

华中科技大学

课程实验报告

课程名称：面向对象程序设计

实验名称：继承对象的整型队列编程

院 系：计算机科学与技术

专业班级：计卓 1501 班

学 号：U201514898

姓 名：胡思勛

指导教师：纪俊文

2017 年 11 月 28 日

一、需求分析

1. 题目要求

整型队列是一种先进后出的存储结构，对其进行的操作通常包括判断队列是否为空、向队列顶添加一个整型元素、出队列等。整型队列类型及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函数采用 C++ 编程。然后写一个 main 函数对队列的所有操作函数进行测试，要求 main 按照《关于 C++ 实验自动验收系统说明》给定的方式工作。注意，请用实验三的 SATCK 继承形成新的类 QUEUE。分析说明除构造函数以外的函数，加 virtual 说明与不加 virtual 说明有无区别。并说明为什么不将 s2 也作为基类。

```
class QUEUE: public STACK {
    STACK s2;
public:
    QUEUE(int m);           //每个栈最多m个元素，要求实现的队列最
多能入2m个元素
    QUEUE(const QUEUE&s);   //用队列s拷贝初始化队列
    virtual operator int ( ) const; //返回队列的实际元素个数
    virtual int full ( ) const; //返回队列是否已满，满返回1，否则返回0
    virtual int operator[ ](int x)const; //取下标为x的元素，第1个元素下标为0
    virtual QUEUE& operator<<(int e); //将e入队列，并返回队列
    virtual QUEUE& operator>>(int &e); //出队列到e，并返回队列
    virtual QUEUE& operator=(const QUEUE &s); //赋s给队列，并返回被赋值的队列
    virtual void print( ) const; //打印队列
    virtual ~QUEUE( ); //销毁队列
};
```

注意：虽然实现的队列 最多能入 2m 个元素，但是由于模拟中入出操作序列的限制，即操作不能违背先进先出的原则，可能还没入 2m 个元素队列就“满”了，所以要注意 full() 函数的实现：不能简单判断队列是否装了 2m 个元素。

在完成上述程序及测试无误后，请使用队列解决如下舞伴问题，此时 main 用非命令行的方式工作。假定在一次舞会上，男士排成一队，女士排成另一队。每次舞曲响起时，从男队和女队的队头各出一人，配成舞伴跳完此曲，跳完后各自进入自己队列的尾部。若男女两队的初始人数分别为 M 和 F（M 和 F 均为素数，且 $M \neq F$ ），男队中排在位置 m（ $m \leq M$ ）的男士，非常想和女队位置 f（ $f \leq F$ ）的女士跳舞，问他在第几支曲舞曲才能和该女士跳舞？请编程解决上述问题。

2. 需求分析

在整型队列的实现中，对于部分方法有功能上的需求，具体需求如下：

1. 对于所有分配空间的操作，在内存不足时应该抛出异常。
2. 对于需要访问数组下标的操作，如 `operator []`，`operator <<`，`operator >>` 等，在下表访问越界（小于 0 或大于等于数组长度时应该抛出异常，包括：访问数组访问越界，队列满时尝试入队列，队列空时尝试出队列。
3. 程序不能有内存泄漏，所有申请的内存最终都需要释放。
4. 程序性能在要求下应该尽可能高。
5. 对于舞伴问题而言，需要对输入进行合法性判断。

二、系统设计

1. 概要设计

由于要使用栈来实现队列，按照给定的接口形式，队列的结构应该为：使用 `QUEUE` 类继承 `STACK` 类，并让这个 `QUEUE` 类持有另一个 `STACK` 实例，然后通过 `STACK` 的入栈和出栈方法，通过将元素在两个栈之间交换来实现入队列和出队列的过程。

具体模块关系如下：首先定义栈类，在其中定义构造函数，复制构造函数，析构函数，`size()`，`operator int`，`operator []`，`operator <<`，`operator >>`，`operator =`，`print` 方法。栈类作为基类，在其上公有派生出队列类并让这个队列类持有另外一个栈类，这样这一队列类中就包含了继承而来的和作为成员的两个栈类，然后通过栈的方法来实现队列类的方法。栈类的所有方法均为虚方法，队列的方法重载了栈的方法，因此在操作继承而来的栈时，需要显式地调用父类的方法以避免歧义。总体的模块图以及调用关系图如图 1 所示：

