

**数据结构课程实践**

实习报告-魔王语言解释

2023

目录

[需求分析 2](#_Toc87012154)

[测试数据 2](#_Toc87012155)

[基本要求 2](#_Toc87012156)

[测试数据 2](#_Toc87012158)

[概要设计 2](#_Toc87012159)

[ADT定义 2](#_Toc87012160)

[主程序流程图 2](#_Toc87012161)

[模块层次与调用关系 2](#_Toc87012162)

[详细设计 2](#_Toc87012163)

[主程序代码 3](#_Toc87012165)

[函数调用关系图 3](#_Toc87012166)

[调试分析 3](#_Toc87012167)

[遇到的问题与解决 3](#_Toc87012168)

[复杂度分析 3](#_Toc87012169)

[经验与体会 3](#_Toc87012170)

[测试结果 3](#_Toc87012172)

# 需求分析

**1.问题描述**

有一个魔王总是使用自己的一种非常精练而抽象的语言讲话，没有人能昕得懂，但他的语言是可以逐步解释成人能听懂的语言，因为他的语言是由以下两种形式的规则由人的语言逐步抽象上去的：

(1)α->β1β2β3

(2)（θδ1δ2δ3）->θδ1θδ2θδ3θ

在这两种形式中，从左到右均表示解释。试写一个魔王语言的解释系统，把他的话解释成人能听得懂的话。

**2.基本要求**

用下述两条具体规则和上述规则形式(2)实现。设大写字母表示魔王语言的词汇；小写字母表示人的语言词汇；希腊字母表示可以用大写字母或小写字母代换的变量。魔王语言可含人的词汇。

(1)B->tAdA

(2)A->sae

**3. 测试数据**

B(ehnxgz)B解释成tsaedsaeezegexenehetsaedsae

# 概要设计

**1.ADT定义**

ADT Stack{

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,…,n}

基本操作：

void InitStack(Stack\* S)

操作结果：初始化栈S

bool Push(Stack\* S,char e)

操作结果：把字符e压入栈S

char Pop(Stack\* S)

操作结果：弹出栈顶元素，返回值为栈顶元素值

bool EmptyStack(Stack\* stk)

操作结果：判断栈是否为空

char GetTop(Stack\* stk)

操作结果：取栈顶元素并返回

void DestroyStack(Stack\* S)

操作结果：销毁栈

}ADT Stack

ADT Queue{

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,…,n}

基本操作：

void InitQueue(Queue\* Q)

操作结果：初始化队列

void Enqueue(Queue\* Q,char e)

操作结果：把字符e插入队列

char Dequeue(Queue\* Q)

操作结果：把队首元素出列并返回

void DestroyQueue(Queue\* Q)

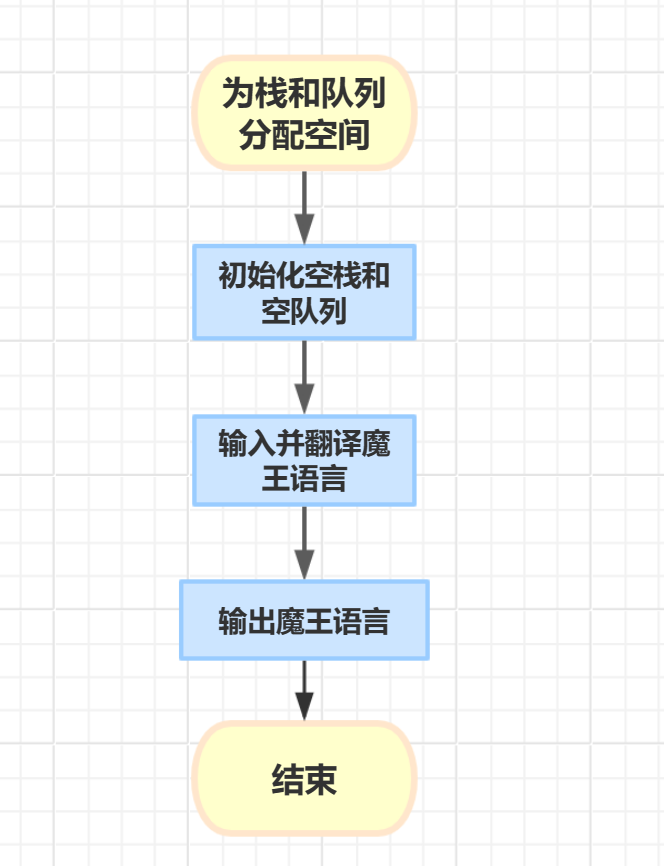
操作结果：销毁队列

void PrintQueue(Queue\* Q)

