数据结构第三次实验报告

(-)实验题目

1 试验3.1 实现顺序栈各种基本运算

2 试验3.2 实现链栈各种基本运算

3 试验3.4 实现队列各种基本运算

4 试验3.5 应用实例汉诺塔问题

(二)实验目的

1 熟练掌握栈的逻辑结构和操作规则，能在相应的实际问题中正确选用它

2 熟练掌握栈的两种实现方法(顺序栈和链式栈)，两种存储结构和基本运算的

实现算法，注意栈空和栈满的判断条件及他们的描述

3 熟练掌握队列的逻辑结构和操作规则，能在相应的实际问题中正确选用

4 掌握循环队列与链队列两种存储结构实现上的不同及解决办法，熟练掌握各

种队列基本运算的实现。

(三)实验要求

1)顺序栈的插入，删除，栈顶数据元素的读取。

2)链栈的插入，删除，栈顶数据元素的读取。

3)链队列的插入，删除。

(四)实验内容和实验步骤

**3.1实现顺序栈各种基本运算**

实验内容：

1 以顺序栈作为存储方式

2 实现顺序栈的入栈

3 实现顺序栈的出栈

4 实现顺序栈的读取

**代码：**

#include<stdio.h>

#include<stdlib.h>

#define Maxsize 20

#define ElemType int

typedef struct SqStack{

//定义结构体

ElemType elem[Maxsize]; //栈元素的存储空间

int top; //栈顶指针

}SqStack;

void Init\_SqStack(SqStack \*s){ //初始化栈

s->top = -1;

}

void Push(SqStack \*s,ElemType e){ //入栈

if(s->top < Maxsize-1){

s->top += 1;

s->elem[s->top] = e; //栈未满的时候，栈顶指针递增1，元素e入栈

}else{

printf("栈已满,不能入栈!\n");

}

}

ElemType Pop(SqStack\*s){

ElemType x;

if(s->top != -1){

x = s->elem[s->top];

s->top--;

return x; //栈未空时，栈顶指针递减1，元素x出栈

}else{

printf("栈已空，不能再出栈!\n");

return 0;

}

}

ElemType Get\_Top(SqStack \*s,ElemType x){

//读取栈顶元素

if(s->top != -1){

x = s->elem[s->top]; //栈不为空时，将栈顶元素存入变量x中

return x; //返回整数1，表示读取成功

}else{

printf("栈为空!\n");

return 0; //栈为空时，返回整数0，表示读取失败

}

}

void Display\_SqStack(SqStack \*s){

//顺序栈元素的显示

int m;

if(s->top == -1){

printf("顺序栈为空!\n");

}else{

for(m = s->top;m>-1;m--)

printf("%2d ",s->elem[m]);

printf("\n");

}

}

int main(){

SqStack s;

int y,cord;

ElemType a,x;

Init\_SqStack(&s);

Push(&s,1);

Push(&s,2);

Push(&s,3);

Push(&s,4);

printf("初始化\n依次进栈元素为1 2 3 4:\n");

printf("从栈顶到栈底元素为: \n");

Display\_SqStack(&s);

do{

printf(" 主菜单 \n");

printf(" 1 入栈 \n");

printf(" 2 出栈 \n");

printf(" 3 读栈顶元素 \n");

printf(" 4 结束程序 \n");

printf("请输入您选择的菜单号<1 2 3 4>: ");

scanf("%d",&cord);

switch(cord){

case 1:

printf("请输入入栈元素!");

scanf("%d",&a);

Push(&s,a);

printf("由栈顶到栈底元素为: ");

Display\_SqStack(&s);

break;

case 2:

x = Pop(&s);

printf("出栈元素为: %d\n",x);

printf("由栈顶到栈底元素为: ");

Display\_SqStack(&s);

break;

case 3:

if((y=Get\_Top(&s,x))!=0)

printf("栈顶元素为: %d\n",y);

printf("由栈顶到栈底元素为: ");

Display\_SqStack(&s);

break;

case 4:

exit(0);

break;

default:

printf("输入有误!\n");

}

}while(cord<=4);

return 0;

