作业 HW2 实验报告 姓名：陆明奇 学号：2050283 日期：2021 年 10 月 31 日 1. 涉及数据结构和相关背景 栈是限定仅在表尾进行插入或删除的线性表。栈的修改是按后进先出的原则 进行的。因此，栈又称为后进先出的线性表（LIFO 结构）。和栈相反，队列是 一种先进先出（FIFO）的线性表。它只允许在表的一端进行插入，而在另一 端删除元素。 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。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 第 1 行 1 个正整数 n, 表示栈的容量。 接着读入多行，每一行执行一个动作。 若输入"pop"，表示出栈，当栈空时，输出一行“Stack is Empty”;否则，输出出 栈的元素。 若输入"push 10"，表示将元素 10 入栈,当栈满时，输出"Stack is Full"，否则， 不输出。 若输入"quit",输出栈中所有元素，以空格分割。 2.1.3 数据结构设计 此题考察的是栈的基本操作。因为题目给定了栈的最大容量，所以我采用栈 的顺序存储结构——顺序栈。在操作中不再对栈的容量作任何修改。 2.1.4 功能说明（函数、类） bool InitStack(Node& s, int n) 功能：初始化一个最大容量为 n 的栈。 Para s: 顺序栈 Para n：栈的最大容量 返回值：初始化栈是否成功 bool Push(Node& s, int e) 功能：插入元素 e 为新的栈顶元素 Para s: 顺序栈 Para e: 插入的元素 返回值：插入元素操作是否成功 bool Pop(Node& s, int& e) 功能：若栈不空，则删除 s 的栈顶元素，用 e 返回其值 Para s: 顺序栈 Para e: 被删除的元素 返回值：删除元素操作是否成功 2.1.5 调试分析（遇到的问题和解决方法） 最初各个函数的参数没有传入栈 s 的引用，导致函数对栈的各个操作都没有 实现。之后我把参数改为栈 s 的引用之后，问题得到解决。 2.1.6 总结和体会 顺序栈，即栈的顺序存储结构是利用一组地址连续的存储单元依次存放自栈 底到栈顶的数据元素，同时附设指针 top 指示栈顶元素在顺序栈中的位置。每当 插入新的元素时，指针 top+1，删除栈顶元素时，指针 top-1。因此，非空栈中 的栈顶指针始终在栈顶元素的下一个位置上。 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 第 1 行，输入一个串，入站序列。后面多行，每行一个串，表示出栈序列 当输入=EOF 时结束。输出多行，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 列车进站有一个特点:后进站的列车需要先出站。满足栈的先进后出结构，因 此采用栈这一数据结构来模拟列车进站。 2.2.4 功能说明（函数、类） bool InitStack(Node& s,) 功能：初始化一个顺序栈。 Para s: 顺序栈 返回值：初始化栈是否成功 bool Push(Node& s, char e) 功能：插入元素 e 为新的栈顶元素 Para s: 顺序栈 Para e: 插入的元素 返回值：插入元素操作是否成功 bool Pop(Node& s, char& e) 功能：若栈不空，则删除 s 的栈顶元素，用 e 返回其值 Para s: 顺序栈 Para e: 被删除的元素 返回值：删除元素操作是否成功 char showtop(Node s) 功能：若栈不空，显示栈的栈顶元素 Para s: 顺序栈 返回值：栈的栈顶元素 2.2.5 调试分析（遇到的问题和解决方法） 最初的算法是设置一个指向输入串中元素的指针来和输出串一一比对，若元 素不匹配则进栈。但这样考虑是不周全的，列车并不一定在最后一起出站。正确 的算法需要设置两个指针分别指向输入串和输出串。若元素匹配，则两个指针均 向后移动一位。若不匹配，则输入串指针指向的当前元素进栈，指针+1，直到找 到匹配元素。 2.2.6 总结和体会 这道题是利用栈解决实际问题的一个例子。需要注意的是，在判断输入串和 输出串元素是否匹配时，需要同时考虑栈顶的元素和输入串指针当前指向的元素。 若两个均不匹配，再进行入栈操作。 2.3 布尔表达式 2.3.1 问题描述 计算如下布尔表达式 ( V | V ) & F & ( F | V ) 其中 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） 2.3.2 基本要求 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。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。 2.3.3 数据结构设计 本题采用“算符优先法”。首先建立两个栈，一个称做 OPTR，用以寄存运算符。 另一个称做 OPND，用以寄存操作数或运算结果。算法基本思想：首先置操作数 栈为空栈，表达式起始符“#”为运算符栈的栈底元素，依次读入表达式中的每个 字符，若是操作数则进 OPND 栈，若是运算符则和 OPTR 栈的栈顶运算符比较 优先级后作相应操作，直至整个表达式求值完成。 2.3.4 功能说明（函数、类） int Precede(char a, char b) 功能：比较a，b两个运算符的优先级 Para a：运算符a Para b：运算符b 返回值：a和b优先级的比较结果 bool InitStack(Node& s,) 功能：初始化一个顺序栈。 