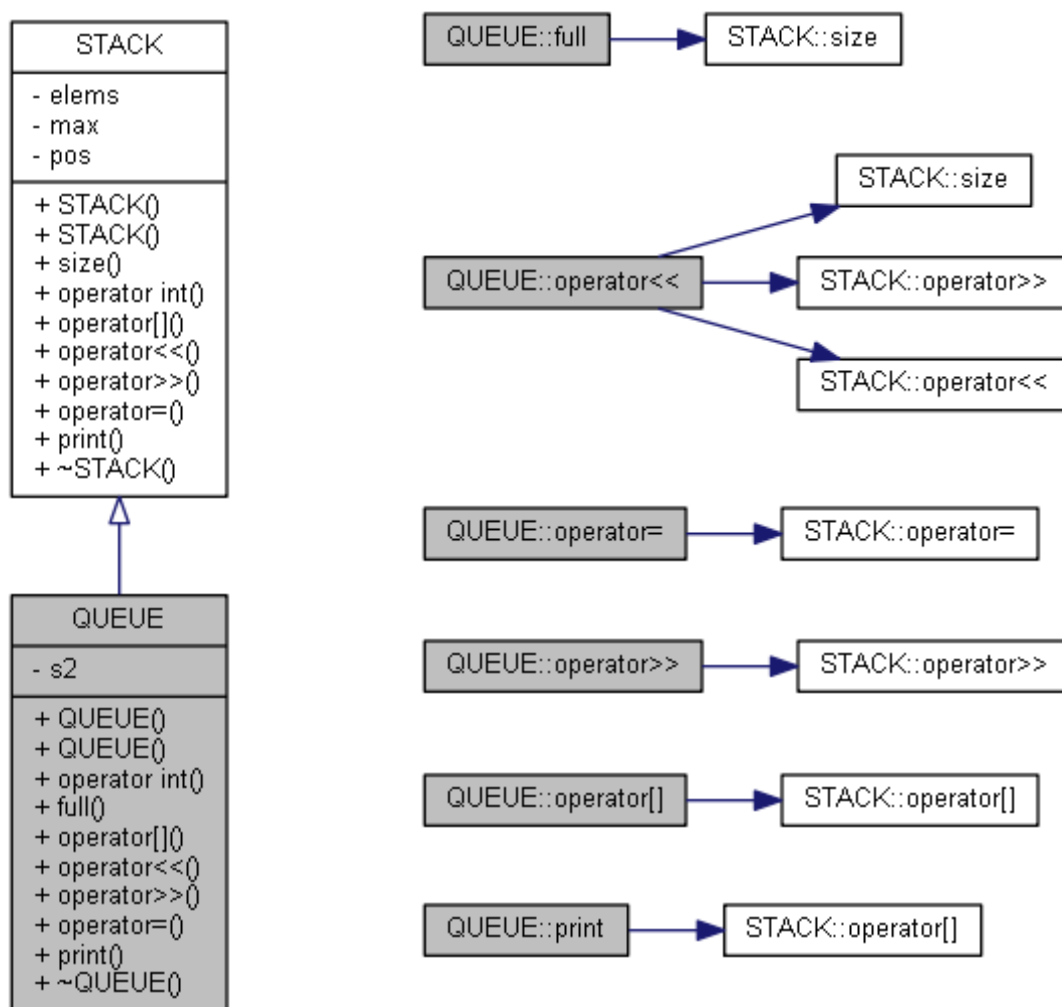


图 1 模块——调用关系图

对于主函数而言，其总体流程如下：首先读取 `argc` 判断参数的个数，若没有参数(`argc == 1`) 则执行舞伴问题，否则执行对于队列类的测试。总体的流程图框架如图 2 所示。

