華中科技大學

课 程 实 验 报 告

课程名称：C++程序设计

实验名称：面向对象的整型栈编程

院　　系 ：计算机科学与技术

专业班级 ：　计科 2011 班

学　　号 ：　U202015084

姓　　名 ：　张文浩

指导教师 ：　金良海

2021 年 12 月 15 日

一、需求分析

1. 题目要求

整型栈是一种先进后出的存储结构，对其进行的操作通常包括：向栈顶压入一个整型元素、从栈顶弹出一个整型元素等。整型栈类 STACK 采用之前定义的两个 QUEUE 类模拟一个栈，其操作函数采用面向对象的 C++语言定义，请将完成上述操作的所有如下函数采用 C++语言编程， 然后写一个 main 函数对栈的所有操作函数进行测试，请不要自己添加定义任何新的函数成员和数据成员。

```
class STACK : public QUEUE {
    QUEUE q;
public:
    STACK(int m);                              //初始化栈：最多存放 2m-2 个元素
    STACK(const STACK& s);                     //用栈 s 深拷贝初始化栈
    STACK(STACK&& s)noexcept;                 //用栈 s 移动拷贝初始化栈
    int  size()const noexcept;                 //返回栈的容量即 2m
    operator int() const noexcept;             //返回栈的实际元素个数
    STACK& operator<<(int e);                  //将 e 入栈，并返回当前栈
    STACK& operator>>(int& e);                 //出栈到 e，并返回当前栈
    STACK& operator=(const STACK& s); //深拷贝赋值并返回被赋值栈
    STACK& operator=(STACK&& s)noexcept;//移动赋值并返回被赋值栈
    char * print(char *b)const noexcept; //从栈底到栈顶打印栈元素
    ~STACK()noexcept;                          //销毁栈
};
```

编程时应采用 VS2019 开发，并将其编译模式设置为 X86 模式，其他需要注意的事项说明如下：

（1）在用 STACK(int m)对栈初始化时， 为其基类和成员 q 的 elems 分配 m 个整型元素内存，并初始化基类和成员 q 的 max 为 m，以及初始化对应的 head=tail=0。

（2）对于 STACK(const STACK& s)深拷贝构造函数，在用已经存在的对象 s 深拷贝构造新对象时，新对象不能共用 s 的基类和成员 q 为 elems 分配的内存，新对象要为其基类和成员 q 的 elems 分配和 s 为其基类和成员 q 的 elems 分配的同样大小的内存，并且将 s 相应的 elems 的内容深拷贝至新对象为对应 elems 分配的内存;新对象应设置其基类和成员 q 的 max、head、tail 和 s 的对应值相同。

（3）对于 STACK(STACK&& s)noexcept 移动拷贝构造函数，在用已经存在的对象 s 移动构造新对象时，新对象接受使用 s 为其基类和成员 q 的对应 elems 分配的内存，并且新对象的 max、head、tail 应和 s 的基类和成员 q 的对应值相同；s 的基类和成员 q 的 elems 设置为空指针以表示内存被移走，同时其对应的 max、head、tail 均应置为 0。

（4）对于 STACK& operator=(const STACK& s)深拷贝赋值函数，在用等号右边的对象 s 深拷贝赋值等号左边的对象 s 时，等号左边对象的基类和成员 q 不能共用 s 的基类和成员 q 为 elems 分配的内存，若等号左边的对象为其基类和成员 q 的 elems 分配了内存，则应先释放掉以避免内存泄漏，然后为其 elems 分配和 s 为其基类和成员 q 的 elems 分配的同样大小的内存，并且将 s 对应两个 elems 的内容拷贝至等号左边对象对应两个 elems 的内存；等号左边对象中的 max、head、tail 应设置成和 s 中基类和成员 q 的对应值相同。

（5）对于 STACK& operator=(STACK&& s)noexcept 移动赋值，在用等号右边的对象 s 移动赋值给等号左边的对象时，等号左边的对象如果已经为其基类和成员 q 中的 elems 分配了内存，则应先释放以避免内存泄漏，然后接受使用 s 的基类和成员 q 为 elems 分配的内存，并且等号左边对象中的 max、head、tail 应和 s 中基类和成员 q 中的对应值相同；s 中基类和成员 q 的 elems 设置为空指针以表示内存被移走，同时其对应的 max、head、tail 均应设置为 0。

