过将问题划分为相同形式的较小子问题来解决问题的过程。

相同形式是递归的基本特征；如果形式不同，那么求解策略将转为之前介绍过的逐步求精。

递归分解生成的子问题与原始问题具有相同的形式，这意味着递归程序将使用**相同的函数或方法**来解决不同级别的子问题。

从代码的结构上看，递归程序的特征是在分解过程中直接或间接调用自身函数的程序。

复习：Factorial!

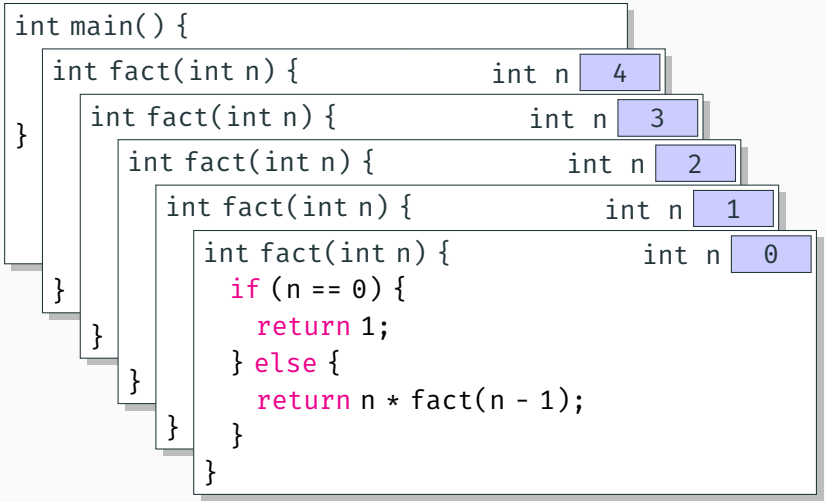
$$n! = \begin{cases} 1 & n = 0 \\ n \times (n-1)! & n > 0 \end{cases}$$

```
int fact(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}
```

复习：Factorial!

```
int main() {
    cout << "4! = " << fact(4) << endl;
    return 0;
}
```

复习：Factorial!



递归范式

大部分的递归函数遵循以下递归范式：

```
if (问题最简单的形式) {
    无需递归，直接处理并返回结果
} else {
    将问题分解为一个或多个形式相同的子问题
    调用同样的函数或方法解决每一个子问题
    整合所有子问题的处理结果，并返回最终结果
}
```

寻找递归解决方案主要是要弄清楚如何分解问题，使其符合递归范式。

递归范式

```
if (问题最简单的形式) {
    无需递归，直接处理并返回结果
} else {
    将问题分解为一个或多个形式相同的子问题
    调用同样的函数或方法解决每一个子问题
    整合所有子问题的处理结果，并返回最终结果
}
```

寻找递归解决方案主要是要弄清楚如何分解问题，使其符合递归范式。你必须做两件事：

- 识别无需递归处理的**最简单形式**（base case）
- 寻找一个**递归分解**（recursive decomposition）策略，可以将问题分解为相同形式、略微简单的子问题

练习：Palindrome

回文（palindrome）是一个正序和倒序都完全相同的字符串，例如“level”，“noon”等。

我们已经练习过回文的迭代处理方法，使用递归同样可以解决这个问题。例如，“level”可以直接判断首尾两个字符是否相等，然后再对子字符串“eve”进行递归处理。

为了检查一个字符串是否是回文，参照递归范式可以这么做：

基本形式 空字符串和单字符字符串

递归分解 检查首尾字符是否相等；接着判断去除首尾的子字符串是否是回文

练习：Palindrome

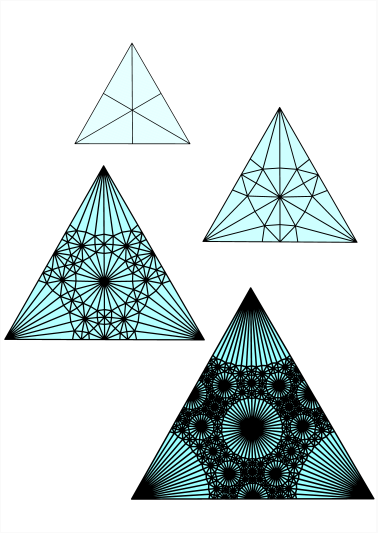
```
bool isPalindromeRec(string str) {
    int n = str.length();
    if (n < 2) {
        return true;
    } else {
        if (str[0] != str[n - 1]) {
            return false;
        } else {
            string substr = str.substr(1, n - 2);
            return isPalindromeRec(substr);
        }
    }
}
```

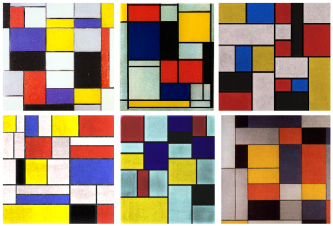
递归分形

自相似

自相似（Self-similarity）是指一个物体与自身的一部分完全或近似地相似（即，整体与一个或多个部分具有相同的形状）。

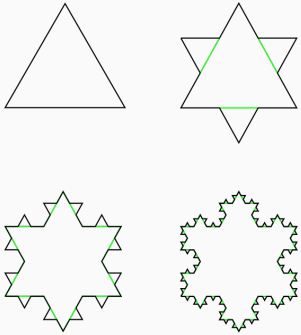
现实世界中的许多物体，如海岸线，在统计上是自相似的：它们的某些部分在许多尺度上显示出相同的统计属性。





