HW2 栈和队列 实验报告

姓名：徐恒 学号：2050035 日期：2022 年 4 月 26 日

1.涉及数据结构和相关背景

栈（stack）又名堆栈，它是一种运算受限的线性表。限定仅在表尾进行插入和删除操作的线性表。这一端被称为栈顶，相对地，把另一端称为栈底。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。

2. 实验内容

2.1 栈的基本操作

2.1.1 问题描述

栈是限制仅在表的一端插入和删除的线性表。栈的操作简单，重点掌握栈具有后进先出（LIFO）的特性。顺序栈是栈的顺序存储结构的实现。链栈是栈的链式存储结构的实现。 本题练习顺序栈的基本操作，包括入栈、出栈、判栈空、判栈满、取栈顶元素、栈的遍历。

2.1.2 基本要求

搭建顺序栈，实现顺序栈的基本操作，包括入栈、出栈、判栈空、判栈满、取栈顶元素、栈的遍历。

2.1.3 数据结构设计

先输入栈的容量，然后建立顺序栈。借助 while 循环输入要执行的操作函数，并进行函数的运行。其中，输入 pop 执行出栈，输入 push 执行入栈，输入 quit 执行栈的遍历出栈、终止程序。

2.1.4 功能说明（函数、类）

void pop(stack &st)：出栈
void push(stack &st,int p)：入栈
void quit(stack &st)：遍历出栈，结束程序

2.1.5 调试分析（遇到的问题和解决方法）

一开始未在函数的变量前添加&，使得函数操作无效。

2.1.6 总结和体会

顺序栈的建立与操作需熟练掌握。

2.2 列车进站

2.2.1 问题描述

每一时刻，列车可以从入口进车站或直接从入口进入出口，再或者从车站进入出口。即每一时刻可以有一辆车沿着箭头 a 或 b 或 c 的方向行驶。 现在有一些车在入口处等待，给出该序列，然后给你多组出站序列，请你判断是否能够通过上述的方式从出口出来。

2.2.2 基本要求

若给定的出栈序列可以得到，输出 yes,否则输出 no。

2.2.3 数据结构设计

首行输入入站序列，之后多行表示多个出栈序列。对合理的出栈序列输出 yes,不然输出 no。

2.2.4 功能说明（函数、类）

int InitStack(SqStack &S)　　//对栈 S 初始化
int Push(SqStack &S,char e)　//将元素 e 入栈
int Pop(SqStack &S)　　//对栈 S 进行出栈操作
int judge(char c[],char d[]) //d 指向出栈序列，c 指向入栈序列，依次函数判断 d 作为 c 的出栈序列是否合理，若合理返回 0，不然返回-1

2.2.5 调试分析（遇到的问题和解决方法）

题目要求输入=EOF 时程序终止，一开始将其理解为输入字符“EOF”，导致 the limit exceeded。后查阅了相关资料发现，cin 时，若输入为 EOF(Ctrl+Z)，则返回值为 0，故借助语句 while((cin>>d)!=0) 将程序完善。

2.2.6 布尔表达式

本题的难点在于如何判断对给定的入栈序列，出栈序列是否合理，这个过程需要经过详细的分析。而自己在 EOF 时犯得错误主要由于对其理解不到位。

2.3 顺序表的去重

2.3.1 问题描述

在布尔表达式中，V 表示 True，F 表示 False，|表示 or，&表示 and，！表示 not（运算符优先级 not> and > or）。对给出的布尔表达式进行计算，True 输出 V，False 输出 F。

2.3.2 基本要求

输入时采用文件输入，有若干（A<=20）个表达式，其中每一行为一个表达式。 表达式有（N<=100）个符号，符号间可以用任意空格分开，或者没有空格，所以表达式的总长度，即字符的个数，是未知的。

输出时，对测试用例中的每个表达式输出“Expression”，后面跟着序列号和“: ”，然后是

相应的测试表达式的结果（V 或 F），每个表达式结果占一行（注意冒号后面有空格）。

2.3.3 数据结构设计

对输入的布尔型表达式（即字符串 s）进行预处理，即去除其中的空格与"\r"，并将处理好的表达式储存到 temp 中。初始化两个栈 OPTR 和 OPND，分别储存运算符和操作数。定义优先级函数与对应不同字符的运算函数，对布尔表达式进行运算。

2.3.4 功能说明（函数、类）

int InitStack(SqStack &S)　　//对栈 S 进行初始化
int Push(SqStack &S,char e) //将元素 e 入栈
int Pop(SqStack &S,char &e) //对栈 S 作出栈处理，并将出栈元素返回值给 e
char Precede(char a,char b) //优先级函数，用于比较运算符 a 和 b 的优先级
char Operate1(char a,char b,char theta)//运算函数，对 a,b 作 theta 的运算(theta 不能取！)
char Operate2(char a,char theta) //运算函数，对 a 作 theta 的运算(theta 只取！)
int EvaluateExpression(string s) //对处理过的布尔表达式 s 运算

2.3.5 调试分析（遇到的问题和解决方法）

Windows 操作系统中结尾采用"\r\n",但测试平台使用的时 linux，结尾为"\r"。本地调试时没有出错，但提交后出错，因为没有考虑\r 也被输入字符串 s 中了，后在对字符串处理中对\r 进行剔除解决。