（6）栈空弹出元素或栈满压入元素均应抛出异常，并且保持其内部状态不变。

（7）打印栈时从栈底打印到栈顶，打印的元素之间以逗号分隔。

2. 需求分析

本次实验要求用 C++实现对栈的基本操作，栈相关信息用类储存，整型栈类 STACK 采用之前定义的两个 QUEUE 类模拟一个栈，作为之前实现的队列类的派生类。基本操作包括初始化栈（用最大元素个数初始化、用深拷贝方法初始化、用移动构造方法初始化）、获取栈的实际元素个数、获取栈的容量、入栈、出栈、用深拷贝和移动构造方法实现赋值、打印栈中的元素、销毁栈等功能。

二、系统设计

1. 概要设计

本次实验用类来表示栈，作为之前实现的队列类的派生类。类的成员变量包括一个栈类的对象。所实现的对栈的操作包括初始化栈（用最大元素个数初始化、用深拷贝方法初始化、用移动构造方法初始化）、获取栈的实际元素个数、获取栈的容量、入栈、出栈、用深拷贝和移动构造方法实现赋值、打印栈中的元素、销毁栈等。

本次实验采用专用测试库进行功能测试，没有人机交互界面。

所实现的函数功能模块主要包括：

```
STACK(int m);                          //初始化栈：最多存放 2m-2 个元素
STACK(const STACK& s);                 //用栈 s 深拷贝初始化栈
STACK(STACK&& s)noexcept;            //用栈 s 移动拷贝初始化栈
int  size()const noexcept;             //返回栈的容量即 2m
operator int() const noexcept;         //返回栈的实际元素个数
STACK& operator<<(int e);              //将 e 入栈，并返回当前栈
```

```
STACK& operator>>(int& e);              //出栈到 e，并返回当前栈
STACK& operator=(const STACK& s); //深拷贝赋值并返回被赋值栈
STACK& operator=(STACK&& s)noexcept;//移动赋值并返回被赋值栈
char * print(char *b)const noexcept;  //从栈底到栈顶打印栈元素
~STACK()noexcept;                       //销毁栈
```

总体流程图如下：

图 2-1　总体流程图

2. 详细设计

①栈类：

```
class STACK : public QUEUE {
    QUEUE q;
public:
    STACK(int m);                          //初始化栈：最多存放 2m-2 个元素
    STACK(const STACK& s);                 //用栈 s 深拷贝初始化栈
    STACK(STACK&& s)noexcept;          //用栈 s 移动拷贝初始化栈
    int size()const noexcept;              //返回栈的容量即 2m
```

```
    operator int() const noexcept;          //返回栈的实际元素个数
    STACK& operator<<(int e);               //将 e 入栈，并返回当前栈
    STACK& operator>>(int& e);              //出栈到 e，并返回当前栈
    STACK& operator=(const STACK& s); //深拷贝赋值并返回被赋值栈
    STACK& operator=(STACK&& s)noexcept;//移动赋值并返回被赋值栈
    char* print(char* b)const noexcept;   //从栈底到栈顶打印栈元素
    ~STACK()noexcept;                       //销毁栈
};
```

②函数原型：STACK::STACK(int m);
函数功能：初始化栈：最多存放 2m-2 个元素
入口参数：长度 int m（隐含参数 this）
出口参数：无
流程：用 m 来初始化基类和成员变量队列

③函数原型：STACK::STACK(const STACK& s)；
函数功能：用栈 s 深拷贝初始化栈
入口参数：已知栈引用 const STACK& s（隐含参数 this）
出口参数：无
流程：将 s,s.q 强制转换成 QUEUE&类型，分别初始化基类和成员变量队列

④函数原型：STACK::STACK(STACK&& s)noexcept;
函数功能：用栈 s 移动拷贝初始化栈
入口参数：已知栈引用 STACK&& q（隐含参数 this）
出口参数：无
流程：将 s,s.q 强制转换成 QUEUE&&类型，分别初始化基类和成员变量队列

