《数据结构》上机报告

2020 年 10 月 20 日

姓名：马家昱　学号：1950509　班级：计科 1 班　得分：300

<table>
<tr><td>实验题目</td><td colspan="2">栈</td></tr>
<tr><td>问题描述</td><td colspan="2">栈，是限定仅在表尾进行插入或删除操作的线性表，具有后进先出（LIFO）的特性。这一端称为栈顶，相对的另一端称为栈底。向一个栈插入新元素又称作进栈、入栈或压栈，它是把新元素放到栈顶元素的上面，使之成为新的栈顶元素；从一个栈删除元素又称作出栈或退栈，它是把栈顶元素删除掉，使其相邻的元素成为新的栈顶元素。</td></tr>
<tr><td rowspan="2">基本要求</td><td colspan="2">1. (p1)实现栈的基本操作，包括栈的建立、入栈、出栈、栈判空
栈判满、取栈顶元素、栈的遍历。
2. (p2)栈的基本应用：表达式求值，表达式中只包含+-*/四种运算符和括号（）。
3. (p3)解决“列车进站”问题，其基本思路使用到栈的调用。
4. 运用栈模拟阶乘函数的调用过程，用栈模拟 n 的阶乘的递归调用过程。</td></tr>
<tr><td>已完成基本内容（序号）:</td><td>1，2，3，4</td></tr>
<tr><td rowspan="2">选做要求</td><td colspan="2">1. 程序要添加适当的注释，程序的书写要采用缩进格式。
2. 程序要具有一定的健壮性，当输入数据或操作非法时，程序应当拥有报错提示。
3. 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。</td></tr>
<tr><td>已完成选做内容（序号）</td><td>1，2，3</td></tr>
<tr><td>数据结构设计</td><td colspan="2">本实验第一题，实现栈的基本操作中，使用了两种实现方法，分别是链栈与顺序栈，从底层开始搭建栈的基本结构，包括数据节点与栈底栈顶指针，并且通过具体函数实现了入栈、出栈、遍历等功能。
第二题、第三题栈的应用与运用栈模拟阶乘函数的调用过程中，均直接使用 c++STL 标准模板库中的栈类，自己定义了数据节点的类型。</td></tr>
</table>

功能(函数)说明	1. 栈的基本操作 **链栈：** 栈的节点类与节点类的初始化 `class SNode {//栈的节点类` `public:` `int data;` `SNode* next;` `public:` `SNode();` `void assign();` `};` `SNode::SNode() {//节点初始化` `data = 0;` `next = (SNode*)malloc(sizeof(SNode));` `next = NULL;` `}` `void SNode::assign() {//给节点赋值` `cin >> this->data;` `}` 栈类与栈类的初始化 `class Stack {//栈类` `public:` `SNode* base;//栈底` `SNode* top;//栈顶` `int stackSize;//容量` `int size;//实际大小` `public:` `Stack();//初始化` `void sizeInit();//给栈设置最大容量` `bool push();//入栈` `int pop();//出栈` `void printStack();//遍历打印` `};` `Stack::Stack() {` `top = base = (SNode*)malloc(sizeof(SNode));//栈底与栈顶均指向同一个节点` `size = 0;` `stackSize = 0;` `}` `void Stack::sizeInit() {` `cin >> this->stackSize;` `}` bool push();入栈

```
bool Stack::push() {
    if (size == stackSize)//栈满
        return false;
    top->assign();
    SNode* newNode = (SNode*)malloc(sizeof(SNode));
    newNode->next = top;
    top = newNode;//top指针移动到新节点
    size++;
    return true;
}
```

int pop();出栈

```
int Stack::pop() {
    if (size == 0)
        return 0;
    int result = top->next->data;
    top = top->next;//top指针下移
    size--;
    return result;
}
```

void printStack();遍历打印栈

```
void Stack::printStack() {//遍历打印
    SNode* cur = top->next;
    for (int i = 0; i < size; i++) {
        cout << cur->data << ' ';
        cur = cur->next;
    }
    cout << endl;
}
```

顺序栈：

顺序栈类及其初始化

```
class Stack {//顺序栈类
public:
    int* elem;//首地址
    int length;//当前元素个数
    int size;//大小
public:
    Stack(int n);//初始化栈的长度
    bool push(int i);//入栈
    int pop();//出栈
    void printStack();//遍历打印
};

Stack::Stack(int n) {//初始化函数，申请空间，将长度与容量置零
    elem = (int*)malloc(sizeof(int) * n);
    size = n;
    length = 0;
}
```