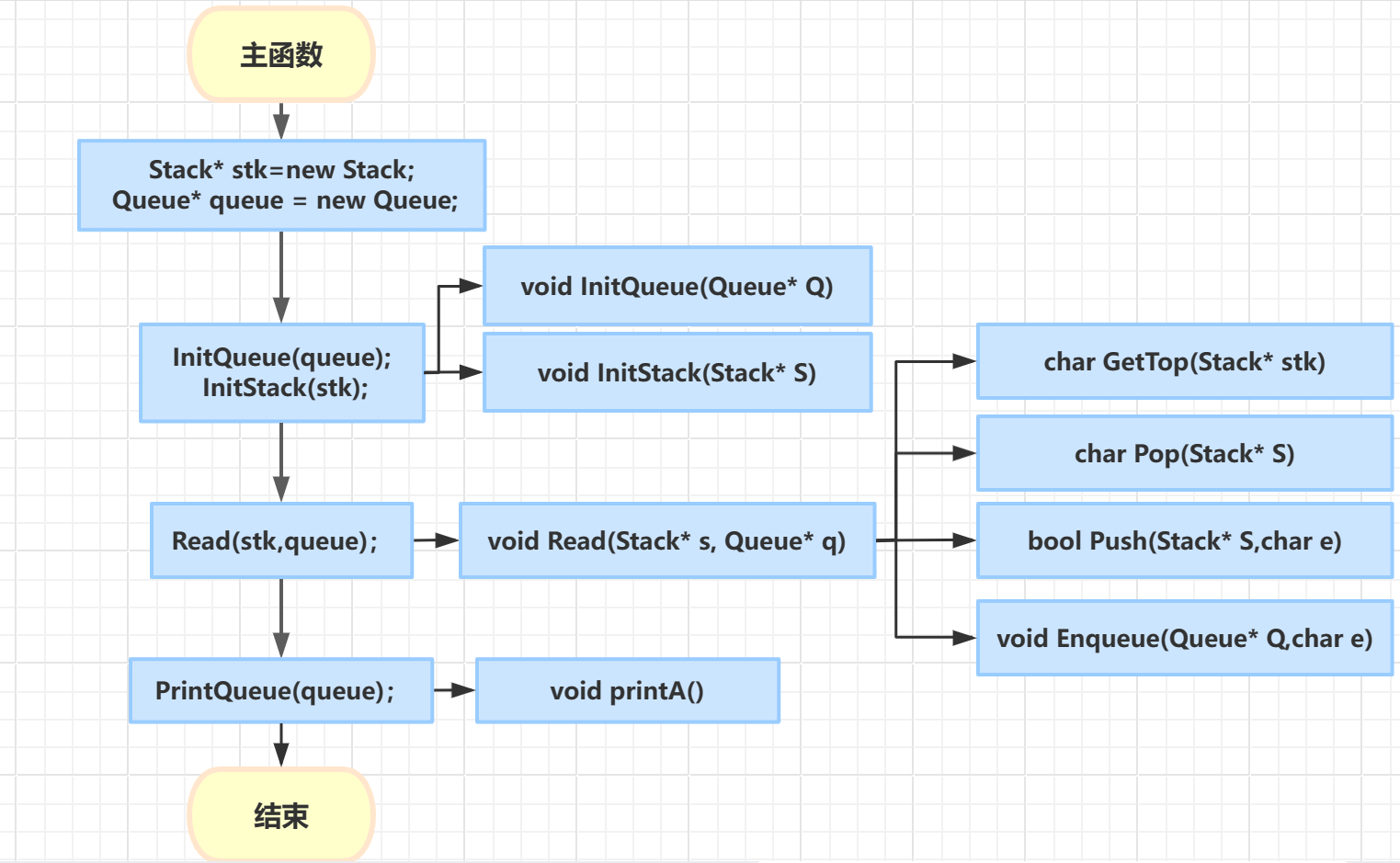
操作结果：打印队列

}ADT Queue

**2.主程序流程图**



**3.模块层次与调用关系**



# 详细设计

**1.主程序代码**

#include<iostream>

using namespace std;

typedef struct QueueNode {

char data;

