实现栈和队列的各种基本运算

姓名：凌强 班级：21计算机二班 学号：2021413579

实验目的：

1. 掌握栈和队列的基本运算。
2. 掌握栈和队列的概念，学会对栈和队列数据结构进行操作。
3. 加深对栈和队列数据结构的理解，逐步培养解决实际问题的编程能力。

实验内容：

1. 初始化栈和队列，接收输入的数据，以栈和队列两种方式存储存储。
2. 通过输入的数字序号选择对应的功能以完成栈和队列的入栈、出栈、入队、出队等操作。
3. 采用模块测试化方式进行实验。

# **需求分析**

1. 完成对栈和队列的创建、销毁、入栈、出栈、入队、出队、输出等基本操作。
2. 演示程序中，规定 ElemType 为 int 型。
3. 演示程序以用户和计算机的对话方式执行，在运行框中出现提示信息之后，由用户在键盘上输入完成相关命令的执行；相应结果显示在其后。
4. 程序执行的命令包括：
   1. 输出用户界面。
   2. 读取用户输入的数据，通过栈和队列两种方式创建。
   3. 输入数字选择对应的功能。
   4. 将一个元素入队。
   5. 将一个元素入栈。
   6. 将一个元素出栈。
   7. 将一个元素出队。
   8. 销毁栈和队列。
   9. 退出程序。
5. 测试数据：见（[六、测试结果](#_测试结果)）。

# **概要设计**

1. 栈和队列的抽象数据定义：

typedef struct Queue

{

    int data[MaxSize];

    int front, rear;

} SqQueue;

typedef struct Stack

{

    int data[MaxSize];

    int top;

} SqStack;

1. 基本操作：

*//初始化队列*

void InitQueue(SqQueue \*&*q*)

*//入队*

bool InQueue(SqQueue \*&*q*,int *e*)

*//初始化栈*

void InitStack(SqStack \*&*s*)

*//入栈*

bool Push(SqStack \*&*s*, int *e*)

*//初始化队列*

void InitQ(SqQueue \**Q*)

*//初始化栈*

void InitS(SqStack \**S*)

*//入列*

void InQ(SqQueue \*&*q*)

*//入栈*

void InS(SqStack \*&*s*)

*//出队*

void OutQ(SqQueue \*&*q*)

*//出栈*

void OutS(SqStack \*&*s*)

*//输出主界面*

void welcome(SqQueue \*&*Q*,SqStack \*&*S*)

1. 程序包含两个模块：
2. 主函数模块：

用于与测试用户交换数据。

1. 操作函数模块：

用于指引测试用户下一步操作，操作栈和队列。

# **详细设计**

## 栈和队列的定义

*#include* <iostream>

*#define* MaxSize 200

using namespace std;

typedef struct Queue

{

    int data[MaxSize];

    int front, rear;

} SqQueue;

typedef struct Stack

{

    int data[MaxSize];

    int top;

} SqStack;

## 操作函数

void InitQueue(SqQueue \*&*q*){

*q* = (SqQueue \*)malloc(sizeof(SqQueue));

*q*->front = *q*->rear = 0;

}

bool InQueue(SqQueue \*&*q*,int *e*){

*if*((*q*->rear+1) % MaxSize == *q*->front)

*return* false;

*q*->rear = (*q*->rear+1) % MaxSize;

*q*->data[*q*->rear] = *e*;

*return* true;

}

void InitStack(SqStack \*&*s*)

{

*s* = (SqStack \*)malloc(sizeof(SqStack));

*s*->top = -1;

}

bool Push(SqStack \*&*s*, int *e*)

{

*if*(*s*->top == MaxSize-1)

*return* false;

*s*->top++;

*s*->data[*s*->top] = *e*;

*return* true;

}

*//初始化队列*

void InitQ(SqQueue \**Q*){

    int k;

    cout<<"请输入要测试的数据(队列):"<<endl;

*while*(cin>>k)

    {

        InQueue(*Q*,k);

*if*(cin.get() == '\n') *break*;

    }

    cout<<endl<<"初始化队列成功！"<<endl;

}

*//初始化栈*

void InitS(SqStack \**S*){

    int k;

    cout<<"请输入要测试的数据(栈):"<<endl;

