

**课 程 实 验 报 告**

**课程名称：面向对象程序设计**

**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 ： CS1804**

**学 号 ： U201814604**

**姓 名 ： 黄俊淇**

**指导教师 ： 纪俊文**

**年 月 日**

1. **需求分析**
2. **题目要求**

**整型栈是一种先进先出的存储结构，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。为了便于测试程序测试，压栈时请大家默认先将元素进入基类队列中。若基类队列已满而队列q不满时，设法从基类队列挪出一个元素到q，然后再往基类队列加入要压栈的元素。**

**（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。**

1. **需求分析**

只能对一个队列进行出栈，入栈操作。打印时，直接用继承的打印函数即可。

1. **系统设计**
2. **概要设计**

继承队列类，利用两个循环队列模拟栈，将先入先出变换为后进先出。如图3-1所示。

图示

描述已自动生成

图3-1总体结构

1. **详细设计**
2. 构造函数：STACK(int m)。参数m为int型，输入m初始化基类和成员对象q，创建两个大小为m的循环队列。
3. 复制构造函数：STACK(const STACK& s)。参数s为STACK类，输入一个STACK类初始化基类和成员对象q，创建两个大小与s相同的循环队列。
4. 移动构造函数：STACK(STACK&& s)noexcept。参数s为右值STACK类，输入一个右值STACK类初始化基类和成员对象q，将基类和成员对象的队列的地址赋为s的基类和成员对象q的队列，再释放s的基类和成员对象q的队列。
5. 求容量：int size()const noexcept。无参，返回值为int型。返回2\*q.size()。
6. 求实际元素数量：operator int() const noexcept。返回基类和成员对象q的实际个数的和。
7. 入栈：STACK& operator<<(int e)。参数e为int型，返回值为STACK类的引用。流程图如图3-2所示。

图示

描述已自动生成

图3-2入栈流程图

1. 出栈：STACK& operator>>(int& e)。参数e为int型引用，返回值为STACK类的引用。流程图如图3-3所示。

图示

描述已自动生成

图3-3出栈流程图

1. 深拷贝复制：STACK& operator=(const STACK& s)。参数s为STACK类引用，返回值为STACK类的引用。利用父类QUEUE的深拷贝复制函数，为子类STACK深拷贝复制，返回\*this。
2. 移动拷贝：STACK& operator=(STACK&& s)noexcept。参数s为右值STACK类，返回值为STACK类的引用。利用父类QUEUE的移动拷贝函数，为子类STACK移动拷贝，返回\*this。
3. 打印：char\* print(char\* b)const noexcept。参数b为指向char的指针，返回值为指向char的指针。将队列元素以栈的方式输出到b，返回b。
4. 析构函数：~STACK()noexcept。
5. **软件开发**

开发环境VS2019，在VC++目录增加测试库目录，在链接目录增加测试文件。

1. **软件测试**

（1）自我测试

测试代码：

char t[2018];

int x;

STACK a(10);

a << 1;

a << 2;

a.print(t);

cout << t << endl;

STACK b(a);

a >> x;

cout << x << endl;

b.print(t);

cout << t << endl;

a.print(t);

cout << t << endl;

a = b;

a.print(t);

cout << t << endl;

a >> x;

a >> x;

cout << x << endl;

a = move(b);

a.print(t);

cout << t << endl;

测试结果：如图3-4所示

文本

描述已自动生成

图3-4自我测试结果

（2）测试库测试，如图3-5所示

图片包含 文本

描述已自动生成

图3-5测试库测试结果

由两次测试结果可知，所有函数功能均以实现且满足任务要求。

1. **特点与不足**
2. **技术特点**

利用循环队列的先进先出，可以通过出队列再入队列的方式将先进的元素“挪”到后面，不需要额外空间，实现栈的后进先出特性。

1. **不足和改进的建议**

一开始想的是利用两个队列互相入队列和出队列实现后进先出，但忘记了是循环队列，可以直接将队列的元素通过操作实现后进先出。

1. **过程和体会**
2. **遇到的主要问题和解决方法**

测试时，测试库显示success，但却只有96分。通过老师改进的测试库发现是入栈和出栈未符合文档要求。

解决办法：只对基类队列进行入栈和出栈操作。

1. **课程设计的体会**

队列和栈在一定程度上可以互相转化，继承是面向对象设计的重要特性，通过继承可以在原有类的基础上保留共性，新增特性。

1. **源码和说明**
2. **文件清单及其功能说明**

QUEUE.h：QUEUE类的声明

QUEUE.cpp：QUEUE类的定义

Lab3.cpp：主要子程序

Lab3.exe：可执行文件

1. **用户使用说明书**

运行可执行文件即可。

1. **源代码**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

#include<iostream>

#include<cstring>

#include "QUEUE.h"

extern const char\* TestSTACK(int& s);

