

**课 程 实 验 报 告**

2024 -2025 学年 第 一 学期

课程名称： 数据组织与管理

专业： 计算机科学与技术

班级： 03

姓名： 陈明浩

学号： 423109110333

信息工程学院

2024年 11 月

# **实验二 栈与队列定义和应用**

## 实验目的

1. 通过本次实习，重点加强对抽象数据类型Stack及Queue的认识与理解；
2. 实现栈在两种存储形式下的基本操作
3. 实现队在两种存储形式下的基本操作
4. 应用栈与队列结构求解实际问题，对问题分析建模，设计并实现ADT，最终实现问题的求解；
5. 在应用中体会算法的实现与存储结构的选择之间的关系

## 实验内容

1. 采用顺序存储结构表示抽象数据类型栈，并实现栈的初始化、入栈、出栈等各种操作。
2. 采用顺序存储结构表示抽象数据类型队列，并实现队的初始化、入队、出队等各种操作。

要求实现：

1. 实现对用户输入的十进制数转化成其他进制并输出
2. 计算用户输入的后缀表达式的值：例如52+3\*。输入的表达式中数字为一位整数即可。
3. 用队列实现计算和输出杨辉三角的前n行。

## 实验过程

## 3.1 分析问题……设计所用数据结构并用ADT表示，基本操作的函数声明

实验要求可知，本实验主要使用的数据结构是队列和栈。队列的抽象数据类型如下:

ADT Queue{ //所有代码字体：Consolas

数据对象：D={ai|ai∈ElemSet,i=1,2, ……,n,n>=0}

数据关系：R={<ai-1,ai>| ai-1,ai∈D,i=2,3, ……,n}

基本操作：

initQueue( &Q )

操作结果：构造一个空队列 Q。

isEmptyQueue ( Q )

初始条件：队列 Q 已存在。

操作结果：若 Q 为空队列，则返回 TRUE，否则返回 FALSE。

pushQueue( &Q, e )

初始条件：队列 Q 已存在。

操作结果：插入元素 e 为 Q 的新的队尾元素。

popQueue( &Q, &e )

初始条件：队列 Q 已存在且非空。

操作结果：删除 Q 的队头元素，并用 e 返回其值。

}ADT Queue;

栈的抽象数据类型如下:

ADT Stack{ //所有代码字体：Consolas

数据对象：D={ai|ai∈ElemSet,i=1,2, ……,n,n>=0}

数据关系：R={<ai-1,ai>| ai-1,ai∈D,i=2,3, ……,n}

基本操作:

void initStack(Stack &stk);

操作结果:初始化一个空栈

void pushStack(Stack &stk,ElementType val);

初始条件:存在一个栈

操作结果:将val元素放到栈顶

ElementType popStack(Stack &stk);

初始条件:存在一个不为空的栈

操作结果:将栈顶元素移除并且返回

ElementType peekStack(Stack stk);

初始条件:栈顶存在元素

操作结果:将栈顶元素返回

bool isEmptyStack(Stack stk);

操作结果:如果栈为空则返回true反之返回false

bool isFullStack(Stack stk);

操作结果:如果栈满则返回true反之返回false

//基本操作的函数声明:

bool isEmptyQueue(Queue queue);

bool isFullQueue(Queue queue);

void initQueue(Queue &queue);

void pushQueue(Queue &queue,ElementType val);

ElementType popQueue(Queue &queue);

ElementType getFront(Queue queue);

void initStack(Stack &stk);

void pushStack(Stack &stk,ElementType val);

ElementType popStack(Stack &stk);

ElementType peekStack(Stack stk);

bool isEmptyStack(Stack stk);

bool isFullStack(Stack stk);

## 3.2 详细设计

### 数据结构实现…………C 语言描述所用数据结构

队列的数据结构:

struct Queue {

int rear, front;

ElementType \*elem;

};

栈的数据结构:

struct Stack{

int top, end;

ElementType \*elem;

};

### 主要算法………伪代码描述

1. 实现进制转换

fn(int number,int n){

//number代表要进行转换的数字,n代表要转换的进制

if(!number) return ; //出口

fn(number / n,n); //递归调用

cout << number % n;

}

1. 计算后缀表达式的和

initStack(stk);

String str; //给出一个后缀表达式

for(遍历字符串){

if(元素为数字) pushStack(stk,e) //e代表元素

e1 = popStack(); //出两个要计算的元素

e2 = popStack();

else 判断符号{

“+” : e2 + e1;pushStack(e1 + e2);

“-” : e2 - e1;pushStack(e2 - e1);

“\*”:e2 \* e1;pushStack(e2 \* e1);

“/”:e2 / e1;pushStack(e2/e1);

}

cout << peekStack();

}

1. 杨辉三角

int rows; //表示要进行输出的行数

vector<vector<int>> res; //存放输出结果

initQueue(queue);

for(int i = 1; i <= rows; i ++){

vector<int> row; //用来存储一行的结果

for(int j = 0;j < i;j ++){ //用来计算每一行的元素

if(j为0或者i - 1说明到了一行中的最左或最右)row.push\_back(1);

else{

int left = getFront(queue);

popQueue(queue);

int right = getFront(queue);

pushQueue(queue,left + right);//维护队列以便生成下一行

row.push\_back(left + right);

}

}

res.push\_back(row);//保存每一行的结果

pushQueue(queue,1);//将该行的最后一个1进行补齐

}

最终res存放的就是n行杨辉三角的值

## 3.3 实验结果分析

1.由于每次都对number除n,所以空间复杂度和时间复杂度都为O(log number)

2.有一层for循环从1到n，时间复杂度为O(n),对于空间复杂对来说，最坏的情况是将n个数字都入栈所以空间复杂度也为O(n)

3.该程序有两层循环，一个控制行数，一个用来计算每行的数据，时间复杂度为O(n2)，空间复杂度，由于res进行对row的存储所以为O(n2)

感悟和心得:基本掌握了栈和队列数据结构，并且在实现队列时使用了环形数组这个概念，防止队列出现假溢出，理解了递归调用的过程本质为将一个大的问题转换为更小规模的问题。

## 评语和成绩

评语：

实验项目的背景、目的以及预期的功能需求阐述清晰。选择了适合的数据结构和算法，系统模块划分合理。算法描述准确，注释详细，图表准确，讲解较清楚。代码整洁规范、可读性强，且正确实现了设计文档中的功能需求。通过有效的注释和测试用例，展示了良好的编程习惯和测试覆盖率。

成绩：优

**说明：**

1. **报告要求认真排版，严格按模板给定的字体字号撰写，截图要做适当缩放，不要出现大片空白页；**
2. **必须认真填写心得体会，会成为重要评分依据之一**
3. **提交的报告要去掉所有批注、红色的提示语句及最后的说明！！！**