

栈实验报告

课程名称：数据结构与算法

学 院： 计算机与信息安全学院

专业班级：19003603

学 号：1900301517

姓 名：陆洪业

报告日期：2021年 01 月 10 日

目录

**[1. 栈与递归函数](#_Toc81698963_WPSOffice_Level1)** **[1](#_Toc81698963_WPSOffice_Level1)**

[1.1. 实验目的](#_Toc1175904417_WPSOffice_Level2) [1](#_Toc1175904417_WPSOffice_Level2)

[1.2. 程序设计概要](#_Toc80728084_WPSOffice_Level2) [1](#_Toc80728084_WPSOffice_Level2)

[1.3. 系统测试与结果](#_Toc793249130_WPSOffice_Level2) [5](#_Toc793249130_WPSOffice_Level2)

[1.4. 实验小结](#_Toc88645464_WPSOffice_Level2) [6](#_Toc88645464_WPSOffice_Level2)

**[2. 中缀表达式转换为后缀表达式](#_Toc1175904417_WPSOffice_Level1)** **[6](#_Toc1175904417_WPSOffice_Level1)**

[2.1. 实验目的](#_Toc723446325_WPSOffice_Level2) [6](#_Toc723446325_WPSOffice_Level2)

[2.2. 程序设计概要](#_Toc284905462_WPSOffice_Level2) [6](#_Toc284905462_WPSOffice_Level2)

[2.3. 系统测试与结果](#_Toc459475836_WPSOffice_Level2) [10](#_Toc459475836_WPSOffice_Level2)

[2.4. 实验小结](#_Toc390973318_WPSOffice_Level2) [11](#_Toc390973318_WPSOffice_Level2)

**[3. 后缀表达式求值](#_Toc80728084_WPSOffice_Level1)** **[11](#_Toc80728084_WPSOffice_Level1)**

[3.1. 实验目的](#_Toc1976733394_WPSOffice_Level2) [11](#_Toc1976733394_WPSOffice_Level2)

[3.2. 程序设计概要](#_Toc705108956_WPSOffice_Level2) [11](#_Toc705108956_WPSOffice_Level2)

[3.3. 系统测试与结果](#_Toc353330969_WPSOffice_Level2) [14](#_Toc353330969_WPSOffice_Level2)

[3.4. 实验小结](#_Toc1096721698_WPSOffice_Level2) [14](#_Toc1096721698_WPSOffice_Level2)

# 栈与递归函数

## 实验目的

使用栈将递归函数转化为非递归函数。解决火车上下车问题。

## 程序设计概要

### 设计目标

设计一个栈， 用于计算和储存斐波那契数列。然后通过斐波那契解决火车问题。

需要实现入栈函数，出栈函数、取栈顶函数和应用栈解决火车问题函数。

### 选用储存结构

双向循环链表

### 有关类型和函数的定义

栈的双链表结构体：

struct stack

{

int data;

struct stack \* next;

struct stack \* pre;

};

/\*在此定义并完成第一关函数，参考main要求\*/

typedef struct stack \*Stack;

void push(Stack s, int x)

功能: 入栈

参数: 1栈链表头指针, 2 入栈数据

int pop(Stack s)

功能: 栈顶出栈

参数: 栈链表头指针

int top(Stack s)

功能: 取栈顶值

参数: 栈链表头指针

int result(int n, int m, int a, int x)

功能: 利用斐波那契数列和求解火车上下车问题;

参数: 1 车站总数; 2 最后一站的下车人数;

3 始发站上车人数; 4要求下车人数的站

返回值: 第x站下车人数

### 数据结构的设计以及函数的设计

void push(Stack s, int x)

原理: 用头插法将节点插入到栈里面.

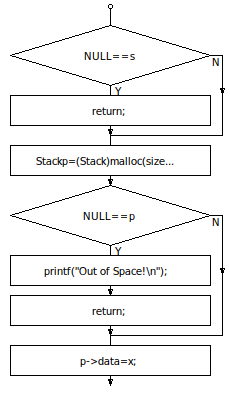
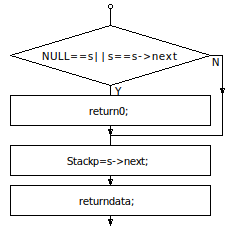


表 1pop

int pop(Stack s)

原理: 链表节点删除



int top(Stack s)

原理: 读取链表节点数值

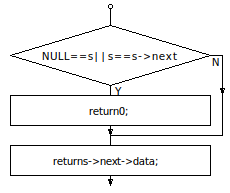


表 2top

int result(Stack s)

原理: (1)设第二站下车的人数为a+a2,第n-1站下车的人数为c\_a \* a + c\_a2 \* a2 , c\_a 比c\_a2领先1个斐波那契数(2)上下车人数系数为斐波那契和的形式(3)算非波那契数列的时候没能算出具体数值的时候先将这个结构体压入栈, 压到第一第二项计算出具体值之后再逐个出栈.

