**西南石油大学实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目编号** | 001063000502 | **项目名称** | **特殊线性表的应用** | **成绩** |  |
| **专业年级** | **软件工程2022级** | | | **指导教师** | **胡卫东** |
| **姓名** | **李浩楠** | **学号** | **202231060920** | **实验日期** | **2023/6/3** |

1. **实验目的**
2. 熟悉并掌握对栈和队列的结构与操作。
3. 使用顺序存储结构和链式存储结构对栈和队列实现相关操作。
4. 利用C/C++语言对相关操作进行实现，加深对相关操作的理解。
5. **实验工具**

PC微机、Windows、PTA或Dev C++。

1. **实验步骤**
2. 排队叫号系统。

编写程序实现银行排队叫号系统，采用链队列作为存储结构。

1. 括号匹配。

下面程序可实现表达式中括号匹配检查，约定只有‘()’、'[]'、'{}'三种括号。请将以下程序补充完整。

1. 软硬车厢交替排列。

设车辆厂生产了硬座车厢和软座车厢共n节（混合在一起），要求使用队列的基本操作，编程实现所有硬座车厢和所有软座车厢交替排列。例如硬座车厢用H来表示，软座车厢用S来表示，从键盘上输入8节车厢代号为SHHSSHSS ，输出为HSHSHSSS。若从键盘输入8节车厢代号为SHHSHHSH ，输出为HSHSHSHH。

1. 后缀式求值（一位整型操作数版）。

我们人类习惯于书写“中缀式”，如 3 + 5 \* 2 ，其值为13。 (p.s. 为什么人类习惯中缀式呢？是因为中缀式比后缀式好用么？）

而计算机更加习惯“****后缀式****”（也叫“逆波兰式”，****Reverse Polish Notation****）。上述中缀式对应的后缀式是： 3 5 2 \* +

现在，请对输入的后缀式进行求值。为了简化输入处理和运算，运算数（操作数）不超过300个且均为1位正整数，运算符（操作符）仅有+ - \* /（加减乘除）四种，运算数和运算符之间没有空格间隔，且****题目保证运算的中间结果和最终结果都在整型范围内****。

但是注意，题目输入的后缀式可能错误，例如：

* 1234+- 错误，缺少运算符
* 123+-\* 错误，缺少运算数
* 122-/ 错误，除数为0

题目保证以上三种错误不会同时发生。

### 输入格式:

第一行给出一个不超过10的正整数k；

接下来k行，每行给出一个后缀式，后缀式的格式如上文所描述，1位操作数且无空格间隔。

### 输出格式:

输出有k行，对于所输入的每个后缀式，判断是否正确（可求值），并在一行里输出：

* 如果后缀式无误、可求值，输出结果
* 如果发现除数为0，则输出Division By Zero!
* 如果发现其它错误，则输出Expression Error!

1. **实验结果**
2. 测试用例：1 1 1 2 2 2 2 0

代码：Status InitLinkQueue(LinkQueue &Q)

{

Q.front=Q.rear=new QNode;

Q.front->next=NULL;

return OK;

}

Status EnLinkQueue(LinkQueue &Q,QElemType e)

{

QueuePtr p;

p=new QNode;

p->data=e;

p->next=NULL;

Q.rear->next=p;

Q.rear=p;

return OK;

}

Status DeLinkQueue(LinkQueue &Q,QElemType &e)

{

QueuePtr p;

if(Q.front==Q.rear)

return ERROR;

p=Q.front->next;

e=p->data;

Q.front->next=p->next;

if(Q.rear==p)

Q.rear=Q.front;

delete p;

return OK;

}

Status QueueEmpty(LinkQueue Q)

{

if(Q.front->next==NULL&&Q.rear->next==NULL)

return OK;

}

运行结果：您的序号为:1

您的序号为:2

您的序号为:3

>>请1号办理业务

>>请2号办理业务

>>请3号办理业务

>>没有排队的客户!