Bool push（int i）入栈

```
bool Stack::push(int i) {//入栈，栈已满则返回false
    if (length + 1 > size)
        return false;
    elem[length] = i;
    length++;
    return true;
}
```

Int pop（）出栈

```
int Stack::pop() {//出栈，返回出栈元素，栈空则返回零
    if (length == 0)
        return 0;
    length--;
    return elem[length];
}
```

Void printStack（）遍历打印

```
void Stack::printStack() {//倒序遍历打印
    for (int i = length-1; i!=-1; i--)
        cout << elem[i]<<' ';
}
```

2. 表达式求值

bool In（char c）判断是否为运算符

```
bool In(char c) {//判断是否为运算符
    if (c == '+' || c == '-' || c == '*' || c == '/' || c == '(' || c == ')' || c == '=')
        return true;
    else return false;
}
```

Char Precede（char a，char b）比较两个运算的优先级

```
char Precede(char a, char b)//比较两个运算符的优先级，返回值为< > =
{
    char result;
    if (a == '+' || a == '-')
    {
        if (b == '+' || b == '-' || b == ')' || b == '=')
            result = '>';
        else if (b == '*' || b == '/' || b == '(')
            result = '<';
    }
    else if (a == '*' || a == '/')
    {
        if (b == '+' || b == '-' || b == '*' || b == '/' || b == ')' || b == '=')
            result = '>';
        else if (b == '(')
            result = '<';
    }
    else if (a == '(')
    {
        if (b == '+' || b == '-' || b == '*' || b == '/' || b == '(')
            result = '<';
        else if (b == ')')
            result = '=';
    }
    else if (a == ')')
    {
        if (b == '+' || b == '-' || b == '*' || b == '/' || b == ')' || b == '=')
            result = '>';
    }
    else if (a == '=')
    {
        if (b == '+' || b == '-' || b == '*' || b == '/' || b == '(')
            result = '<';
        else if (b == '=')
            result = '=';
    }
    return result;
}
```

int operate（int a，char theta，int b）计算一个二元运算符的结果

```
int Operate(int a, char theta, int b) {//计算一个二元运算符的结果，并返回其值
    int result;
    if (theta == '+')
        result = a + b;
    else if (theta == '-')
        result = a - b;
    else if (theta == '*')
        result = a * b;
    else if (theta == '/')
        result = a / b;
    return result;
}
```

Int EvaluateExoression（）表达式求值

```
int EvaluateExpression() {//表达式求值
    stack<char> OPTR;//存放运算符的栈
    stack<int> OPND;//存放操作数的栈
    stack<char> singleNum;//用来计算单个操作数的值的栈
    OPTR.push('=');
    char theta;
    int c;
    int a, b;
    c = getchar();
    while (c != '=' || OPTR.top() != '=') {
        if (!In(c)) {//如果是操作数
            singleNum.push(c - '0');
            int cnt = 1;
            while (!In(c = getchar())) {//继续输入，直至输入一个运算符
                singleNum.push(c - '0');
                cnt++;
            }
            int num = 0;
            for (int i = 0; i < cnt; i++) {//计算这个多位数的数值
                num += singleNum.top() * pow(10, i);
                singleNum.pop();
            }
            OPND.push(num);//操作数入栈
        }
        else switch (Precede(OPTR.top(), c)) {//比较运算符优先级
        case '<'://直接入栈
            OPTR.push(c);
            c = getchar();
            break;
        case '='://左右括号相遇，出栈
            OPTR.pop();
            c = getchar();
            break;
        case '>'://计算
            theta = OPTR.top();
            OPTR.pop();
            b = OPND.top();
            OPND.pop();
            a = OPND.top();
            OPND.pop();
            if (b == 0 && theta == '/')//若出现除0运算，则返回
                return -99999999;
            OPND.push(Operate(a, theta, b));
            break;
        }
    }
    if (OPND.empty())
        return 0;
    else return OPND.top();
}
```

