**《数据结构》实验报告**

**姓名： 杨孟衡 学号： 8002118240**

**班级： 软件工程1809班 专业： 软件工程**

**报告日期： 2019 年 9 月 28 日**

**实验二 栈的的实际应用**

**一、问题描述与分析**

1. 设某线性数据元素类型为字符型，以线性字符栈为存储结构。试编程实现：

1. 编写程序，键盘输入中序表达式
2. 将中序表达式转换成后序表达式
3. 按照后序表达式计算结果
4. 无论输入还是输出，要给出适当的提示信息
5. 按照后序表达式计算结果
6. 用栈实现

2.分析

先创建两个线性空栈，提示输入数据元素，然后对实验数据进行转换计算处理，最后提示输出数据元素。

**二、数据结构与算法设计**

1.数据结构：

静态线性字符栈采用基本数据类型字符数组作为基本元素构成，搭配一个整型变量top作为栈顶下标。代码如下：

#define MAXSIZE 200

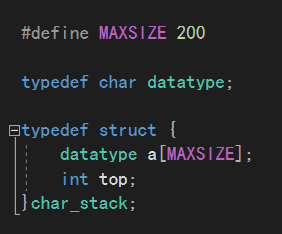
typedef char datatype;

typedef struct {

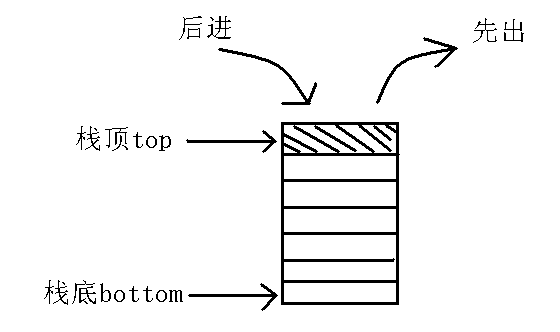
datatype a[MAXSIZE];

int top;

}char\_stack;



栈图示如下：



2.算法

⑴ 读入数据

Do-while循环，直到读入#才结束中缀表达式的读入；

⑵ 转换为后缀表达式

设置一个while循环，直到上一步所读入的所有字符处理完结束循环，实现实现中缀表达式转换为后缀表达式；

⑶ 计算后缀表达式

设置一个while循环，不断取后缀表达式的字符进行处理，得到一个一个的值以及操作符，再根据操作符进行相应的计算，最终将结果存储在临时结果数组的头位置(result[0])；

⑸ 提示输出信息，输出计算后的后缀表达式最终结果

Pirntf即可；

**三、算法复杂度分析**

⑴ 循坏读入中缀表达式

采用for循环方式读入字符串，算法复杂度为O(n);

⑵ 循环处理操作数，操作符

采用while循环的方式不断对数据项加以处理，算法复杂度为O(n)。

⑶ 循环输出后缀表达式

采用for循环方式输出后缀表达式，算法复杂度为O(n)。

**四、测试计划**

1.编写目的

加深对数据结构栈的顺序表存储结构的理解。

2.开发及运行环境

Visual Studio 2017

3.代码流程

（1）中缀表达式初始化：

提示用户输入中缀表达式

（2）转换

利用算法将中缀表达式转换为后缀表达式

（3）计算后缀表达式的值

调用相关函数处理字符数据，计算结果

4.测试截图

（1）输入中缀表达式



（2）转换



（3）计算



**五、源程序**

**<char\_stack.h>:**

#pragma once

#include<stdio.h>

#include<stdlib.h>

#define MAXSIZE 200

typedef char datatype;

typedef struct {

datatype a[MAXSIZE];

int top;

}char\_stack;

void Initiate(char\_stack\* c);

bool Empty(char\_stack c);

char returnTop(char\_stack c);

void Push(char\_stack\* c, datatype data);

void Pop(char\_stack\* c);

bool IsPlalindrome();

bool IsOp(char op);

int Priority(char op);

double readNumber(char in[], int\* i);

void CalPostFix(char e[], char f[]);

//初始化字符栈函数，top表示下一个将要插入的节点的存储位置

void Initiate(char\_stack\* c)

{

c->top = 0;

}

//根据top值判断是否为空，若top为0则为空，若不为0则不是空栈

bool Empty(char\_stack c)

{

return (c.top ? 0 : 1);

}

//取得当前栈顶值，若不为空则返回栈顶值，若为空输出错误信息，返回NULL值代表出错

datatype returnTop(char\_stack c)

{

if (Empty(c))

{

printf("\n这是一个空栈！读取失败！\n");

return NULL;

}

return c.a[c.top - 1];

}

//进栈操作，若栈还未满则根据top值直接插入，若已满输出错误提示信息

void Push(char\_stack\* c, datatype data)

{

if (c->top == MAXSIZE)

{

printf("\n字符栈已满！插入失败！\n");

exit(-1);

}

c->a[c->top] = data;

c->top++;

}

//出栈操作，若栈不为空则top值减一即可，若为空栈输出错误提示信息

void Pop(char\_stack\* c)

{

if (c->top == 0)

{

printf("\n字符栈还是空的！弹出元素失败！\n");

exit(-2);

}

c->top--;

}

/\*判断回文函数，采用两个栈分别正序和反序插入字符串数据,首先提示输入字符串数据，

输入以后，用一个临时申请内存的字符数组存放字符串。接着按照正序先给栈1(temp[0])插入字符串数据，

再按照倒序给栈2（temp[1]）插入字符串数据。算法核心是通过循环判断两个栈是否为空，每次循环首先

判断栈顶值是否相等（字符相同值相等），若不相等则说明不是回文，返回false，若相等则弹出栈顶元素，

继续下一轮栈顶值判断，直到两个栈弹出所有元素为空栈，那么就说明这个字符串正序读和倒序读都一样，

这是一个回文字符串，返回true

\*/

bool IsPlalindrome()