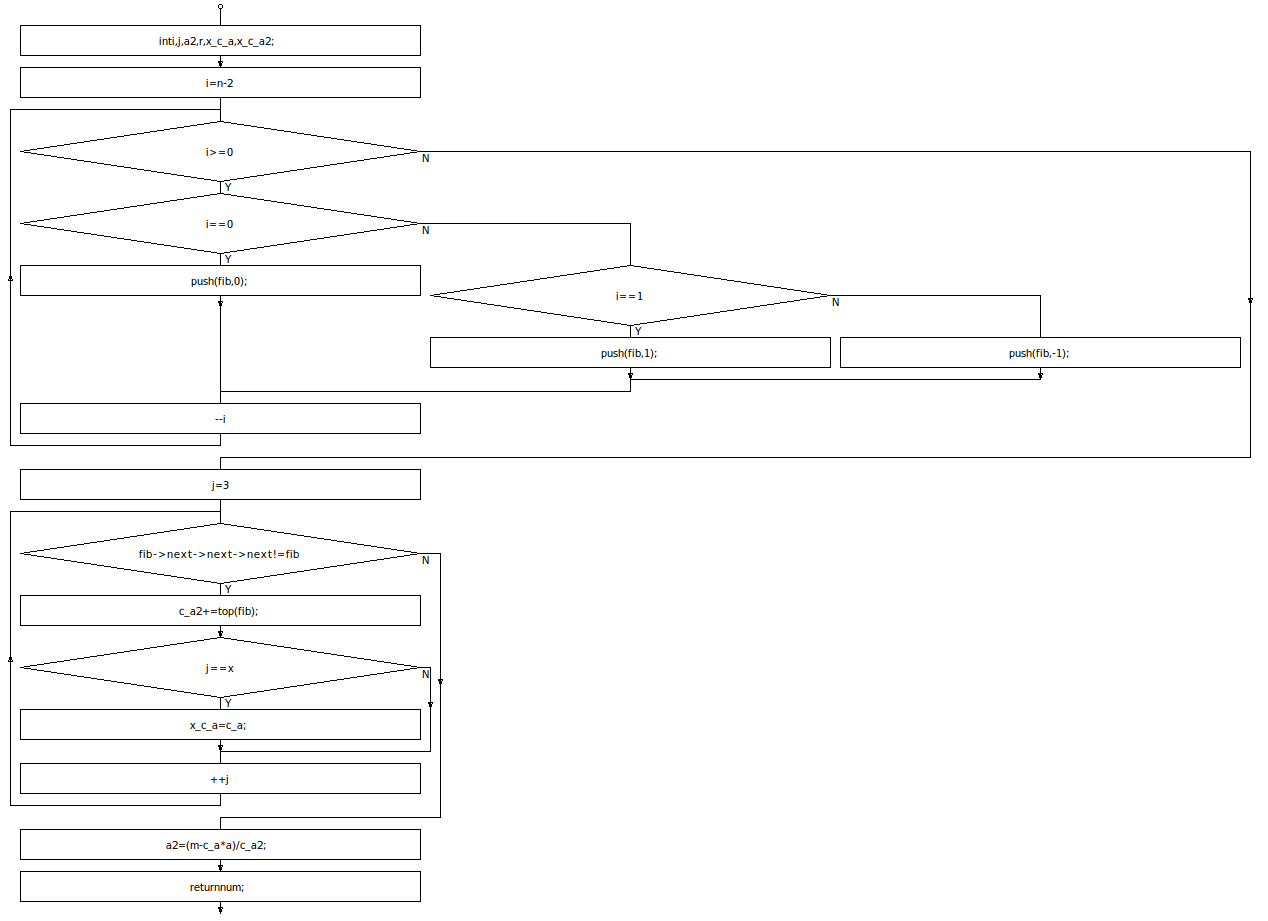


表 3result

### 复杂度的分析

push,pop链表节点的插入删除, 复杂度O(n)

top取头节点的值,复杂度O(n)

result分为入栈和出栈两个部分, 入栈为一个循环,复杂度为O(n),出栈又一个循环,复杂度为O(n),根据加法规则, 总复杂度为O(n)

|  |  |  |  |
| --- | --- | --- | --- |
| push | pop | top | result |
| O(1) | O(1) | O(1) | O(n) |

## 系统测试与结果





上述测试及结果证明了系统测试通过,各项操作函数能正常执行,并达到了实验要求的目标,实现了火车站上下车人数的求解, 实现了斐波那契数列递归转栈.