⑤函数原型：STACK::operator int() const noexcept;
函数功能：运算符重载，返回栈的实际元素个数
入口参数：无（隐含参数 this）
出口参数：栈实际元素个数
流程：利用队列类中已经实现的运算符重载，直接返回 q+QUEUE::operator int()即可

⑥函数原型：int STACK::size()const noexcept;
函数功能：返回栈的容量即 2m
入口参数：无（隐含参数 this）

出口参数：栈的容量

流程：返回 2*q.size()

⑦函数原型：STACK& STACK::operator<<(int e);

函数功能：运算符重载，将 e 入栈，并返回当前栈

入口参数：入队元素 int e（隐含参数 this）

出口参数：栈引用

流程图：

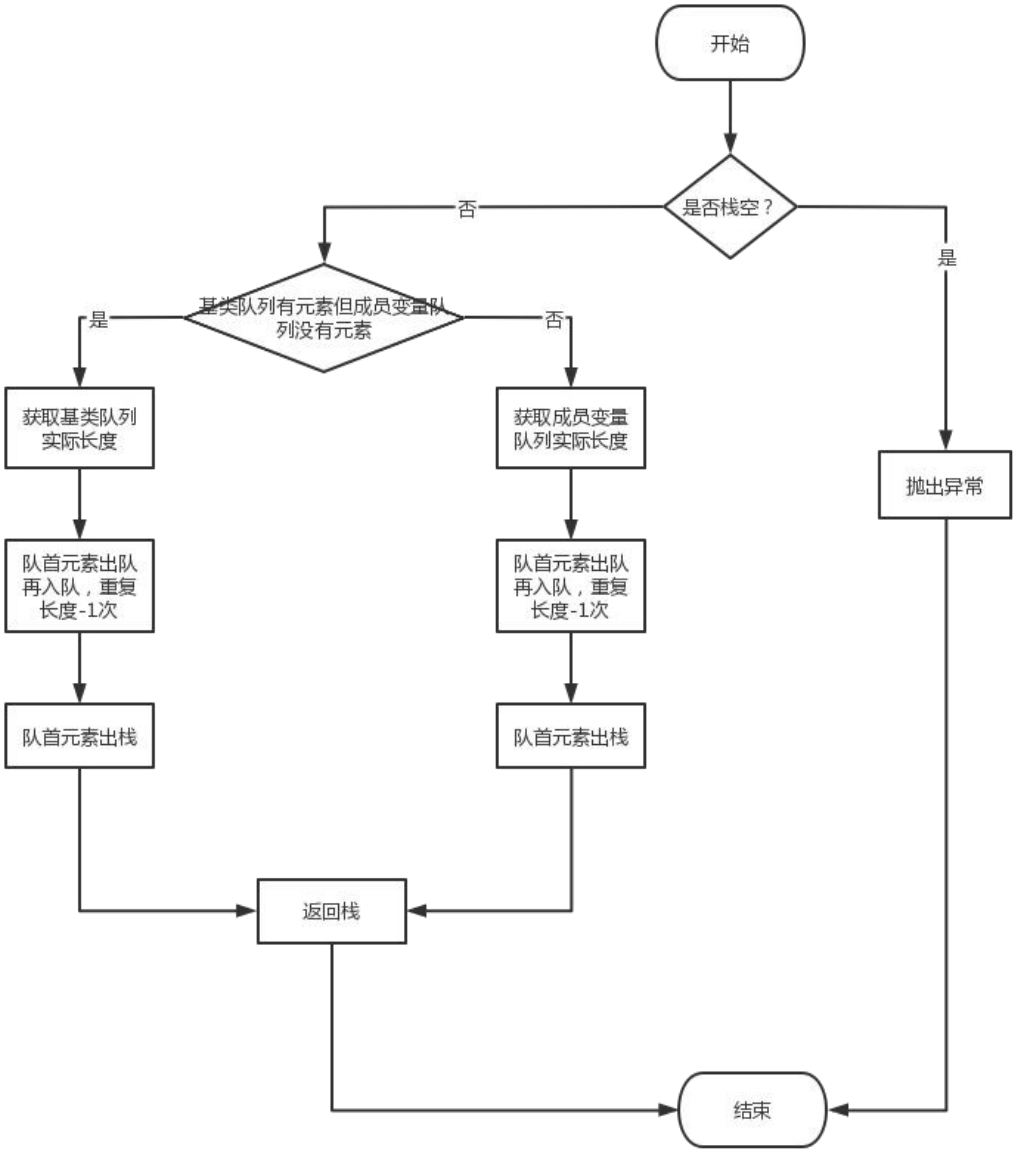

图 2-2 入栈函数流程图

⑧函数原型：STACK& STACK::operator>>(int& e);