1. 测试用例：{[4+2\*(2+3)]-1}#

代码：Status push(Sqstack &S,SElemType x) //x入栈S

{

if(S.top-S.base==S.stacksize)

return ERROR;

\*S.top++=x;

return OK;

}

Status pop(Sqstack &S,SElemType &e)//从S栈出栈1次元素放入e

{

if(S.top==S.base)

return ERROR;

e=\*--S.top;

return OK;

}

Status Compare(char s[]) //s为表达式

{

Sqstack S;

SElemType e;

Status flag=TRUE;

int i=0;

iniStack(S);

while(s[i]!='#' && flag==TRUE )

{

switch(s[i])

{

case '(':

case '[':

case '{':push(S,s[i]);break;

case ')': if(pop(S,e)==ERROR || e!='(')//如果是(

flag=FALSE;break;

case ']': if(pop(S,e)==ERROR || e!='[')//如果是[

flag=FALSE;break;

case '}': if(pop(S,e)==ERROR || e!='{')//如果是{

flag=FALSE;break;

}

i++;

}

if(flag==TRUE && s[i]=='#' && S.top==S.base)

return TRUE;

else

return FALSE;

}

运行结果：检查结果：括号匹配！

1. 测试用例：8

SHHSSHSS

代码：#include<bits/stdc++.h>

using namespace std;

int main()

{

int n;

string s;

cin>>n>>s;

if(n<2||n>10){//如果输入的n节车厢不合法直接ERROR

cout<<"ERROR";

}else{

int ls=0;//统计软座的个数

int lh=0;//统计硬座的个数

for(int i=0;i<s.size();i++){

if(s[i]=='S'){

ls++;

}else{

lh++;

}

}

while(ls>0||lh>0){

if(lh>0){

cout<<"H";

lh--;

}

if(ls>0){

cout<<"S";

ls--;

}

}

}

return 0;

}

运行结果：HSHSHSSS

1. 测试用例：2

1234+-\*

123+4-\*

2

12+

1234+-

2

2222+-\*

22+-\*

1

2222-/+

代码：#include<bits/stdc++.h>

using namespace std;

stack<int> ans;

int main()

{

int n;

cin >> n;

while(n --)

{

string s;

cin >> s;

int num = 0, way = 0;

bool flag = 0;

for(int i = 0; i < s.size(); i ++)

{

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

{

num ++;

}

if(s[i] == '+' || s[i]== '-' ||s[i] == '\*' || s[i] == '/')

{

way ++;

}

}

if(num - way != 1)

{

cout << "Expression Error!" << endl;

continue;

}

for(int i = 0; i < s.size(); i ++)

{

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

{

ans.push(s[i] - '0');

}

if (s[i] == '+')

{

int a = ans.top();

ans.pop();

int b = ans.top();

ans.pop();

ans.push(a + b);

}

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

{

int a = ans.top();

ans.pop();

int b = ans.top();

ans.pop();

ans.push(b - a);

}

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

{

int a = ans.top();

ans.pop();

int b = ans.top();

ans.pop();

ans.push(b \* a);

}

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

{

int a = ans.top();

ans.pop();

int b = ans.top();

ans.pop();

if (a)

{

ans.push(b / a);

}

else

{

cout << "Division By Zero!" << endl;

flag = 1;

break;

}

}

}

if (flag)

{

continue;

}

cout << ans.top() << endl;

}

return 0;

}

运行结果：-5

1

3

Expression Error!

-4

Expression Error!

Division By Zero!

1. **实验总结**
2. 栈是限定仅在表尾进行插入或删除的线性表，又称为后进先出的线性表。栈有两种友储表示，顺序表示（顺序栈）和链式表示（链栈）。栈的主要操作是进栈和出栈，对于顺序栈的进栈和出栈操作要注意判断栈满或栈空。
3. 队列是一种先进先出的线性表。它只允许在表的一端进行插入，而在另一端进行删除。队列也有两种存储表示，顺序表示（循环队列）和链式表示（链队）。队列的主要操作是进队和出队，对于顺序表示的循环队列的进队和出队操作要注意判断队满或队空。凡是涉及队头或队尾指针的修改都要将其对MAXQSIZE模。

3.栈和队列是在程序设计中被广泛使用的两种数据结构，其具体的应用场景 都是与其表示方法和运算规则相互联系的。