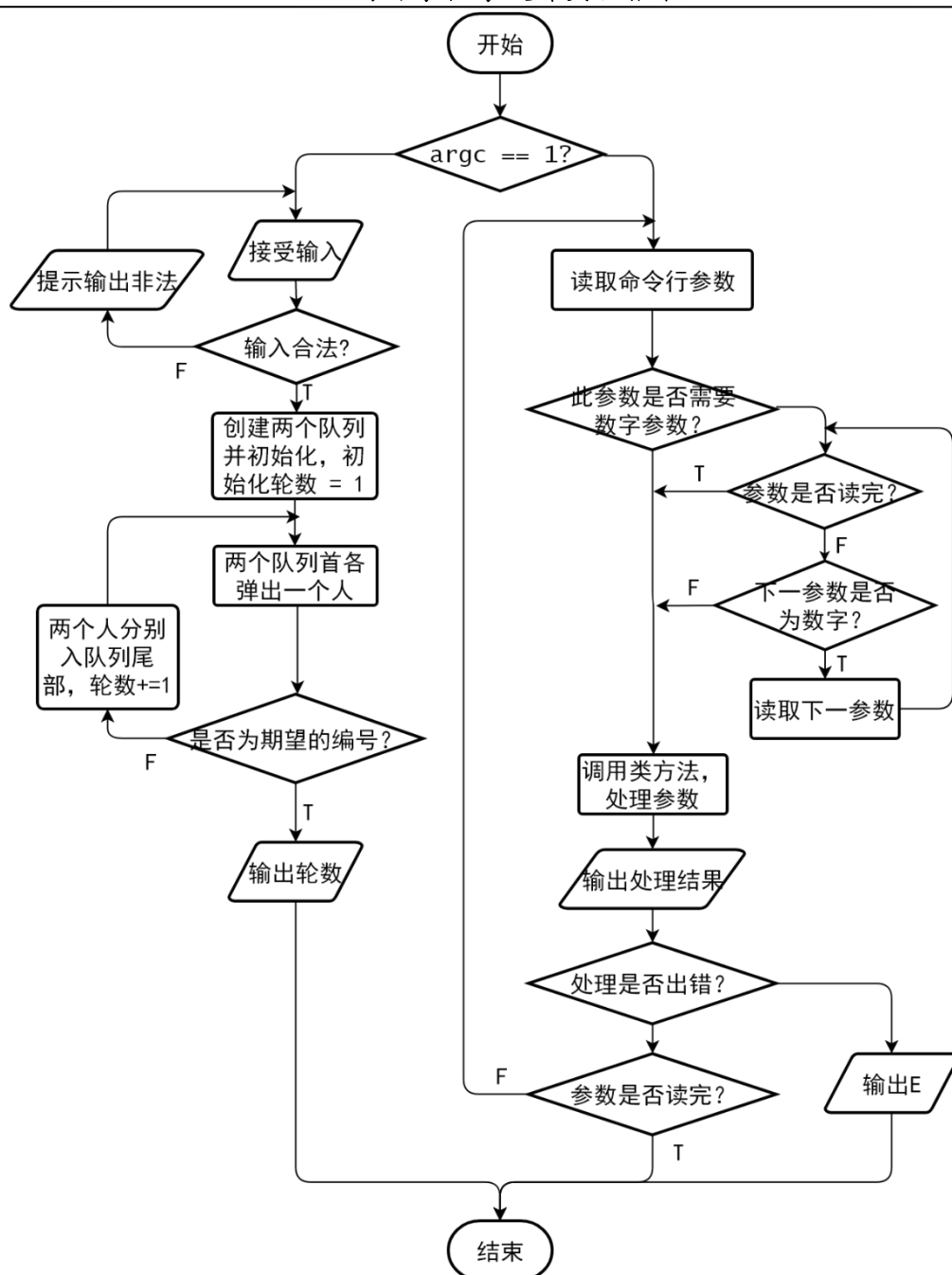


图 2 总体流程图

对于舞伴问题而言，首先要接受 4 个输入，然后判断输入是否合法，包括判断男士和女士人数是否为质数，m 和 f 是否越界等，若输入不合法，需要提示用户重新输入。若输入检查无误，则初始化两个队列代表男士和女士，大小分别为输入的两个人数，并将两个队列按 1, 2, 3, ... 的顺序初始化，并初始化计数器为 1。在每一轮分别从两队列首弹出两个元素，若两个元素的值分别为 m 和 f，则代表男士 m 和女士 f 成功跳上舞，输出计数器的值并结束程序，否则将弹出的值压入队列尾部，计数器加一并进行下一轮模拟。

当 argc 不为 1 时进行自动测试，开始逐个读入参数。对于要读入的字母参数分为两类，一类如 -I 等后面需要接一个或多个数字的参数，另一类如 -N 等后面不需要接数字，依次为依据决定是否继续读入参数。当一组参数读入完成后，调用需要用于测试的类方法来测试参数，如果在测试中出现错误，则输出 E 并停止测试，否则按照测试的需求输出测试结果并继续测试，直到全部参数读完或出现错误。

2. 详细设计

以下展示了 `STACK` 和 `QUQUE` 类各个类方法的程序说明，入口参数说明，出口参数（返回值）说明，并对每个方法方法给出了详细的解释和流程图。

STACK 部分

栈使用整型数组 `elems` 实现，在构造时指定其大小并分配空间，并在析构时销毁。使用两个辅助成员 `pos` 和 `max` 用于指定当前栈中的元素个数和栈的容量。在构造时 `pos` 初始化为 0 并随着入栈和出栈而变化，在入栈时 `pos` 自增，在出栈时 `pos` 自减，`max` 为 `const` 值，初始化为一个固定大小后不再变化，栈的结构如图 3 所示。