函数功能：运算符重载，出栈到 e，并返回当前栈

入口参数：出栈元素引用 int& e（隐含参数 this）

出口参数：栈引用

流程图：

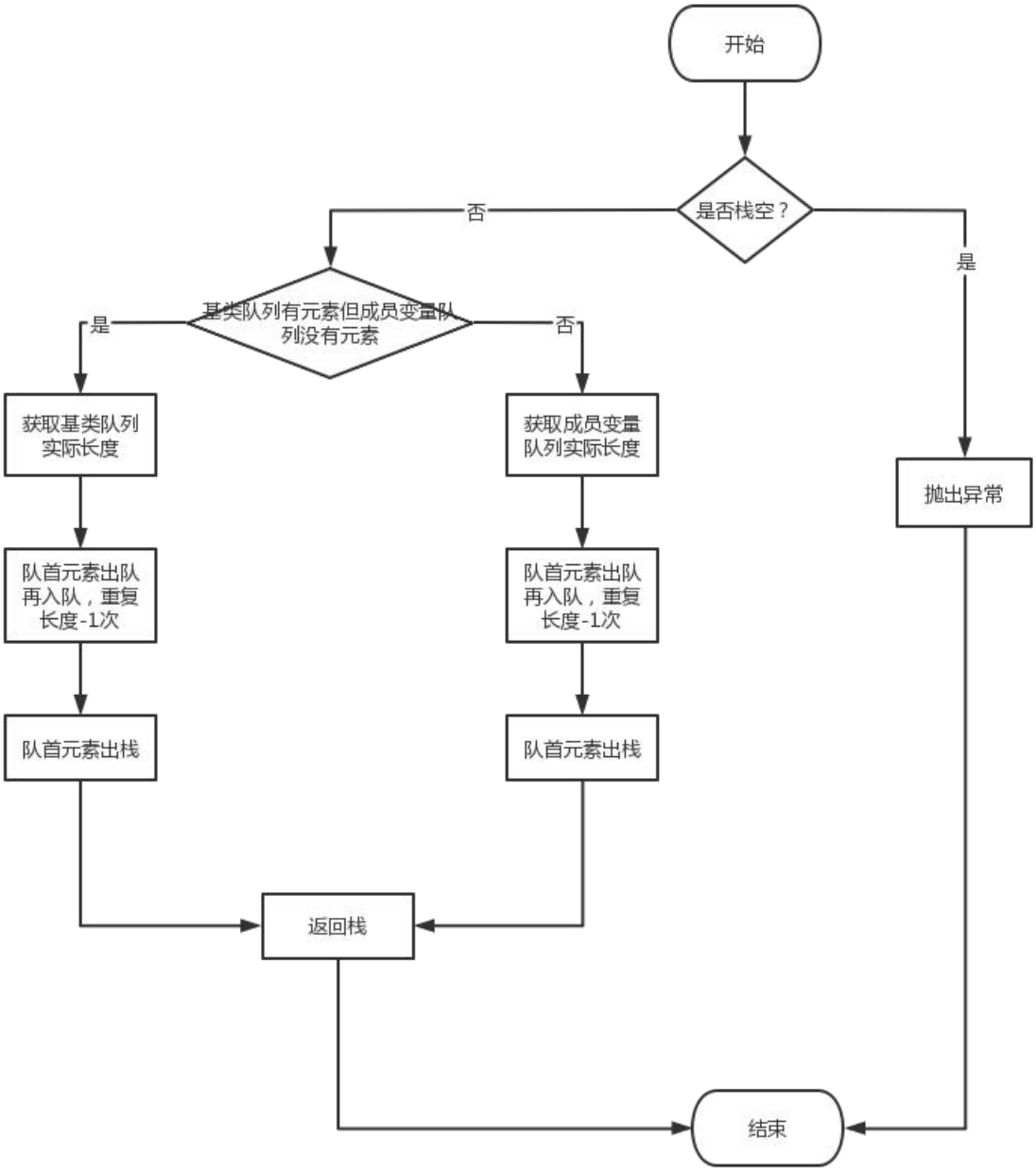

图 2-3 出队函数流程图

⑨函数原型：STACK& STACK::operator=(const STACK& s)；

函数功能：运算符重载，深拷贝赋值并返回被赋值栈

入口参数：已知栈引用 const STACK& s（隐含参数 this）

出口参数：栈引用

流程：如果 this==&s，直接返回*this，之后与前面定义的深拷贝初始化函数相同。

⑩函数原型：STACK& STACK::operator=(STACK&& s)noexcept;
函数功能：运算符重载，移动赋值并返回被赋值栈
入口参数：已知栈引用 STACK&& s（隐含参数 this）
出口参数：栈引用
流程：如果 this==&q，直接返回*this，之后与前面定义的移动初始化函数相同。

⑪函数原型：char* STACK::print(char* b)const noexcept;
函数功能：打印栈至 b 并返回 b
入口参数：输出字符串 char* b（隐含参数 this）
出口参数：输出字符串
流程：利用队列的 print 函数，先将基类队列打印到 b 中，再将成员变量队列打印到 b 中

⑫函数原型：STACK::~STACK()noexcept;
函数功能：销毁栈
入口参数：无（隐含参数 this）
出口参数：无

三、软件开发

采用 VS2019 开发，编译模式为 Debug-X86 模式，利用本地 Windows 进行调试。

四、软件测试

采用实验三测试库，测试结果如图 4-1 所示。

图 4-1　实验三测试图