2.3.6 总结和体会

难点在于如何去定义优先级和运算函数，而最后出现的\r 和\r\n 的问题是由于自身知识积累还不足，一些常识性问题不清楚。

2.4 队列的应用

2.4.1 问题描述

输入一个 n*m 的 0 1 矩阵，1 表示该位置有东西，0 表示该位置没有东西。所有四邻域联通的 1 算作一个区域，仅在矩阵边缘联通的不算作区域。求区域数。此算法在细胞计数上会经常用到。

2.4.2 基本要求

要求输入时
第 1 行为 2 个正整数 n，m，表示要输入的矩阵行数和列数
第 2—n+1 行为 n*m 的矩阵，每个元素的值为 0 或 1。

2.4.3 数据结构设计

先将待考察矩阵存放在二维数组 a[ ][ ]中并定义技术变量 k=0 作为计数指标，然后对矩阵内部元素（不在边界处，即对 a[i][j]考虑对 i 从 1—n-1 的循环，对 j 从 1—m-1 的循环）逐个考察，用 check 函数进行标记，每成功找到一个区域，k 就自增一次。最终输出 k，即为矩阵中区域个数。

定义 check 函数时考虑递归。若元素不在边界处，对其四邻域的元素做 check 标记，依次往复

2.4.4 功能说明（函数、类）

int a[1000][1000]; //定义全局变量数组 a，用来存放矩阵元素
int check(int i,int j,int n,int m) //定义 check 函数，用于判断矩阵中以（i,j）为中心的四邻域是否为区域，若为区域将该区域中的元素全部标记为-1；不然不对其进行操作。n 和 m 是数组维界。

2.4.5 调试分析（遇到的问题和解决方法）

忘记考虑在边界处的情况，即最初提交时主函数中循环是 for(i=0;i<n;i++) for(j=0;j<m;j++)，导致程序出错

另外刚开始忘记定义 a[ ][ ]为全局变量，在 check 函数中对 a 做运算会无定义

2.4.6 总结和体会

未使用到队列，在 check 中借助 if 和递归将程序搞定。

2.5 一元多项式的相加与相乘

2.5.1 问题描述

给定一个队列，有下列 3 个基本操作:
（1）Enqueue(v): v 入队
（2）Dequeue(): 使队首元素删除，并返回此元素
（3）GetMax(): 返回队列中的最大元素
请设计一种数据结构和算法，让 GetMax 操作的时间复杂度尽可能地低。
要求运行时间不超过一秒

2.5.2 基本要求

第 1 行 1 个正整数 n，表示队列的容量(队列中最多有 n 个元素)

接着读入多行，每一行执行一个动作。

若输入"dequeue"，表示出队，当队空时，输出一行“Queue is Empty”;否则，输出出队的元素；

若输入"enqueue m"，表示将元素 m 入队,当队满时(入队前队列中元素已有 n 个)，输出"Queue is Full"，否则，不输出；

若输入"max",输出队列中最大元素，若队空，输出一行“Queue is Empty”。

若输入"quit",结束输入，输出队列中的所有元素。

2.5.3 数据结构设计

为降低 GetMax 操作的时间复杂度，定义一个双向队列 D。从队列性质我们可以看出每一个元素 e 前面比他小的元素对取最大值操作的结果没有影响。故从队列尾部插入元素时，我们可以提前取出队列中所有比这个元素小的元素，使得队列中只保留对结果有影响的数字。这样的方法等价于要求维持队列单调递减，即要保证每个元素的前面都没有比它小的元素。而弹出元素时需要判断弹出的是否为该最大值。最终操作完成后 D 的队首元素即为队列最大值。

为作为比对，我们还需定义一个正常队列 Q。为确定队列容量后方便操作，定义为循环

队列。

输入 n 确定队列的容量。分别定义两个队列，一个用于正常存放元素（Q），一个用于快速判断队列的最大值（D）。输入元素 e 时，Q 队列正常输入，对 D 队列队尾元素进行判断，将队尾元素中小于 e 的依次弹出，直至队尾元素大于等于 e 或队列为空停止，并将元素 e 从队尾输入队列。弹出元素 e 时，Q 队列正常弹出，对 D 队列队首元素判断，若队首元素正是 e，则弹出，并将队首指向下一位置，否则对 D 不做操作。最终求 max 时，只需输出 D 队列队首元素即可。

2.5.4 功能说明（函数、类）

int InitQueue(SqQueue &Q,int n)　//对容量为 n 的队列 Q 进行初始化操作
int EnQueue(SqQueue &Q,SqQueue &D,int e,int n) //元素 e 分别输入正常队列 Q 和判断最大值的双向队列 D 中，其中 n 代表两个队列的容量
int DeQueue(SqQueue &Q,SqQueue &D,int &e,int n) //将正常队列 Q 和判断最大值的双向队列 D 分别弹出元素 e，其中 n 代表两个队列的容量

2.5.5 调试分析（遇到的问题和解决方法）

由于 max 操作时未考虑队列为空的情况，导致出现 wrong。后添加判断语句 if 后解决。

2.5.6 总结和体会

本题重点在于如何简化取 max 时的操作，降低时间复杂度。由于题目要求，对最终队列中的元素一个一个比较判断显然是不可取。通过分析和搜集查找资料后，采用另外建立一个双向队列 D 的方法来存放原队列的最大值，这样可以直接提取出队列最大值。

3. 实验总结

经过本章的学习，不仅对栈和队列的内容和操作更加熟练，也进一步理解了数据结构。在对一些算法的分析中也常常惊叹于其精妙之处，有了很大的收获。