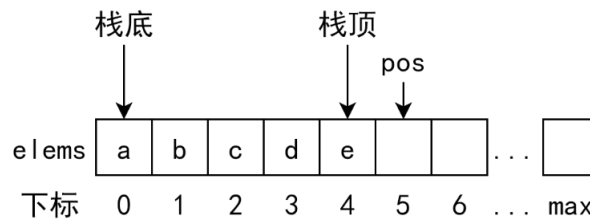


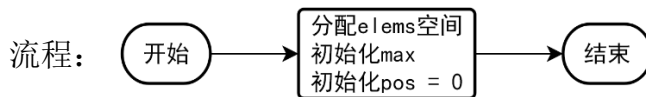
图 3 栈的结构

➤ `STACK::STACK(int m)`

参数: `m`: 栈的大小

返回: 构造函数无返回值

说明: 构造函数，构造一个栈并将其大小初始化为 `m`。

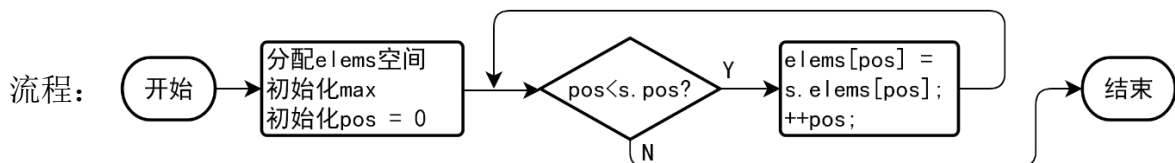


➤ `STACK::STACK(const STACK &s)`

参数: `s`: 要复制的栈

返回: 构造函数无返回值

说明: 复制构造函数，构造一个大小与 `s` 相同的栈并将 `s` 中的元素拷贝到新栈中。

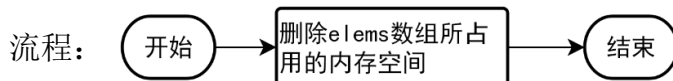


➤ `STACK::~~STACK()`

参数: 析构函数无参数

返回: 析构函数无返回值

说明: 析构函数，销毁当前栈并释放构造时所申请的栈空间。



➤ `STACK::operator int() const`

参数： 无

返回： 栈中元素的个数

说明： 重载 `int()` 运算符，获取当前的栈中所存放的元素个数。

流程：



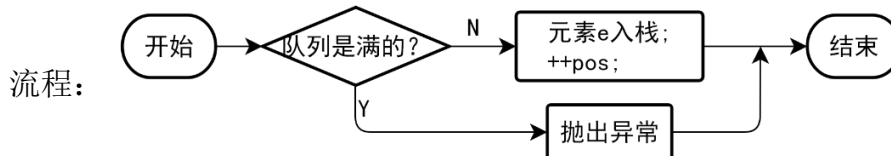
➤ `STACK &STACK::operator<<(int e)`

参数： `e`: 将要压入栈中的元素

返回： 栈本身

说明： 重载 `<<` 运算符，将元素 `e` 压入栈顶，并返回栈本身。

异常： 当栈空间已满时，抛出 `std::logic_error` 异常。