## 实验小结

通过本次实验, 我对栈的有了进一步的理解, 能够更灵活地将栈运用到实践中, 并对递归用栈实现有了更进一步的了解.

# 中缀表达式转换为后缀表达式

## 实验目的

编写一个能将中缀表达式转换为后缀表达式的程序。

## 程序设计概要

### 设计目标

设计设计两个双循环链表, 一个用于储存转换结果, 一个是运算符栈. 要设计一个栈的逆向入栈函数, 一个栈的逆向出栈函数和一个中缀转后缀函数.

### 选用储存结构

双向循环链表

### 有关类型和函数的定义

struct stack

{

int data;

struct stack \* next;

struct stack \* pre;

};

/\*在此定义并完成第一关函数，参考main要求\*/

typedef struct stack \*Stack;

void push\_rev(Stack s, int x)

反向入栈函数

int pop\_rev(Stack s)

反向出栈函数

Stack inToPost(char \*expression)

函数名：inToPost，本函数名和参数不能更改

函数功能：将中缀表达式转换为后缀表达式输出

函数参数：中缀表达式，放在字符数组中

返回值：返回放有后缀表达式的栈

void print(Stack s)  
print函数用于输出后缀表达式，参数是 inToPost的返回值

### 数据结构的设计以及函数的设计

void push\_rev(Stack s, int x)

原理: 利用双链表的特性将节点插入到链表头指针的s->pre上

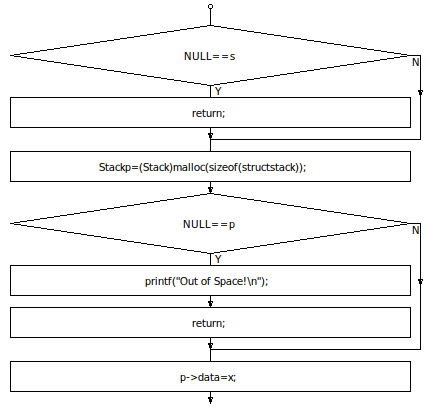


表 4push\_rev

int pop\_rev(Stack s)

原理： 将链表的next换成了pre出栈int pop\_rev(Stack s)

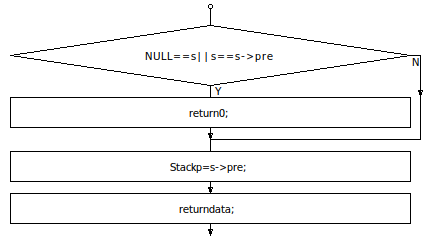


表 5pop

Stack inToPost(char \*expression)

原理： (1)顺序扫描中缀表达式(2)读入一个运算分量立即输出(3)在读入一个操作负时,首先将它放进一个运算符栈,直到两个匀速啊安抚分量都读到后才能将它输出到储存链表中 (4)优先级高的先入运算符栈(5)输出储存栈使用反向输出,保证输出的时候是正向的.