分形

分形（Fractals）是由相同形状或模式的重复实例组成，以结构化的方式排列。自相似性是分形的典型性质，科赫曲线在放大时具有无限重复的自相似性。



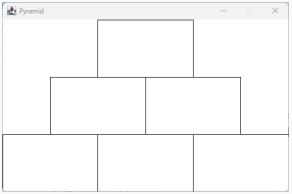
分形如何终止

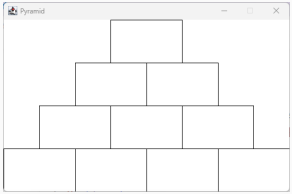
递归一般是把规模较为复杂的问题，简化为规模较为简单的问题来处理。但是，递归分形常常根据一些规则，不断由简单状态变向复杂状态的一个过程。

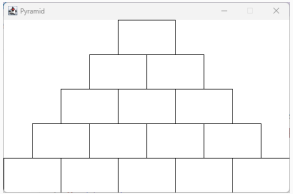
> *一尺之棰，日取其半，万世不竭。*
>
> *——庄子*

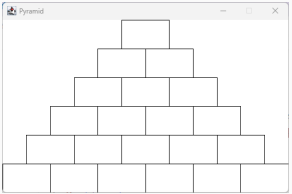
为了使递归过程能够终止，对于分形和自相似的递归问题，通常使用**阶**（Order）来定义其复杂性，人为打断其演变过程。

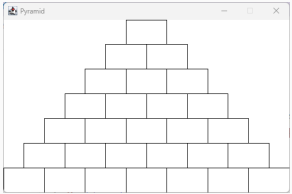
练习：Pyramid











包装器函数

包装器函数 Wrapper Function

在优化函数性能时，有时需要增加参数提供额外的信息。但是，修改函数声明破坏了接口设计的稳定性原则。

为了避免这个问题，可以将优化后的函数，封装到原接口函数中。通过原接口函数间接调用优化后的新函数。

这样，原接口函数就扮演了一个**包装器函数**（Wrapper Function）的作用。包装器函数在递归和接口设计中非常普遍。

练习：Palindrome

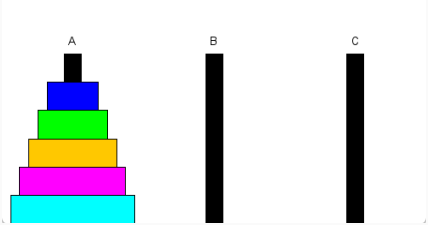
```
bool isPalindromeBetter(string str, int p1, int p2) {
  if(p1 >= p2) {
    return true;
  } else {
    if(str[p1] != str[p2])
      return false;
    else
      return isPalindromeBetter(str, p1 + 1, p2 - 1);
  }
}

bool isPalindromeRec(string str) {
    return isPalindromeBetter(str, 0, str.length() - 1);
}
```

递归信任

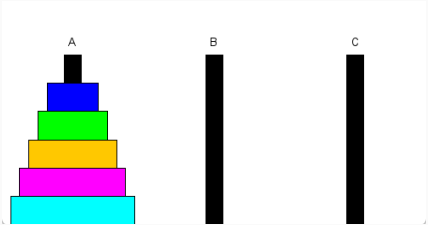
练习：汉诺塔 Hanoi

汉诺塔（Towers of Hanoi）是由法国数学家爱德华・卢卡斯在 1880 年提出。这里简化为将 A 柱上的 5 个圆盘移动到 B 柱上，同时要保证：

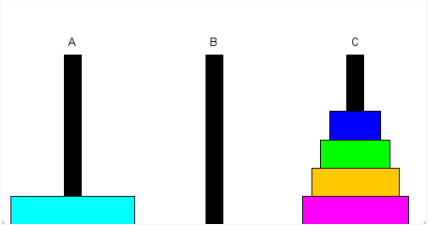
- 每次只能移动一个圆盘
- 始终要保证较大的圆盘在下面
- 可以利用 C 柱作临时中转



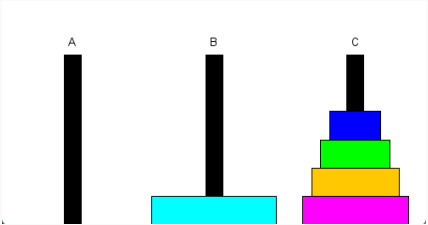
练习：汉诺塔 Hanoi



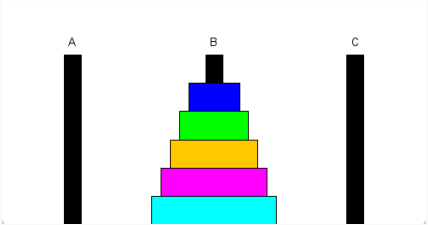
练习：汉诺塔 Hanoi



练习：汉诺塔 Hanoi



练习：汉诺塔 Hanoi



递归信任 Recursive Leap of Faith

解决 Towers of Hanoi 问题必须要保持整体观，你要相信这个过程是有效的。递归调用的内部操作与其他方法调用没有本质不同。

不要尝试追踪递归过程的细节！只有当你对这个过程有足够的信心，编写递归程序才会变得很自然。

在编写递归程序时，重要的是要相信！只要参数定义了一个更简单的子问题，任何递归调用都会返回正确的答案。

这种心理上的策略可以称为**递归信任**（Recursive Leap of Faith）。

如何使用递归思维解决问题?

问题?