五、特点与不足

1. 技术特点

代码规范性有了很大提升，同时运用类的继承，实现了代码复用，提升了便捷性。

2. 不足和改进的建议

没有直接包含实验二的源文件，而是将实验二的代码复制过来，这样显得代码比较冗余；同时有些函数可能在基类中是正常调用的，而如果在派生类中调用就会出现一些问题，比如下文中将要提到的 print 函数。

下一步要在代码简洁性上下一点功夫，同时应该仔细考虑之前的代码能不能直接在现在编写的程序中使用，要培养思维的严谨性。

六、过程和体会

1. 遇到的主要问题和解决方法

自己编写出栈函数的时候不知道该怎样将队尾元素出队，后来询问了同学，借鉴了他的“将队首元素出队再入队 n-1 次”的想法，最终成功编写出了符合功能的程序。

自己编写打印栈元素的函数的时候，刚开始的写法是((QUEUE)*this).print(b)，并不能通过测试，后来在老师的帮助下，发现运行此语句的时候，先进入队列的 print 函数，由于在队列的 print 函数中的循环条件是 i <（*this），而这里的 this 实际指向的是 stack，所以原本希望的是（*this）返回基类队列的实际长度，却由于 this 指向的是 stack 而返回了 stack 的实际长度，导致打印元素的时候出错，最后在老师的帮助下，将错误语句改写成了 QUEUE::print(b)，问题得以解决。

2. 课程设计的体会

自己在编写程序的时候可能会遇到很多问题，而遇到问题的时候不应该仅是询问别人之后改正自己的错误写法，而是要深入探究错误的原因，再次深入学习一下其中蕴含的理论知识，牢牢得以此为戒，防止下次再犯。同时应该开拓自己的思路，不能因为现有的函数无法直接实现功能而变得手足无措，应该去思考如何灵活的运用现有的函数，实现目标功能。