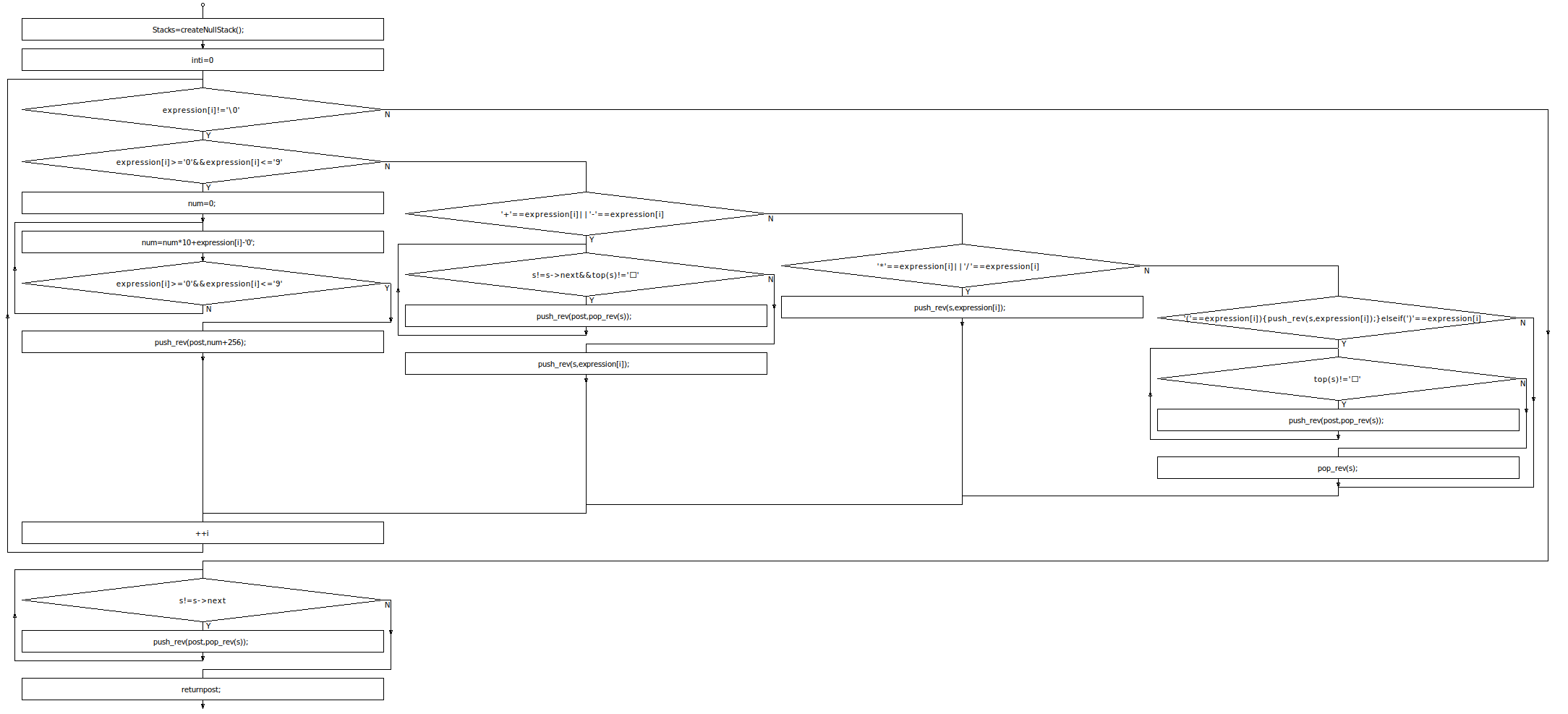


表 6inToPost

void print(Stack s)

原理: 正向遍历inToPost反向保存的链表栈,输出

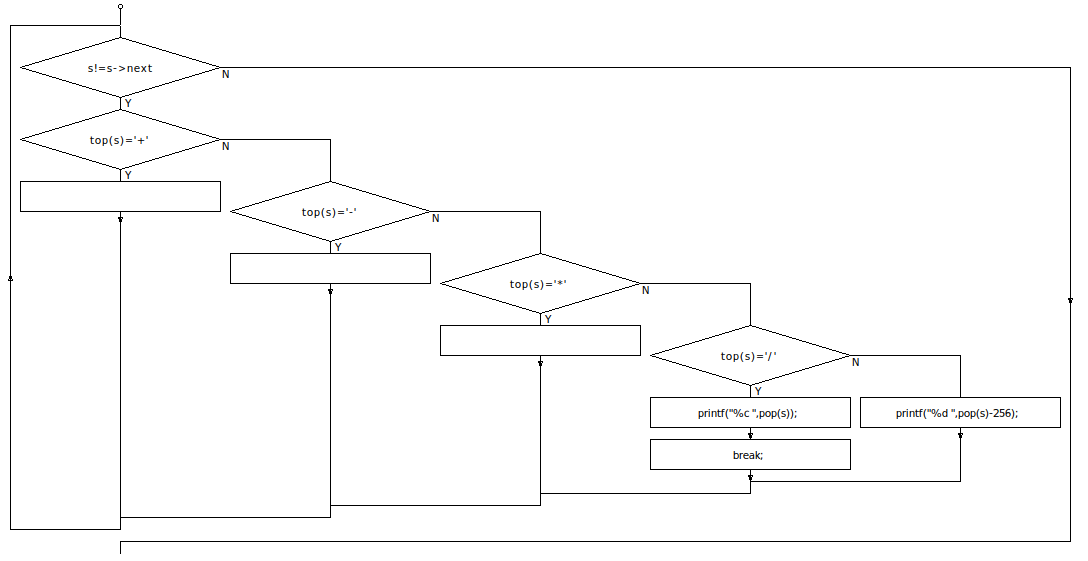


表 7print

### 复杂度的分析

正向出入栈和反向出入栈操作差不多, 复杂度均为O(1)

inToPost 的处理是在遍历链表的过程中进行的, 每次循环复杂度都是O(1),所以总复杂度为O(n)

print正向遍历链表, 复杂度为O(n)

|  |  |  |  |
| --- | --- | --- | --- |
| Push\_rev | Pop\_rev | inToPost | Print |
| O(1) | O(1) | O(n) | O(n) |

## 系统测试与结果



上述测试及结果证明了系统测试通过,各项操作函数能正常执行,并达到了实验要求的目标,实现了中缀表达式转换为后缀表达式,而且成功地输出表达式.