*while*(cin>>k)

    {

        Push(*S*,k);

*if*(cin.get() == '\n') *break*;

    }

    cout<<endl<<"初始化栈成功！"<<endl;

}

*//入列*

void InQ(SqQueue \*&*q*)

{

    int n;

    cout<<"请输入要入队的元素:";

    cin>>n;

*if*((*q*->rear+1) % MaxSize == *q*->front)

    {

        cout<<"error"<<endl;

*return*;

    }

*q*->rear = (*q*->rear+1) % MaxSize;

*q*->data[*q*->rear] = n;

}

*//入栈*

void InS(SqStack \*&*s*)

{

    int n;

    cout<<"请输入要入栈的元素:";

    cin>>n;

*if*(*s*->top == MaxSize-1)

    {

        cout<<"error"<<endl;

*return*;

    }

*s*->top++;

*s*->data[*s*->top] = n;

}

*//出队*

void OutQ(SqQueue \*&*q*)

{

*if*(*q*->front == *q*->rear){

        cout<<"error"<<endl;

*return*;

    }

*q*->front = (*q*->front+1) % MaxSize;

    cout<<"当前出队元素为："<<*q*->data[*q*->front]<<endl;

}

*//出栈*

void OutS(SqStack \*&*s*)

{

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

        cout<<"error"<<endl;

*return*;

    }

    cout<<"当前出栈元素为："<<*s*->data[*s*->top--]<<endl;

}

void Gettop(SqStack \*&*s*){

    cout<<"当前栈顶元素为："<<*s*->data[*s*->top]<<endl;

}

*//输出主界面*

void welcome(SqQueue \*&*Q*,SqStack \*&*S*)

{

    system("cls");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓  ☆      栈和队列测试系统      ☆  〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓★★★★★         ★★★★★★★         ★★★★★〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓★★★★★         ★★★★★★★         ★★★★★〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆            1.出队          ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆            2.出栈          ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆            3.入队          ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆            4.入栈          ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆            5.栈顶          ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          0.退出系统        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆                            ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    InitQ(*Q*);

    InitS(*S*);

}

## 主函数完善

int main()

{

    SqQueue \*Q;

    SqStack \*S;

    int choice;

    InitStack(S);

    InitQueue(Q);

    welcome(Q,S);

*while* (1)

    {

        cout<<"请输入操作的序号:";

        cin>>choice;

*switch* (choice)

        {

*case* 1:

            OutQ(Q);

*continue*;

*case* 2:

            OutS(S);

*continue*;

*case* 3:

            InQ(Q);

*continue*;

*case* 4:

            InS(S);

*continue*;

*case* 5:

            Gettop(S);

*continue*;

*case* 0:

*break*;

*default*:

*continue*;

        }

    }

    system("pause");

*return* 0;