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

};

int main() {

using namespace std;

char t[2018];

int x;

STACK a(10);

a << 1;

a << 2;

a.print(t);

cout << t << endl;

STACK b(a);

a >> x;

cout << x << endl;

b.print(t);

cout << t << endl;

a.print(t);

cout << t << endl;

a = b;

a.print(t);

cout << t << endl;

a >> x;

a >> x;

cout << x << endl;

a = move(b);

a.print(t);

cout << t << endl;

//STACK c (move(b));

//c.print(t);

//cout << t << endl;

int nmark = 0;

const char\* strResult = TestSTACK(nmark);

cout << "学号：U201814604" << endl;

cout << "姓名：黄俊淇" << endl;

cout << "实验三测试：" << strResult << ',' << nmark << endl;

return 0;

}

STACK::STACK(int m) :QUEUE(m), q(m) {

}

STACK::STACK(const STACK& s) : QUEUE(s), q(s.q) {

}

STACK::STACK(STACK&& s)noexcept : QUEUE(std::move(s)), q(std::move(s.q)) {

}

int STACK::size()const noexcept {

return 2\*q.size();

}

STACK::operator int()const noexcept {

int q1 = QUEUE::operator int();

int q2 = q.operator int();

return q1 + q2;

}

STACK& STACK::operator<<(int e) {

int q1 = QUEUE::operator int();

int q2 = q.operator int();

int temp;

if((q1+q2)==(2\*q.size()-2)) throw("STACK is full!");

if (q1 == (q.size() - 1) && q2 < (q.size()- 1)) {

this->QUEUE::operator>>(temp);

q.operator<<(temp);

this->QUEUE::operator<<(e);

}

else if (q1 < (q.size() - 1) && q2 == (q.size() - 1)) {

this->QUEUE::operator<<(e);

}

else if (q1 == 0&&q2!=0) {

q.operator<<(e);

}

else if(q2==0&&q1!=0) {

this->QUEUE::operator<<(e);

}

else {

this->QUEUE::operator<<(e);

}

return \*this;

}

STACK& STACK::operator>>(int &e) {

int q1 = QUEUE::operator int();

int q2 = q.operator int();

int num;

if (q1 == 0 && q2 == 0) throw("STACK is empty!");

int temp;

if (q1 == 0 && q2 != 0) {

while (q2 > 1) {

q.operator>>(temp);

this->QUEUE::operator<<(temp);

--q2;

}

q.operator>>(e);

}

else if (q2 == 0 && q1 != 0) {

while (q1 > 1) {

this->QUEUE::operator>>(temp);

q.operator<<(temp);

--q1;

}

this->QUEUE::operator>>(e);

}

else {

while (q1 > 1) {

this->QUEUE::operator>>(temp);

this->QUEUE::operator<<(temp);

q1--;

}

this->QUEUE::operator>>(e);

}

return \*this;

}

STACK& STACK::operator=(const STACK& s) {

if (this == &s) return \*this;

//if (q.size() != 0) q.~QUEUE();

//if (this->QUEUE::size() != 0) this->QUEUE::~QUEUE();

this->QUEUE::operator=(s);

q.operator=(s.q);

return \*this;

}

STACK& STACK::operator=(STACK&& s)noexcept {

if (this == &s) return \*this;

this->QUEUE::operator=(std::move(s));

q.operator=(std::move(s.q));

return \*this;

}

char\* STACK::print(char\* b)const noexcept {

int q1 = this->QUEUE::operator int();

int q2 = q.operator int();

if (q1 == 0 && q2 == 0) throw("STACK is empty!");

//std::string s;

int i = 0;

int j=0;

if (q1 != 0 && q2 != 0) {

q.print(b);

int m = 2 \* q2;

b[m - 1] = ',';

this->QUEUE::print(&b[m]);

b[2\*m-1] = '\0';

}

else if (q1 == 0 && q2 != 0) {

q.print(b);

}

else if (q1 != 0&&q2==0 ) {

this->QUEUE::print(b);

//if (q2 != 0) q.print(b + 6);

}

/\*std::string temp[10];

while (s[i] != '\0') {

if (s[i] == ',') {

i++;

j++;

continue;

}

temp[j] += s[i];

}\*/

//std::cout << b;

return b;

}

STACK::~STACK()noexcept{

// if(q.size()!=0) q.~QUEUE();

//if(this) this->QUEUE::~QUEUE();

}