QueueNode\* next;

};//队列结点

struct Queue {

QueueNode\* front;//队首指向链表第一个元素的前面

QueueNode\* back;//队尾指向链表最后一个元素

int length;//队列长度

};//队列结构体

typedef struct StackNode{

char data;

StackNode\* next;

};//栈结点

struct Stack {

StackNode\* top;//指向栈顶元素的顶部

int length;//栈的长度

};//栈结构体

void InitStack(Stack\* S) {

S->top = new StackNode;

S->top->next = NULL;

S->length = 0;

}//初始化栈

bool Push(Stack\* S,char e) {

StackNode\* node = new StackNode;

node->data = e;

node->next = S->top->next;

S->top->next = node;

S->length++;

return true;

}//入栈

char Pop(Stack\* S) {

StackNode\* node = S->top->next;

char e = node->data;

S->top->next = node->next;

delete node;

S->length--;

return e;

}//出栈

bool EmptyStack(Stack\* stk) {

if (stk->top->next == NULL) return 1;

else return 0;

}//判栈空

char GetTop(Stack\* stk) {

return stk->top->next->data;

}//取栈顶元素

void DestroyStack(Stack\* S) {

while (S->top->next != NULL) {

Pop(S);

}

return;

}//销毁栈

void InitQueue(Queue\* Q) {

Q->front = new QueueNode;

Q->back=Q->front;

Q->length = 0;

}//初始化队列

void Enqueue(Queue\* Q,char e) {

QueueNode\* node= new QueueNode;

node->data = e;

node->next = NULL;

Q->back->next = node;

Q->back = node;

Q->length++;

}//入列

char Dequeue(Queue\* Q) {

if (Q->front == Q->back) {

return -1;

}

QueueNode\* node = Q->front->next;

char e = node->data;

Q->front->next= node->next;

delete node;

Q->length--;

return e;

}//出列

void DestroyQueue(Queue\* Q) {

while (Q->front->next != NULL) {

Dequeue(Q);

}

}//销毁队列

void printA() {

cout << "sae";

}//翻译字符A

void PrintQueue(Queue\* Q) {

QueueNode\* P = Q->front->next;

while (P != NULL) {

if (P->data == 'A') {//翻译‘A’

printA();

}

else if (P->data == 'B') {//翻译‘B’

cout << 't';

printA();

cout << 'd';

printA();

}

else cout << P->data;//直接输出小写字母

P = P->next;

}

cout << endl;

}//打印队列

void Read(Stack\* s, Queue\* q) {

string str;

cin >> str;//输入字符串

int i = 0;

while (i < str.size()) {//按规则翻译字符串

if (str[i] == '(') {//有括号先处理括号内元素

Push(s, str[i]);//入左括号

char theta;

int flag = 0;

while (str[++i] != ')') {//遍历括号内元素

if (flag == 0) {//θ未记录，需要记录θ

theta = str[i];//记录θ

Push(s, str[i]);//小写字母直接入栈

flag = 1;

continue;

}

else {//θ记录好了，按规则入栈

Push(s, str[i]);

Push(s, theta);

}

}

while (GetTop(s) != '(') {//把栈内的元素移到队列内

Enqueue(q, GetTop(s));

Pop(s);

}

Pop(s);//弹出左括号

}

else Enqueue(q, str[i]);

i++;

}

}//读入魔王语言并翻译

int main() {

Stack\* stk=new Stack;

Queue\* queue = new Queue;