3. 列车进站

int initDeque(stack<char>& init)入口处栈的初始化

```
int initDeque(stack<char>& init) {
    stack<char> temp;
    char c;
    int cnt = 0;
    while ((c = getchar()) != '\n') {
        temp.push(c);
        cnt++;
    }
    while (!temp.empty()) {
        init.push(temp.top());
        temp.pop();
    }
    return cnt;
}
```

bool dispatching(stack<char> entry, char* queue)调度函数，返回出站方案是否可行

```
bool dispatching(stack<char> entry, char* queue) {//模拟出站函数 entry为出口处已经排好的栈，queue为需要判断是否可行的顺序
    stack<char> station;//车站
    int i = 0;
    while (true) {
        if (entry.empty() && station.empty())//车站和入口都空，即可行
            return true;
        if (entry.empty() && station.top() != queue[i])//入口空，且车站栈顶元素不等于queue当前元素，不可行
            return false;
        if (!entry.empty()) {
            if (entry.top() != queue[i] && station.empty()) {//将出口栈顶元素入栈至车站
                station.push(entry.top());
                entry.pop();
            }
            else if (entry.top() != queue[i] && station.top() != queue[i]) {//将出口栈顶元素入栈至车站
                station.push(entry.top());
                entry.pop();
            }
            else if (entry.top() == queue[i]) {//出车站
                entry.pop();
                i++;
            }
            else if (station.top() == queue[i]) {//出车站
                station.pop();
                i++;
            }
        }
        else if (entry.empty())
            if (station.top() == queue[i]) {
                station.pop();
                i++;
            }
    }
}
```

4. 利用栈模拟阶乘递归调用

节点类，包括输入参数与函数返回地址

	`` class Data {//数据类，包括输入参数与返回地址 public: int n;//输入参数 int* returnAddress; public: Data() {//构造器 n = 0; returnAddress = (int*)malloc(sizeof(int)); } void setter(int n) {//设定的值n的值 this->n = n; } }; Int factorial（int n）利用栈求 n 的阶乘 int factorial(int n) {//利用栈模拟递归求阶乘 if (n == 0)//0的阶乘为1； return 1; stack<Data> mystack;//栈 Data data;//存储最终结果的数据类 data.setter(n);//初始化 while (data.n != 1) {//n=1? mystack.push(data);//开始递归 data.setter(data.n - 1);//入栈 } while (!mystack.empty()) {//栈空? if (mystack.top().n == 1) {//当n=1时，return1 data.n = 1; mystack.pop(); } else { data.n *= mystack.top().n;//回溯 mystack.pop(); } } return data.n;//end } ```
开发环境	Windows10 Microsoft visual studio 2019

调试分析

1. 栈的基本操作

```
Microsoft Visual Studio 调试控制台
4
pop
Stack is Empty
push 10
push 2
push 3
pop
3
pop
2
push 1
push 2
push 3
push 4
Stack is Full
quit
3 2 1 10
```

2. 表达式求值

```
Microsoft Visual Studio 调试控制台
4+2*3-10/4=
8
```

3. 列车进站

```
C:\Users\majiayu\source\repos\Stack\Debug\Stack.exe
abc
abc
yes
acb
yes
bac
yes
bca
yes
cab
no
cba
yes
```

4. 利用栈模拟阶乘递归调用

```
C:\Users\majiayu\source\repos\Stack\Debug\Stack.exe
Please enter the factorial you want:
5
The result is:120
```

心得体会

（对整个实验过程做出总结，对重要的算法做出性能分析。）

算法复杂度分析：

栈的基本操作中，入栈与出栈的时间复杂度为 O（1），遍历打印的时间复杂度为 O（n）。

表达式求值中，对总数为 n 的操作数与操作符进行运算时，时间复杂度为 O（n）

列车进站中，最坏情况下时间复杂度为 O（n）

对利用栈模拟阶乘递归的测试：

利用递归求阶乘时，当 n 达到 4790 时，出现 stack overflow 错误

```
#define _CRT_SECURE_NO_WARNINGS
#include <stack>
#include <iostream>
using namespace std;

long f(int n) {
    if (n == 1) return 1; //停止调用
    return n * f
}

int main() {
    cout << f(50
}
```

未经处理的异常

0x006017B9 处有未经处理的异常(在 Stack.exe 中): 0xC00000FD: Stack overflow (参数: 0x00000001, 0x01202FA4)。

复制详细信息 | 启动 Live Share 会话...

异常设置

用栈消解递归后则不会出现报错。

心得体会

通过本次关于栈的实验，我从底层开始进一步深入了解了栈的工作原理，亲自实现了与栈有关的操作。在表达式求值与列车进站题目中，使用了 STL 标准模板库中的栈，实现了栈的经典应用。最后通过深入了解递归的原理，使用栈模拟递归的调用，更加熟悉了递归的本质。