七、源码和说明

1. 文件清单及其功能说明

ex3_stack.h 是头文件（类声明）

ex3_stack.cpp 是源文件（栈函数定义）

Queue.cpp 是源文件（队列函数定义）

ex3_test.cpp 是源文件（main 函数）

ex3_stack.sln 用来打开项目

“实验三测试库”文件夹中是本次实验所需要的测试库文件

其余文件为 vs 项目配置文件

2. 用户使用说明书

使用时，双击 ex3_stack.sln 用来打开项目进入 vs 界面，将测试库添加进入工程，然后点击界面上方的“本地 Windows 调试器”即可运行程序。

3. 源代码

ex3_stack.h:

```
#pragma once
#include <stdio.h>
#include <string.h>
class QUEUE {
    int* const  elems;  //elems 申请内存用于存放队列的元素
    const  int  max; //elems 申请的最大元素个数为 max
    int   head, tail;//队列头 head 和尾 tail，队空 head=tail;初始 head=tail=0
public:
    QUEUE(int m);    //初始化队列：最多申请 m 个元素
    QUEUE(const QUEUE& q);          //用 q 深拷贝初始化队列
    QUEUE(QUEUE&& q)noexcept;       //用 q 移动初始化队列
    virtual operator int() const noexcept;    //返回队列的实际元素个数
    virtual int size() const noexcept;    //返回队列申请的最大元素个数 max
    virtual QUEUE& operator<<(int e);     //将 e 入队列尾部，并返回当前队列
    virtual QUEUE& operator>>(int& e);    //从队首出元素到 e，并返回当前队列
    virtual QUEUE& operator=(const QUEUE& q);//深拷贝赋值并返回被赋值队列
    virtual QUEUE& operator=(QUEUE&& q)noexcept;//移动赋值并返回被赋值队列
    virtual char* print(char* s) const noexcept;//打印队列至 s 并返回 s
    virtual ~QUEUE();                     //销毁当前队列
};
class STACK : public QUEUE {
    QUEUE q;
public:
    STACK(int m);                         //初始化栈：最多存放 2m-2 个元素
    STACK(const STACK& s);                //用栈 s 深拷贝初始化栈
    STACK(STACK&& s)noexcept;           //用栈 s 移动拷贝初始化栈
    int size()const noexcept;             //返回栈的容量即 2m
    operator int() const noexcept;        //返回栈的实际元素个数
```

```
    STACK& operator<<(int e);              //将 e 入栈，并返回当前栈
    STACK& operator>>(int& e);             //出栈到 e，并返回当前栈
    STACK& operator=(const STACK& s); //深拷贝赋值并返回被赋值栈
    STACK& operator=(STACK&& s)noexcept;//移动赋值并返回被赋值栈
    char* print(char* b)const noexcept;   //从栈底到栈顶打印栈元素
    ~STACK()noexcept;                      //销毁栈
};
```

Queue.cpp:

```
#include"ex3_stack.h"
QUEUE::QUEUE(int m) :elems(new int[m]), max(m), head(0), tail(0) {}//初始化队列：最多申请 m 个元素
QUEUE::QUEUE(const QUEUE& q) : elems(new int[q.max]), max(q.max), head(q.head), tail(q.tail)
{
    for (int i = 0; i < q.max; i++)
        elems[i] = q.elems[i];
}//用 q 深拷贝初始化队列
QUEUE::QUEUE(QUEUE&& q)noexcept :elems(q.elems), max(q.max), head(q.head), tail(q.tail)
{
    *(int**)&q.elems = 0;
    *(int*)&q.max = 0;
    q.head = q.tail = 0;
}//用 q 移动初始化队列
QUEUE::operator int() const noexcept
{
    if (elems)
        return (tail - head + max) % max;
    return 0;
```

```
}//返回队列的实际元素个数
int QUEUE::size() const noexcept
{
    return max;
}//返回队列申请的最大元素个数 max
QUEUE& QUEUE::operator<<(int e)
{
    if ((tail + 1) % max == head)
        throw"QUEUE is full!";
    elems[tail] = e;
    tail = (tail + 1) % max;
    return *this;
}//将 e 入队列尾部，并返回当前队列
QUEUE& QUEUE::operator>>(int& e)
{
    if (head == tail)
        throw"QUEUE is empty!";
    e = elems[head];
    head = (head + 1) % max;
    return *this;
}//从队首出元素到 e，并返回当前队列
QUEUE& QUEUE::operator=(const QUEUE& q)
{
    if (this == &q) return *this;
    if (elems)
    {
        delete[] elems;
        *(int**)&elems = 0;
    }
    *(int**)&elems = new int[q.max];
```

```
    *(int*)&max = q.max;
    head = q.head;
    tail = q.tail;
    for (int i = 0; i < q.max; i++)
        elems[i] = q.elems[i];
    return *this;
}//深拷贝赋值并返回被赋值队列
QUEUE& QUEUE::operator=(QUEUE&& q)noexcept
{
    if (this == &q) return *this;
    if (elems)
    {
        delete[] elems;
        *(int**)&elems = 0;
    }
    *(int**)&elems = q.elems;
    *(int*)&max = q.max;
    tail = q.tail;
    head = q.head;
    *(int**)&q.elems = 0;
    *(int*)&q.max = 0;
    q.head = q.tail = 0;
    return *this;
}//移动赋值并返回被赋值队列
char* QUEUE::print(char* s) const noexcept
{
    for (int i = 0; i < QUEUE::operator int(); i++)
    s += sprintf(s, "%d,", elems[(head + i) % max]);
    return s;
}//打印队列至 s 并返回 s
```

```
QUEUE::~QUEUE()
{
    if (elems)
    {
        delete[] elems;
        *(int**)&elems = 0;
        head = tail = *(int*)&max = 0;
    }
}//销毁当前队列 e
```