{

char\_stack temp[2];

char\* incs = (char\*)malloc(sizeof(temp[0].a));

int j = 0;

Initiate(&temp[0]);

Initiate(&temp[1]);

if (!incs)

{

printf("\n存储空间不足！无法读入字符串！\n");

exit(-3);

}

printf("输入字符串：\n");

scanf("%s", incs);

for (int i = 0; incs[i] != '\0'; i++, j++)

{

Push(&temp[0], incs[i]);

}

for (j -= 1; j >= 0; j--)

{

Push(&temp[1], incs[j]);

}

while (!Empty(temp[0]) && !Empty(temp[1]))

{

if (returnTop(temp[0]) == returnTop(temp[1]))

{

Pop(&temp[0]);

Pop(&temp[1]);

}

else return false;

}

return true;

}

//判断当前字符是否为操作符，若是返回true，若不是返回false

bool IsOp(char op)

{

switch (op)

{

case '+':

case '-':

case '\*':

case '/':return true;

default:return false;

}

}

//优先级函数，判断当前入参op的操作符优先级

int Priority(char op)

{

switch (op)

{

case '#':return -1;

case '(':return 0;

case '+':

case '-':return 1;

case '\*':

case '/':return 2;

default:return -2;

}

}

//数值转换函数，对于一个字符数组中的数字字符，一个一个读入

double readNumber(char in[], int\* i)

{

double x = 0.0;

int k = 0;

while (in[\*i] >= '0' && in[\*i] <= '9')

{

x = x \* 10 + (in[\*i] - '0');

(\*i)++;

}

if (in[\*i] == '.')

{

(\*i)++;

while (in[\*i] >= '0' && in[\*i] <= '9')

{

x = x \* 10 + (in[\*i] - '0');

(\*i)++;

k++;

}

}

while (k != 0)

{

x /= 10.0;

k--;

}

return x;

}

//

void CalPostFix(char\_stack\* c)

{

int i = 0;

int j = 0;

int t;

char temp[MAXSIZE];

char after[MAXSIZE];

double x1, x2;

double result[MAXSIZE];

int top = 0;

//输入中缀表达式

Initiate(c);

Push(c, '#');

printf("请输入中缀表达式(以#结束):\n");

do {

scanf("%c", &temp[i]);

i++;

} while (temp[i - 1] != '#');

i = 0;

//转换中缀表达式为后缀表达式

while (temp[i] != '#')

{

if ((temp[i] >= '0' && temp[i] <= '9') || temp[i] == '.')

{

after[j++] = temp[i];

}

else if (temp[i] == '(')

{

Push(c, temp[i]);

}

else if (temp[i] == ')')

{

t = c->top - 1;

while (c->a[t] != '(')

{

after[j++] = c->a[--c->top];

t = c->top - 1;

}

Pop(c);

}

else if (IsOp(temp[i]))

{

after[j++] = ' ';

while (Priority(c->a[c->top - 1]) >= Priority(temp[i]))

{

after[j++] = c->a[--c->top];

}

Push(c, temp[i]);

}

i++;

}

while (c->top) after[j++] = c->a[--c->top];

printf("经过转换后的后缀表达式为:");

for (int k = 0; after[k] != '#'; k++)

{

printf("%c", after[k]);

}

//计算后缀表达式

i = 0;

Initiate(c);

while (after[i] != '#')

{

if (after[i] >= '0' && after[i] <= '9')

{

result[top] = readNumber(after, &i);

top++;

}

else if (after[i] == ' ') i++;

else if (after[i] == '+')

{

x2 = result[--top];

x1 = result[--top];

result[top] = x1 + x2;

top++;

i++;

}

else if (after[i] == '-')

{

x2 = result[--top];

x1 = result[--top];

result[top] = x1 - x2;

top++;

i++;

}

else if (after[i] == '\*')

{

x2 = result[--top];

x1 = result[--top];

result[top] = x1 \* x2;

top++;

i++;

}

else if (after[i] == '/')

{

x2 = result[--top];

x1 = result[--top];

result[top] = x1 / x2;

top++;

i++;

}

}

//打印后缀表达式计算值

printf("\n经过计算后，这个中缀表达式的值为: %.3lf\n", result[0]);

}

**<random.cpp>:**

#include <iostream>

#include"char\_stack.h"

using namespace std;

int main()

{

bool flag;

char\_stack cs;

/\*flag = IsPlalindrome();

if (!flag)

{

printf("此字符串不是回文字符串！\n");

}

else printf("此字符串是回文字符串！\n");\*/

CalPostFix(&cs);

/\*线性表

int\_list dlist[2];

int\_list\* li;

Initiate(&dlist[0]);

Initiate(&dlist[1]);

Input(&dlist[0], 10);

Input(&dlist[0], 13);

Input(&dlist[0], 26);

Input(&dlist[0], 31);

Input(&dlist[0], 46);

Input(&dlist[0], 54);

Input(&dlist[1], 8);

Input(&dlist[1], 17);

Input(&dlist[1], 39);

Input(&dlist[1], 40);

Input(&dlist[1], 100);

Input(&dlist[1], 200);

printf("合并前的两个顺序表:");

Display(&dlist[0]);

Display(&dlist[1]);

li = Assemble(dlist);

printf("合并后的顺序表:");

Display(li);

Seperate(li);

\*/

/\*顺序表

sequence\_list li1;

/\*sequence\_list li2;

sequence\_list li3;

Input(&li1);

Input(&li2);

Inter(&li1, &li2, &li3);

Input(&li1);

Partion(&li1);\*/

system("pause");

return 0;

}