## 实验小结

通过这次实验, 我对栈又有了更深刻的认识, 认识到它还可以用双链表实现, 栈只是一种思想, 不能被栈先进后出的思想给束缚住, 认为插入一定要在某个固定的栈顶插入, 有需求的时候也可以进行相应的变通, 比如用双链表可以实现栈的头尾倒置.

# 后缀表达式求值

## 实验目的

在第二关的基础上，实现后缀表达式求值。

## 程序设计概要

### 设计目标

创建一个存放运算分量的栈, 用于计算过程中中间运算结果的处理. 设计一个用栈对后缀表达式裘值的函数.

### 选用储存结构

双向循环链表

### 有关类型和函数的定义

struct node

{//链表结点类型，包含一个存放整型数据的 data 成员，和一个指向下一个结点的next成员

int data ;

struct node \*next ;

};

int calExp(char \*express)

函数名：calExp，本函数名和参数不能更改

函数功能：调用inToPost函数求解后缀表达式，并求解后缀表达式的值返回

函数参数：

返回值：无

### 数据结构的设计以及函数的设计

int calExp(char \*express)

原理: (1)扫描后缀表达式(2)每次遇到运算分量便将它推入栈中;(3)遇到运算符时, 就从运算分量弹出两个数(运算分量)进行计算 , 并且把运算结果推入栈中;

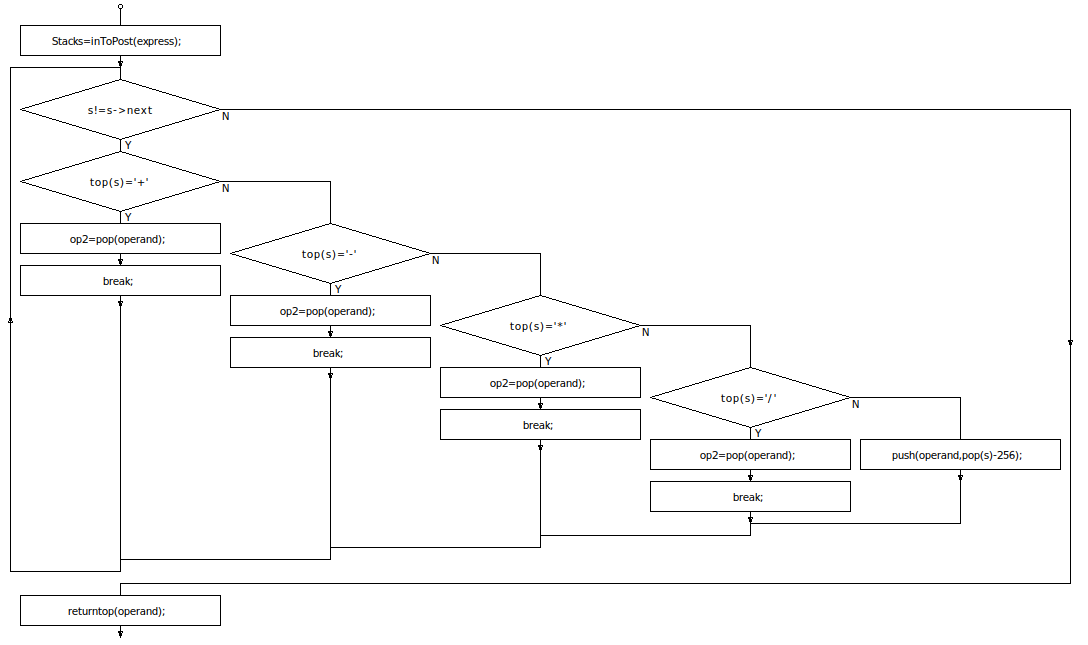


表 8calExp

### 复杂度的分析

calExp扫描后缀表达式,复杂度为O(n)

|  |
| --- |
| calExp |
| O(n) |

## 系统测试与结果

## 

上述测试及结果证明了系统测试通过,各项操作函数能正常执行,并达到了实验要求的目标,实现了后缀表达式的求值

## 实验小结

通过本次实验, 我对栈有了更深入的理解.