}

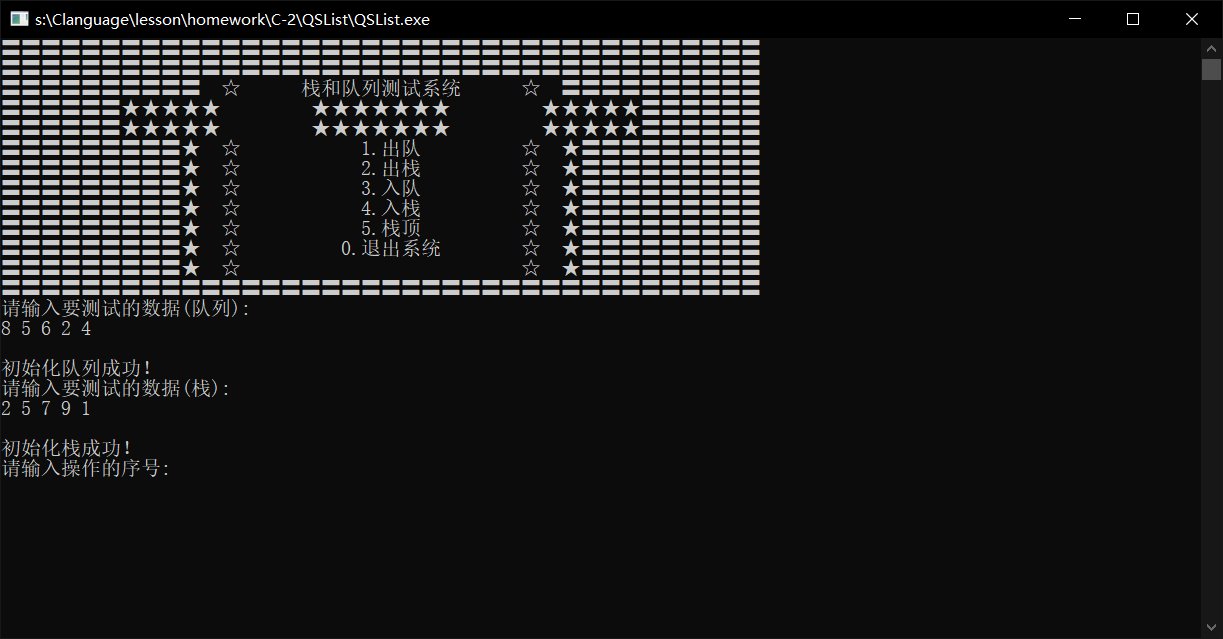
# **调试分析**

1. 模块划分相对合理。
2. 本次实验栈采用数组的方式，队列采用环形队列数组的方式。
3. 栈的基本操作较为简单，大多操作函数能一两行写完，而队列的基本操作相对复杂一点，特别是本次实验用的环形队列，在判断队空或队满时容易出错
4. 本次实验报告完成了对栈和队列的各种基本运算，是一次良好的程序设计训练。

