01.04 回文判断

题目描述

回文，英文palindrome，指一个顺着读和反过来读都一样的字符串，比如madam、我爱我，这样的短句在智力性、趣味性和艺术性上都颇有特色，中国历史上还有很多有趣的回文诗。

那么，我们的第一个问题就是：判断一个字串是否是回文？

分析与解法

回文判断是一类典型的问题，尤其是与字符串结合后呈现出多姿多彩，在实际中使用也比较广泛，而且也是面试题中的常客，所以本节就结合几个典型的例子来体味下回文之趣。

解法一

同时从字符串头尾开始向中间扫描字串，如果所有字符都一样，那么这个字串就是一个回文。采用这种方法的话，我们只需要维护头部和尾部两个扫描指针即可。

代码如下：：

```
bool IsPalindrome(const char *s, int n)
{
        // 非法输入
        if (s == NULL || n < 1)
        {
                return false;
        }
        const char* front,*back;

        // 初始化头指针和尾指针
        front = s;
        back = s+ n - 1;

        while (front < back)
        {
                if (*front != *back)
                {
                        return false;
                }
                ++front;
                --back;
        }
        return true;
}
```

这是一个直白且效率不错的实现，时间复杂度：O(n)，空间复杂度：O(1)。

解法二

上述解法一从两头向中间扫描，那么是否还有其它办法呢？我们可以先从中间开始、然后向两边扩展查看字符是否相等。参考代码如下：

```
bool IsPalindrome2(const char *s, int n)
{
        if (s == NULL || n < 1)
        {
                return false;
        }
        const char* first, *second;

        // m定位到字符串的中间位置
        int m = ((n >> 1) - 1) >= 0 ? (n >> 1) - 1 : 0;
        first = s + m;
        second = s + n - 1 - m;

        while (first >= s)
        {
                if (*first!= *second)
                {
                        return false;
                }
                --first;
                ++second;
        }
        return true;
}
```

时间复杂度：O(n)，空间复杂度：O(1)。

虽然本解法二的时空复杂度和解法一是一样的，但很快我们会看到，在某些回文问题里面，这个方法有着自己的独到之处，可以方便的解决一类问题。

举一反三

1、判断一条单向链表是不是“回文”

分析：对于单链表结构，可以用两个指针从两端或者中间遍历并判断对应字符是否相等。但这里的关键就是如何朝两个方向遍历。由于单链表是单向的，所以要向两个方向遍历的话，可以采取经典的快慢指针的方法，即先位到链表的中间位置，再将链表的后半逆置，最后用两个指针同时从链表头部和中间开始同时遍历并比较即可。

2、判断一个栈是不是“回文”

分析：对于栈的话，只需要将字符串全部压入栈，然后依次将各字符出栈，这样得到的就是原字符串的逆置串，分别和原字符串各个字符比较，就可以判断了。