InitQueue(queue);//初始化队列

InitStack(stk);//初始化栈

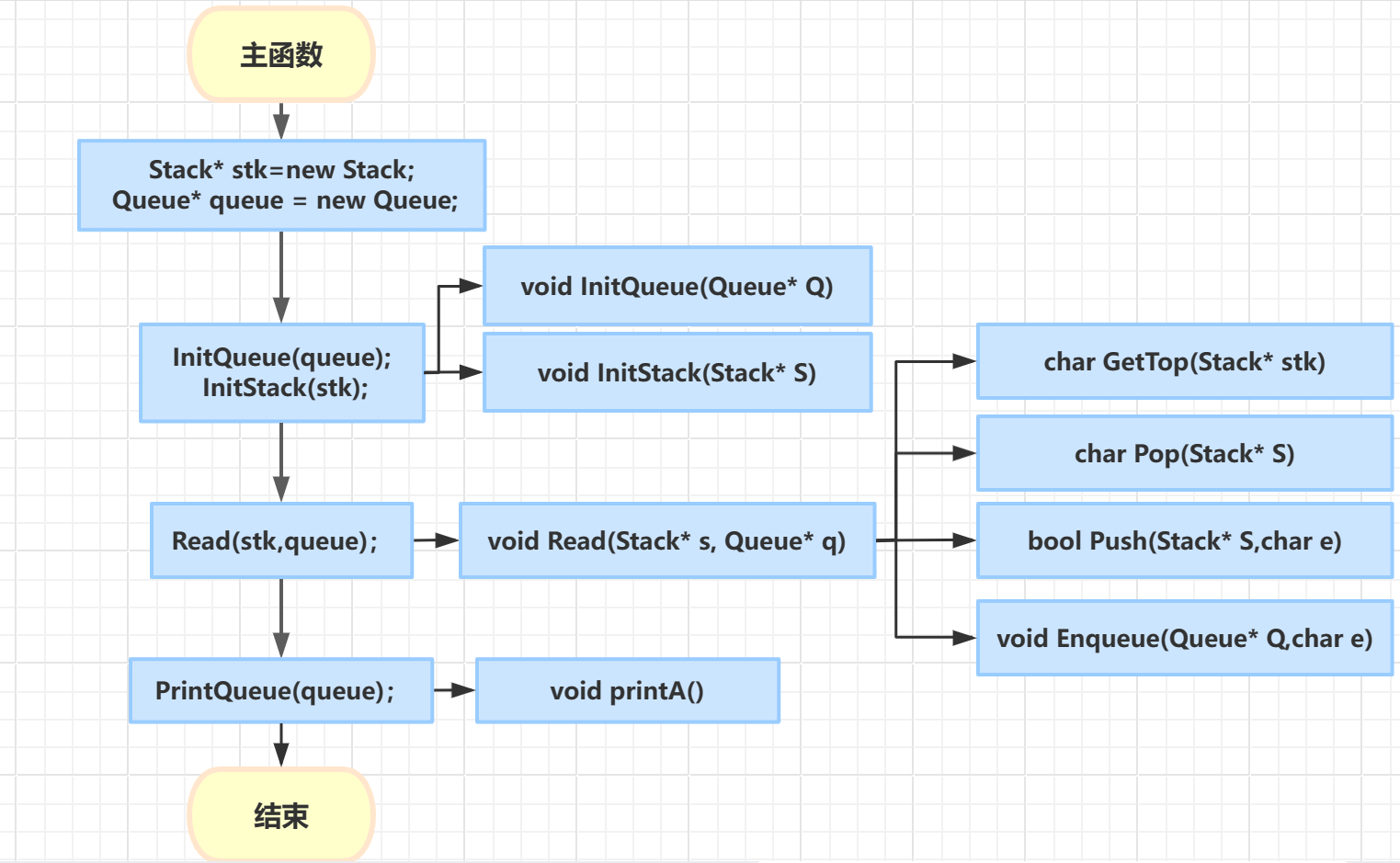
Read(stk,queue);//读取并翻译魔王语言

PrintQueue(queue);//输出魔王语言

return 0;

}

**2.函数调用关系图**



# 调试分析

**1.遇到的问题与解决**

（1）问题：把字符‘B’翻译时，直接输出”tsaedsae”违反了原来的规则

解决：先按规则把字符’B’翻译成“tAdA”，再通过printA()函数翻译字符’A’

（2）问题：在翻译规则(2)（θδ1δ2δ3）->θδ1θδ2θδ3θ时，刚开始不知道如何实现θ的插入

解决：先定义一个字符theta用来记录需要插入的θ，用flag标志是否已经把θ记录下来，后续用循环把θ插入即可，类似“输出空格但末尾不能有空格”的原理。

**2.复杂度分析**

时间复杂度：O(n)

解释：对输入的字符串进行一次遍历

空间复杂度：O(n)

解释：用队列存储所有字符

**3.经验与体会**

通过本次实验，我对栈和队列有更深的理解，能够熟练写出栈和队列的数据结构，引起了我对数据结构这门课程更多的兴趣。

# 测试结果

