### 实验2 栈与队列

|  |  |
| --- | --- |
| **成绩** |  |

**专业班级 信息173班 学 号 201711010228**

**姓 名 苏 栋 报告日期 2019.04.20**

**实验类型**：●验证性实验 ○综合性实验 ○设计性实验

1. **实验目的或任务**

通过指导学生上机实践，对常用数据结构的基本概念及其不同的实现方法的理论得到进一步的掌握，并对在不同存储结构上实现不同的运算方式和技巧有所体会。

1. **实验教学基本要求**

1.了解实验目的及实验原理；

2.编写程序，并附上程序代码和结果图；

3.总结在编程过程中遇到的问题、解决办法和收获。

1. **实验教学的内容或要求**

1.编写函数,采用链式存储实现栈的初始化、入栈、出栈操作

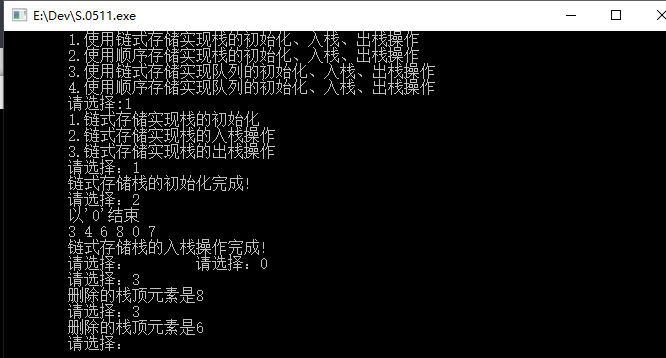
2.编写函数,采用顺序存储实现栈的初始化、入栈、出栈操作

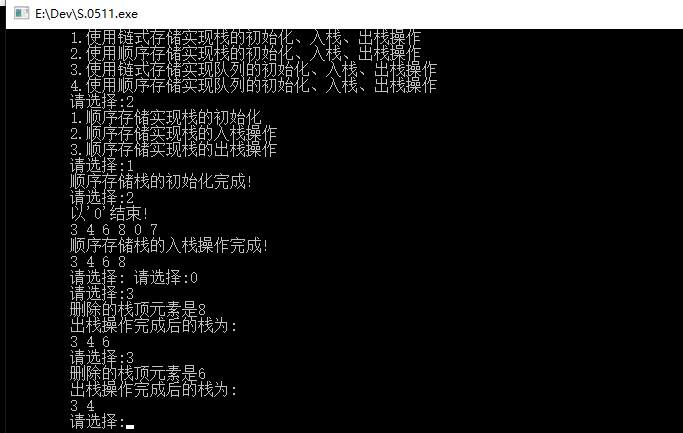
3.编写函数,采用链式存储实现队列的初始化、入队、出队操作

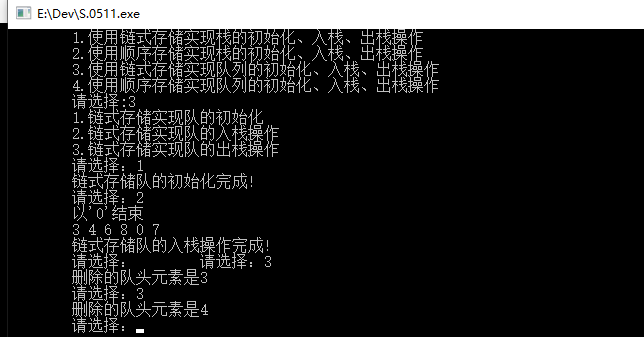
4.编写函数,采用顺序存储实现队列的初始化、入队、出队操作

5.编写一个主函数,在主函数中设计一个简单的菜单,分别调试上述算法

**四、实验任务的程序运行运行界面及运行结果**









**五、实验代码**

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include <stdio.h>

#include <stdlib.h>

#define OVERFLOW -2

#define ERROR 0

#define OK 1

#define MAX 100 //栈的最大值

typedef int SElemType;

typedef int QElemType;

typedef struct

{

SElemType \*base;

SElemType \*top;

