栈与队列的应用——括号配对检验实验报告

崔士强 PB22151743

1 问题描述

假设一个表达式有英文字母（大、小写）、数字、四则运算符 (+ − ∗ /) 和左右小括号、中括号、大括号构成，以“@”作为表达式的结束符。本程序检查表达式中的左右大中小括号是否匹配，若匹配，则返回”YES”; 否则返回”NO”。

输入文件中第一行是表达式数目 N，之后是需要进行括号配对检测的 N 个表达式。N 行输出分别对应输入的 N 行表达式，每行都为”YES” 或”NO”

2 算法描述

2.1 数据结构描述

本程序利用栈存储符号，读取到左括号则进栈，读取到右括号则出栈，相关类型定义如下：

```
typedef char SElemType;

typedef struct {
    char *base;
    char *top;
    int stacksize;
} SqStack;
```

2.2 程序结构描述

栈的相关函数声明如下：

```
// 栈操作函数声明
Status InitStack(SqStack &S);
    // 构造一个空栈S
Status DestroyStack(SqStack &S);
    // 销毁栈S，S不再存在
Status ClearStack(SqStack &S);
    // 把S置为空栈
Status StackEmpty(SqStack S);
    // 若栈S为空栈，则返回TRUE，否则返回FALSE
int StackLength(SqStack S);
```

```
11      // 返回S的元素个数，即栈的长度
12  Status GetTop(SqStack S, SElemType &e);
13      // 若栈不空，则用e返回S的栈顶元素，并返回OK；否则返回ERROR
14  Status Push(SqStack &S, SElemType e);
15      // 插入元素e为新的栈顶元素
16  Status Pop(SqStack &S, SElemType &e);
17      // 若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回ERROR
18  Status StackTraverse(SqStack S, Status (*visit)(SElemType &e));
19      // 从栈底到栈顶依次对栈中每个元素调用函数visit()。一旦visit()失败，则操作失败
```

3 调试分析

3.1 测试数据

选取包含各种错误（如配对不正确，优先级不正确）的表达式进行测试

3.2 问题及解决方法

调试中发现程序并不能正确地对括号优先级进行检验，每次对左括号进栈时检查栈顶元素可以解决这个问题。

4 算法的时空分析

从以下几个方面分析：

1. 文件读取和输出：文件的读取和输出操作在这里是线性的，因为它们依次处理文件中的每一行。但这通常不是主要的时间复杂度来源，除非文件非常大。
2. 外部循环：外部 for 循环迭代的次数等于文件中的行数 LineNum。
3. 内部循环：内部 for 循环遍历每一行中的字符直到遇到'@' 字符。假设每行的平均字符数为 M，则内部循环的复杂度是 $O(M)$。
4. 栈操作：栈操作（如 Push, Pop, GetTop, StackEmpty）是常数时间操作，即 $O(1)$。

因此，总的时间复杂度是外部循环和内部循环的乘积，即 $O(LineNum \times M)$。这意味着算法的执行时间随着输入文件的行数和每行的平均长度的增加而线性增长。

5 测试结果及分析

测试用数据如下所示：

```
1  6
2  3*(4+5)-{6/[7*(8-9)]}@
3  {[1+2*(3-4)]/5}+6-7@
4  1+[2*(3-4]}/5@
5  6*{7+[8-(9*10]}@
```

```
8/{9+(10-[11*12])}@
{[(2+3*4]/(5-6)+7}@
```

输出结果:

```
Yes
Yes
No
No
No
No
```

全部正确。

6 实验体会和收获

掌握了栈的相关结构和算法，对栈的应用场景以及使用栈的原因有了更深刻的理解