Para s: 顺序栈 返回值：初始化栈是否成功 bool Push(Node& s, char e) 功能：插入元素 e 为新的栈顶元素 Para s: 顺序栈 Para e: 插入的元素 返回值：插入元素操作是否成功 bool Pop(Node& s, char& e) 功能：若栈不空，则删除 s 的栈顶元素，用 e 返回其值 Para s: 顺序栈 Para e: 被删除的元素 返回值：删除元素操作是否成功 char showtop(Node s) 功能：若栈不空，显示栈的栈顶元素 Para s: 顺序栈 返回值：栈的栈顶元素 char Operate(char a, char theta, char b) 功能：对运算符和操作数进行求值 Para a：操作数 a Para theta：运算符 Para b：操作数 b 返回值：求值的结果 bool ifOPND(char c) 功能：判断是否为操作数 Para c：读入的字符 返回值：判断的结果 void EvaluateExpression() 功能：对表达式进行计算求值 返回值：无返回值 2.3.5 调试分析（遇到的问题和解决方法） 程序最初提交到 OJ 上一直显示超时，怀疑可能是 getchar（）读入时没有读 到字符故一直处于等待状态。之后对于行末回车以及文件结束标志 EOF 作了特 殊判断后，成功通过了测评。 2.3.6 总结和体会 这道题是利用栈解决实际问题的一个例子。需要注意的是，与和或都是二元 运算符，而非是一元运算符，处理是需要特殊判断。同时，对于文件输入这一形 式也要格外注意，回车以及文件结束标志 EOF 等需要特殊判断。 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。 2.4.2 基本要求 输入 2 个正整数 n，m, 表示要输入的矩阵行数和列数。接着输入 n\*m 的矩 阵，每个元素的值为 0 或 1。输出 1 行，代表区域数。 2.4.3 数据结构设计 本题的要求是找到连通 1 组成的区域的个数。即由一个 1 出发向上下左右分 别去找，再由它的上下左右出发向上下左右接着去找，以此类推。因此，本题可 以使用广度优先算法，借助队列的先进先出这一特性实现此算法。 2.4.4 功能说明（函数、类） bool bfs(bool\* map, int row, int col, int loc) 功能：判断位置为 loc 的点是否形成了一个区域 Para map：矩阵数组 Para row：矩阵的行数 Para col：矩阵的列数 Para loc：该点的位置 返回值：位置为 loc 的点是否形成了一个区域 int solution(bool\* map, int row, int col) 功能：统计矩阵中区域的数量 Para map：矩阵数组 Para row：矩阵的行数 Para col：矩阵的列数 返回值：矩阵中区域的数量 2.4.5 调试分析（遇到的问题和解决方法） 在判断完一个点之后，需要设计一个标记，表示该点已经被判断过了。否则， 程序将陷入死循环之中。 2.4.6 总结和体会 广度优先算法的主要思想是先判断一个点，并让它的上下左右依次进入队列 中，判断完一个点之后，该点出队列。符合队列先进先出的特性。需要特别注意 的是，搜索该点时需要判断该点是否已经被搜索过，一个点不能被重复搜索。否 则，程序可能陷入死循环。 2.5 队列中的最大值 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。 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 本题有关队列的基本操作，因此采用队列这一数据结构。在求队列最大值时， 额外建立一个辅助队列，以空间复杂度换取时间复杂度，使程序能以 O（1）的 时间复杂度获取队列的最大值。 2.5.4 功能说明（函数、类） bool InitQueue(LinkQueue &q) 功能:构造一个空队列 Para q：构造的队列 返回值：队列建立是否成功 bool EnQueue(LinkQueue& q, int data) 功能：插入元素 data 为 q 的新的队尾元素 Para q：队列 Para data：插入的元素 返回值：插入是否成功 bool DeQueue(LinkQueue& q, int& data) 功能：若队列不空，删除 q 的队头元素，用 data 返回其值 Para q：队列 Para data：删除的元素的值 返回值：删除是否成功 void display(LinkQueue q) 功能：输出队列中的所有元素 Para q：队列 返回值：无返回值 2.5.5 调试分析（遇到的问题和解决方法） 一开始的算法采用遍历的方式寻找队列中的最大值，结果超时了。之后我建 立了一个辅助队列来存放队列中的最大值，队列的最大值始终存放在辅助队列的 队首，这样可以以 O（1）的时间复杂度获得最大值。 2.5.6 总结和体会 这道题主要考查了队列的基本操作和求队列中的最大值。其中，为了不让程 序超时，应该建立一个辅助队列来维护队列中的最大值，使其可以以 O（1）的 时间复杂度获得。 3.实验总结 栈和队列是两种重要的线性结构。从数据结构的角度看，栈和队列也是线性 表，其特殊性在于栈和队列的基本操作是线性表操作的子集，他们是操作受限的 线性表，可称为限定性的数据结构。但从数据类型角度看，他们是和线性表大不 相同的两类重要的抽象数据类型。