}SqStack;

SqStack InitStacka( ) //顺序存储实现栈的初始化

{

SqStack S;

S.base = ( SElemType \* )malloc( MAX \*sizeof( SElemType ));

if( !S.base )

exit( OVERFLOW );

S.top = S.base;

return(S);

}

void Pusha(SqStack &S, int x) //顺序存储实现栈的入栈操作

{

if(S.top - S.base >= MAX)

{

exit(OVERFLOW);

}

\*S.top++ = x;

}

void Popa(SqStack &S) //顺序存储实现栈的出栈操作

{

SElemType \*p;

int x;

if(S.top == S.base)

{

return ;

}

else

{

p = S.top;

x = \* --S.top;

printf("\t删除的栈顶元素是%d\n\t出栈操作完成后的栈为:\n", x);

}

}

void printa(SqStack S) //输出

{

SElemType \*p;

p = S.base;

printf("\t");

while(p != S.top)

{

printf("%d ", \*(p++));

}

printf("\n");

}

typedef struct SqNode

{

SElemType data;

SqNode \*Link;

}\*Sqptr,NODE;

typedef struct

{

Sqptr top;

}Stack;

Stack InitStackb() //链式存储实现栈的初始化

{

Stack S;

S.top = ( Sqptr )malloc( sizeof( NODE ) );

if( !S.top )

{

exit (OVERFLOW);

}

S.top -> Link = NULL;

return(S);

}

void Pushb(Stack &S, int x) //链式存储实现栈的入栈操作

{

Sqptr p;

p = ( Sqptr )malloc( sizeof( NODE ) );

if( !p )

{

return ;

}

p -> data = x;

p -> Link = S.top -> Link;

S.top -> Link = p;

}

void Popb(Stack &S) //链式存储实现栈的出栈操作

{

int x;

Sqptr p;

if(S.top -> Link == NULL)

{

return ;

}

else

{

p = S.top -> Link;

x = p -> data;

S.top -> Link = p -> Link;

printf("\t删除的栈顶元素是%d\n", x);

free(p);

}

}

typedef struct QNode

{

QElemType data;

struct QNode \*next;

}\*QueuePtr,QNode;

typedef struct

{

QueuePtr front;

QueuePtr rear;

}LinkQueue;

LinkQueue InitQueue() //链式存储实现队列的初始化

{

LinkQueue Q;

Q.front = Q.rear = ( QueuePtr )malloc( sizeof( QNode ) );

if( !Q.front )

{

exit (OVERFLOW);

}

Q.front -> next = NULL;

return(Q);

}

void EnQueue(LinkQueue &Q, QElemType x) //链式存储实现队列的入队操作

{

QueuePtr p;

p = ( QueuePtr )malloc( sizeof( QNode ) );

if( !p )

{

exit(OVERFLOW);

}

p -> data = x;

p -> next = NULL;

Q.rear -> next = p;

Q.rear = p;

}

void DeQueue(LinkQueue &Q) //链式存储实现队列的出队操作

{

int x;

if(Q.front == Q.rear)

{

return ;

}

QueuePtr p;

p = Q.front -> next;

x = p -> data;

printf("\t删除的队头元素是%d\n", x);

Q.front -> next = p -> next;

if(Q.rear == p)

{

Q.rear = Q.front;

}

free(p);

return;

}

typedef struct

{

SElemType \*base;

int front, rear;

}SqQueue;

SqQueue InitQueueb() //顺序存储实现队列的初始化

{

SqQueue S;

S.base = ( SElemType \* )malloc( MAX \*sizeof( SElemType ));

if( !S.base )

{

exit(OVERFLOW);

}

S.front = S.rear = 0;

return(S);

}

void EnQueueb(SqQueue &S, int x) //顺序存储实现队列的入队操作

{

if((S.rear+1) % MAX == S.front)

{

return ;

}

S.base[S.rear] = x;

S.rear = (S.rear+1) % MAX;

}

void DeQueueb(SqQueue &S) //顺序存储实现队列的出队操作

{

int x;

if(S.front == S.rear)