# **用户使用说明**

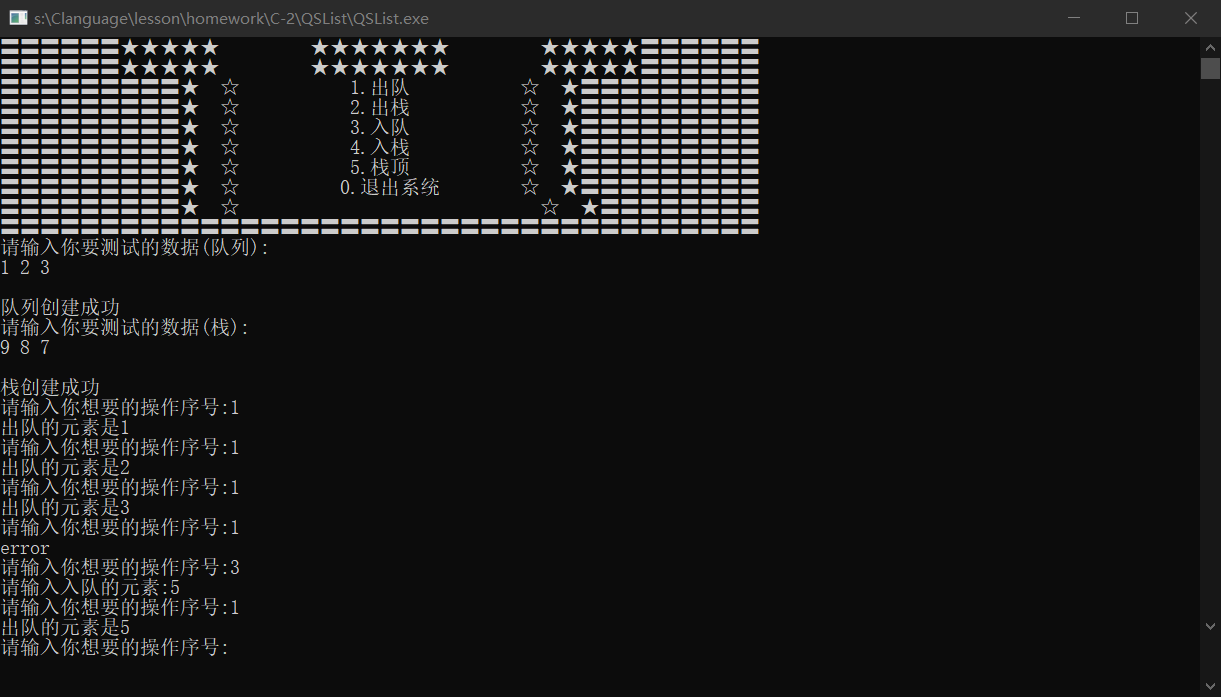
程序运行环境为vscode,执行文件为QSList.cpp

用户界面：先输入元素个数，然后依次输入数据，最后创建成功，选择对应的功能。

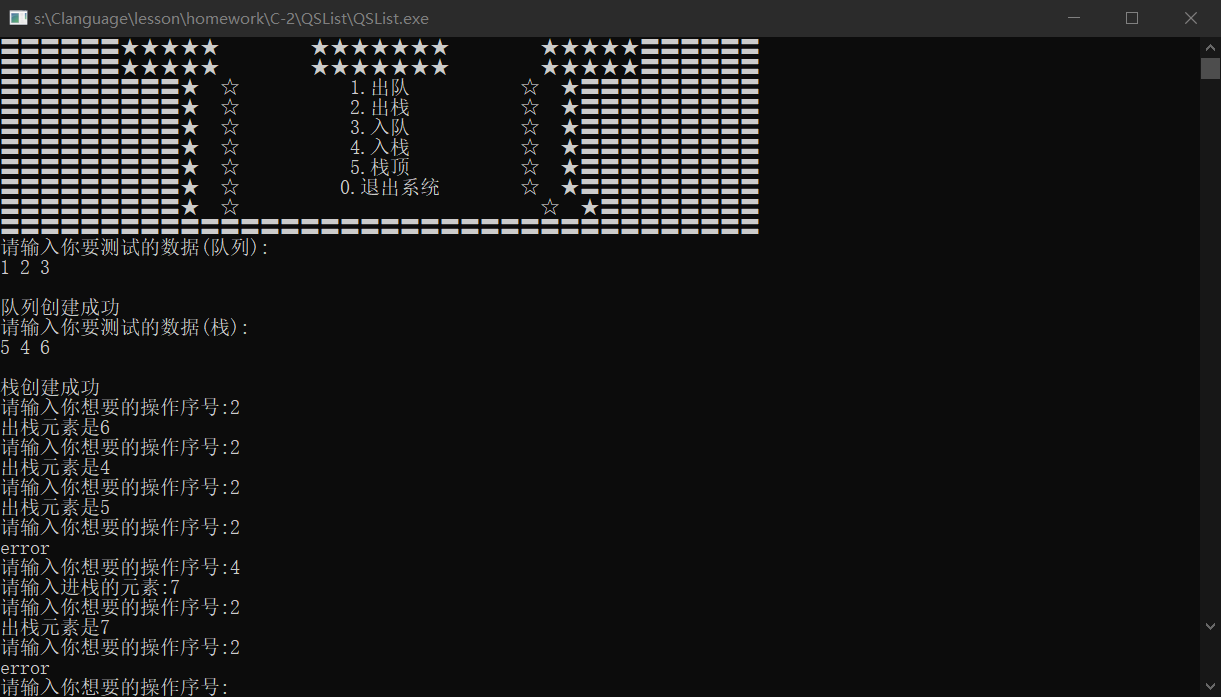


# **测试结果**

## 队列



## 栈



# **实验体会**

通过这次实验的测试，学习了栈和队列的定义和各种基本操作，并且了解到了各自的优缺点和其对应的应用领域，大致总结一下吧：

1. 相同点：
2. 都是线性结构，即数据元素之间具有“一对一”的逻辑关系
3. 都可以在顺序存储结构和链式存储结构上进行实现
4. 在时间代价上，插入和删除操作都需常数时间；在空间代价上，情况相同
5. 不同点：
6. 删除数据元素操作的位置不同。对于栈而言，其在一端插入数据元素，同时在该端删除数据元素，对于队列而言，其在一端插入数据元素，在另一端删除数据元素。
7. 两者的应用场景不同。具有后进先出(或先进后出)特性的应用需求，可以使用栈来进行管理。对于具有先进先出(或后进后出)特性的应用需求，可以使用队列进行管理

# **附录**

源程序：

QSList.cpp