**实验3.2 实现链栈各种基本运算**

实验内容：

1 以链表作为存储结构

2 实现链栈上的插入运算

3 实现链栈上的删除

4 实现链栈元素的读取

**代码：**

**/\*实现链栈的各种基本操作\*/**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <malloc.h>**

**#include <iostream>**

**typedef int ElemType;**

**typedef struct Node{**

**ElemType data;**

**struct Node \* next;**

**}Node,\*LinkStack;**

**/\*函数功能：实现链栈的初始化\*/**

**void Init\_LinkStack(LinkStack &s){**

**s = (LinkStack)malloc(sizeof(Node));**

**s -> next = NULL; }**

**/\*函数功能：判断栈是否为空\*/**

**bool Empty(LinkStack &s){**

**if(s == NULL) return true;**

**else return false;}**

**/\*函数功能：实现元素进栈的操作，头插法进行操作\*/**

**void Push(LinkStack &s, ElemType x){**

**//生成一个新的节点**

**LinkStack p;**

**p = (LinkStack)malloc(sizeof(Node));**

**//对是否已经正确的生成节点进行判断**

**if(p != NULL){**

**p -> data = x;**

**p -> next = s; //将p作为头指针**

**s = p;**

**}else{**

**printf("入栈错误！\n");**

**}**

**}**

**/\*函数功能：实现出栈元素的操作\*/**

**int Pop(LinkStack &s,ElemType &x){**

**LinkStack p;**

**if(s == NULL){**

**printf("链栈为空，无法进行出栈的操作！\n");**

**return 0;**

**}else{**

**p = s;**

**s = p->next;**

**x = p->data;**

**free(p);**

**return 1;**

**}**

**}**

**/\*函数功能：访问链的栈顶元素\*/**

**int Get\_top(LinkStack &s,ElemType &x){**

**if(s == NULL){**

**printf("栈为空！\n");**

**return 0;**

**}else{**

**x = s->data;**

**return 1; }**

**}**

**/\*函数功能：从栈顶开始访问栈中的所有元素\*/**

**void Display\_SqStack(LinkStack &s){**

**LinkStack p;//访问用的指针**

**p = s;**

**printf("从栈顶到栈底的元素依次为：");**

**while(p->next!= NULL){**

**//此处为p->next != NULL的原因为：最开始的一个节点相当于没有用指针域**

**//指向的值为NULL，值域中不存放任何值**

**printf("%d ",p->data);**

**p = p->next;**

**}**

**printf("\n");**

**}**

**int main(){**

**ElemType e;**

**LinkStack s;//指向链栈的头**

**int cord,x,y;**

**Init\_LinkStack(s);**

**Push(s,1);**

**Push(s,2);**

**Push(s,3);**

**Push(s,4);**

**printf("初始化链栈的操作如下：\n");**

**Display\_SqStack(s);**

**do{**

**printf("\*\*\*\*\*\*\*\*\*\*主菜单\*\*\*\*\*\*\*\*\*\*\n");**

**printf("@1--------入栈-------------\n");**

**printf("@2--------出栈-------------\n");**

**printf("@3-----访问栈顶元素--------\n");**

**printf("@0--------结束程序---------\n");**

**printf("请输入您要执行的操作：");**

**scanf("%d",&cord);**

**switch(cord){**

**case 1:**

**printf("请输入您要进栈的元素：");**

**scanf("%d",&e);**

**printf("\n");**

**Push(s,e);**

**Display\_SqStack(s);**

**break;**

**case 2:**

**Pop(s,x);**

**printf("出栈的元素为%d\n",x);**

**Display\_SqStack(s);**

**break;**

**case 3:**

**Get\_top(s,y);**

**printf("栈顶元素为%d\n",y);**

**break;**

**case 0:**

**printf("程序运行结束！\n");**

**exit(0);**

**default:**

**printf("输入有错误!");**

**}**

**}while(cord <= 3);**

**return 0;**

**}**

**实验3.4 实现链队列各种基本运算**

实验内容:

1. 初始化链队列并插入初始数据节点
2. 实现入队运算
3. 实现链队列的出队运算

代码:

#include<iostream>

#include <cstdio>

#include <cstdlib>

#include <malloc.h>

using namespace std;

typedef int ElemType;

typedef struct Qnode

{ //队列链式存储结构定义

int data; //定义数据域

struct Qnode\* next; //定义指针域

}Qnode;

typedef struct

{

Qnode \*front; //队头指针

Qnode \*rear; //对尾指针

}LQueue;

void init\_LQueue(LQueue &q)

{//链队列的初始化

q.front = q.rear = (Qnode\*)malloc(sizeof(Qnode)); //生成空间

q.front->next = NULL; //初始化头尾指针

q.rear->next = NULL;

}

void en\_LQueue(LQueue &q,int e)

{//入队，将元素插入到队列q中，作为新的队尾元素

Qnode \*p;

p = (Qnode\*)malloc(sizeof(Qnode));

p->data = e;

p->next = NULL;

q.rear->next=p; //将链队列中对尾结点指针指向新结点

q.rear = p; //将对尾指针指向新结点

}

int de\_LQueue(LQueue &q,int x)

{//出队，若链队列不为空，则删除队头元素，并返回其值

Qnode \*p;

if (q.front == q.rear) {

return 0;

}

p = q.front->next; //取队头

x = p->data;

q.front->next = p->next; //删除队头元素

if (q.front->next == NULL) {

q.front = q.rear;

}

free(p);

return x;

}

void dispalay\_LQueue(LQueue &q)

{//队列元素的显示

Qnode \*p;

p = q.front->next;

while (p!= NULL) {

printf("%d ",p->data);

p = p->next;

}

printf("\n");

}

int main(int argc, const char \* argv[]) {

int x,y,cord,i,m;

LQueue q;

init\_LQueue(q);

printf("初始化\n");

for (i = 1; i<=4; i++) {

en\_LQueue(q, i);