ex3_stack.cpp：

```
#include"ex3_stack.h"
STACK::STACK(int m) :QUEUE(m), q(m) {}
//初始化栈：最多存放 2m-2 个元素
STACK::STACK(const STACK& s) : QUEUE((QUEUE&)s), q((QUEUE&)s.q) {}
//用栈 s 深拷贝初始化栈
STACK::STACK(STACK&& s)noexcept : QUEUE((QUEUE&&)s), q((QUEUE&&)s.q) {}
//用栈 s 移动拷贝初始化栈
int STACK::size()const noexcept
{
    return 2 * q.size();
}//返回栈的容量即 2m
STACK::operator int() const noexcept
{
    return q + QUEUE::operator int(); //运算符重载
}//返回栈的实际元素个数
STACK& STACK::operator<<(int e)
{
    if (*this == size()-2)
        throw("STACK is full!");
```

```
    if (QUEUE::operator int()<QUEUE::size()-1) //先进入基类队列
        QUEUE::operator<<(e);
    else q << e;
    return *this;
}//将 e 入栈，并返回当前栈
STACK& STACK::operator>>(int& e)
{
    if(*this == 0)
        throw("STACK is empty!");
    if (QUEUE::operator int() && q==0)
    {
        int temp;
        int count = QUEUE::operator int() - 1;
        while (count) //使队尾元素变成队首元素
        {
            QUEUE::operator>>(temp);
            QUEUE::operator<<(temp);
            count--;
        }
        QUEUE::operator>>(e);
    }
    else
    {
        int temp;
        int count = q - 1;
        while (count) //使队尾元素变成队首元素
        {
            q >> temp;
            q << temp;
            count--;
```

```
        }
        q >> e;
    }
    return *this;
}//出栈到 e，并返回当前栈
STACK& STACK::operator=(const STACK& s)
{
    if (this == &s) return *this;
    QUEUE::operator=((QUEUE&)s);
    q = s.q;
    return *this;
}//深拷贝赋值并返回被赋值栈
STACK& STACK::operator=(STACK&& s)noexcept
{
    if (this == &s) return *this;
    QUEUE::operator=((QUEUE&&)s);
    q =(QUEUE&&)s.q;
    return *this;
}//移动赋值并返回被赋值栈
char* STACK::print(char* b)const noexcept
{
    QUEUE::print(b);
    q.print(b + strlen(b)); //print 的参数是地址
    return b;
}//从栈底到栈顶打印栈元素
STACK::~STACK()noexcept {}//销毁栈
```

ex3_test.cpp:

```
#include<iostream>
#include"ex3_stack.h"
```

```
extern const char* TestSTACK(int& s); //测试函数
int main()
{
    int s; //分数
    const char* q = TestSTACK(s); //提示信息字符串
    printf("%d, %s\n", s, q);
    return 0;
}
```