{

return ;

}

x = S.base[S.front];

S.front = (S.front+1) % MAX;

printf("\t删除的队头元素是:%d\n", x);

}

int main()

{

int choice;

int n, x;

printf("\t1.使用链式存储实现栈的初始化、入栈、出栈操作\n");

printf("\t2.使用顺序存储实现栈的初始化、入栈、出栈操作\n");

printf("\t3.使用链式存储实现队列的初始化、入栈、出栈操作\n");

printf("\t4.使用顺序存储实现队列的初始化、入栈、出栈操作\n");

printf("\t请选择:");

scanf("%d", &choice);

switch(choice)

{

case 1:

Stack Sa;

printf("\t1.链式存储实现栈的初始化\n");

printf("\t2.链式存储实现栈的入栈操作\n");

printf("\t3.链式存储实现栈的出栈操作\n");

while(1)

{

printf("\t请选择：");

scanf("%d", &n);

switch(n)

{

case 1:

Sa = InitStackb();

printf("\t链式存储栈的初始化完成!\n");

break;

case 2:

printf("\t以'0'结束\n");

printf("\t");

scanf("%d", &x);

while(x)

{

Pushb(Sa, x);

scanf("%d", &x);

}

printf("\t链式存储栈的入栈操作完成!\n");

break;

case 3:

Popb(Sa);

break;

}

}

break;

case 2:

SqStack S;

printf("\t1.顺序存储实现栈的初始化\n");

printf("\t2.顺序存储实现栈的入栈操作\n");

printf("\t3.顺序存储实现栈的出栈操作\n");

while(1)

{

printf("\t请选择:");

scanf("%d", &n);

switch(n)

{

case 1:

S = InitStacka();

printf("\t顺序存储栈的初始化完成!\n");

break;

case 2:

printf("\t以'0'结束!\n");

printf("\t");

scanf("%d", &x);

while(x)

{

Pusha(S, x);

scanf("%d", &x);

}

printf("\t顺序存储栈的入栈操作完成!\n");

printa(S);

break;

case 3:

Popa(S);

printa(S);

break;

}

}

break;

case 3:

LinkQueue Q;

printf("\t1.链式存储实现队的初始化\n");

printf("\t2.链式存储实现队的入栈操作\n");

printf("\t3.链式存储实现队的出栈操作\n");

while(1)

{

printf("\t请选择：");

scanf("%d", &n);

switch(n)

{

case 1:

Q = InitQueue();

printf("\t链式存储队的初始化完成!\n");

break;

case 2:

printf("\t以'0'结束\n");

printf("\t");

scanf("%d", &x);

while(x)

{

EnQueue(Q, x);

scanf("%d", &x);

}

printf("\t链式存储队的入栈操作完成!\n");

break;

case 3:

DeQueue(Q);

break;

}

}

break;

case 4:

SqQueue Sv;

printf("\t1.顺序存储实现队的初始化\n");

printf("\t2.顺序存储实现队的入栈操作\n");

printf("\t3.顺序存储实现队的出栈操作\n");

while(1)

{

printf("\t请选择：");

scanf("%d", &n);

switch(n)

{

case 1:

Sv = InitQueueb();

printf("\t链式存储栈的初始化完成!\n");

break;

case 2:

printf("\t以'0'结束\n");

printf("\t");

scanf("%d", &x);

while(x)

{

EnQueueb(Sv, x);

scanf("%d", &x);

}

printf("\t链式存储栈的入栈操作完成!\n");

break;

case 3:

DeQueueb(Sv);

break;

}

}

break;

}

system("pause");

return 0;

}

**六、实验总结**

通过本次实验，我掌握了链式存储、顺序存储栈与队列的初始化、入栈、出栈相关知识，明白了如何初始化、进栈、出栈或初始化队列、进队、出队等操作，懂得编写链式存储和顺序存储实现栈以及队列的相关内容。熟悉了结构体的用法，熟悉了嵌套的使用，熟悉了函数的调用和实参形参的使用。在今后的学习中，我会加强对结构体指针以及栈、队列的相关用法。