}

dispalay\_LQueue(q);

do{

printf(" 主菜单 \n");

printf(" 1 入队 \n");

printf(" 2 出队 \n");

printf(" 3 结束程序 \n");

printf("-----------------------------\n");

printf("请输入您选择的菜单号 <1,2,3>: ");

scanf("%d",&cord);

switch (cord) {

case 1:

printf("请输入入队元素:");

scanf("%d",&x);

en\_LQueue(q, x);

printf("元素入列后的新队列如下:\n");

dispalay\_LQueue(q);

break;

case 2:

if((m = de\_LQueue(q, y)) == 0)

printf("队列为空\n");

else{

printf("出队元素为:%d\n",m);

printf("元素入列后的新队列如下:");

dispalay\_LQueue(q);

}

break;

case 3:

exit(0);

break;

default:

printf("输入有误!");

break;

}

}while(cord<=3 && cord>0);

return 0;

}

**实验3.5 应用实例**

实验内容：汉诺塔的移动

递归算法

代码:

#include<stdio.h>

static long m = 0;

void move(char A,char C)

{//从A移动到C

printf("\n移动到顶部盘子从%c to %c",A,C);

m++;

}

void hanoi(int n,char A,char B,char C)

{//把n个盘子从A移动到C,B作为辅助

if(n==1){

move(A,C); //将1号盘子移动到C

}

else{

hanoi(n-1,A,C,B); //将A上1号盘至n-1号盘子通过C辅助移动到B

move(A,C);

hanoi(n-1,B,A,C); //以A为辅助将1至n-1号盘子移至C

}

}

int main(){

int n;

printf("请输入盘子的个数:");

scanf("%d",&n);

printf("\n移动步骤如下:\n");

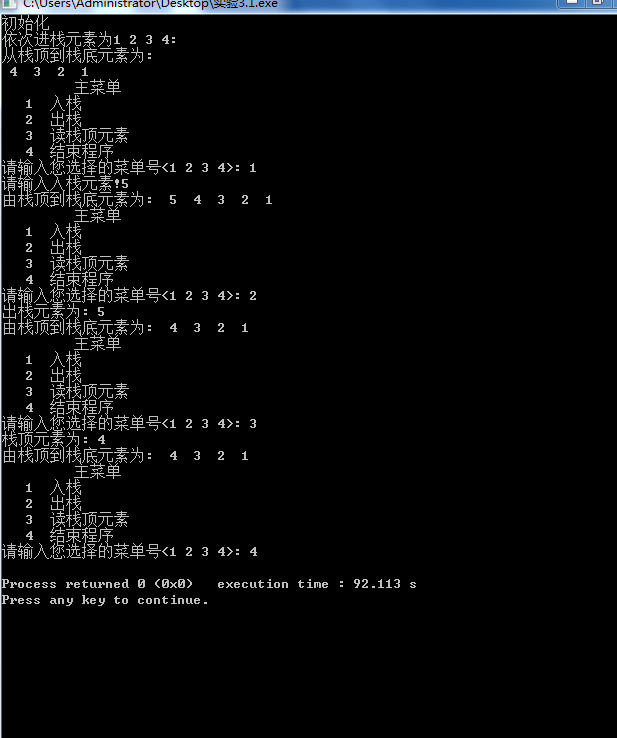
hanoi(n,'A','B','C');

printf("\n\n移动步骤总数为%d\n",m);

}

运行结果:

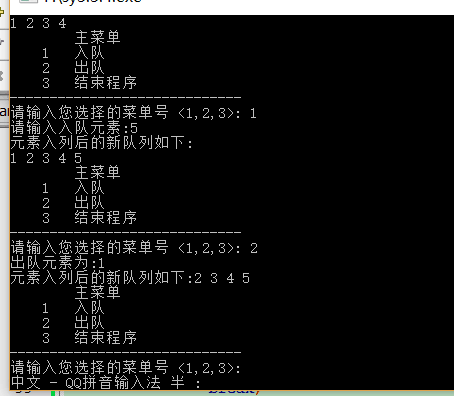
实验3.1



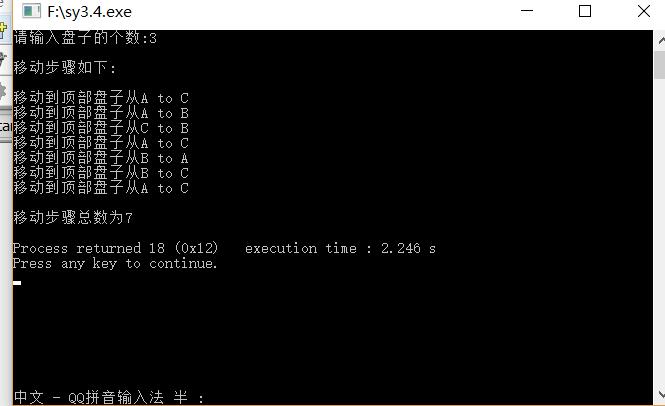
实验 3.2



实验 3.4



实验3.5



**(五) 实验分析**

**实验3.1**

**实验3.2**

**实验3.4**

**实验3.5**