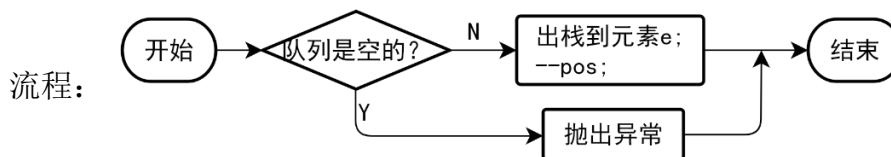
➤ `STACK &STACK::operator>>(int &e)`

参数： `e`: 弹出的元素

返回： 栈本身

说明： 重载 `>>` 运算符，弹出栈顶的元素，将其赋值给 `e`，并返回栈本身。

异常： 当栈为空时，抛出 `std::logic_error` 异常。

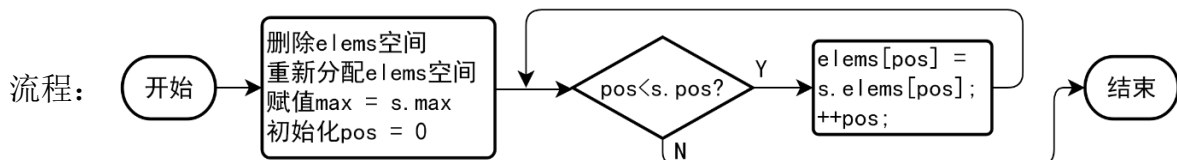


➤ `STACK &STACK::operator=(const STACK &s)`

参数： `s`: 被复制的栈

返回： 栈本身

说明： 重载 `=` 运算符，将本栈的大小更改为与栈 `s` 的大小相同，然后将栈 `s` 的内容赋值给本栈，并返回本栈。由于栈的大小是由 `const int max` 指定的，在更改 `max` 的值时需要首先使用 `const_cast` 获得一个指向 `max` 的非 `const` 指针再赋值。此过程需要删除原来的数组空间并重新分配空间。



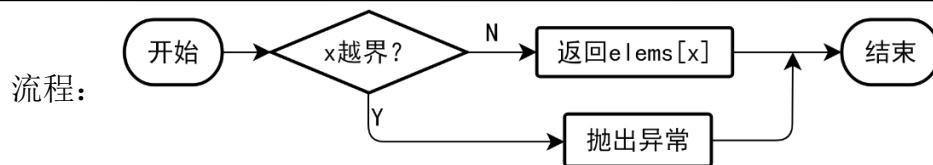
➤ `int STACK::operator[](int x) const`

参数： `x`: 所取值的下标

返回： 栈中下标为 `x` 的值

说明： 重载 `[]` 运算符，取栈中下标为 `x` 的元素并返回其值。

异常： 当 `x` 越界时（即 `x < 0` 或 `x ≥ pos`）则抛出 `std::logic_error` 异常。

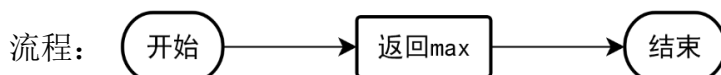


➤ `int STACK::size() const`

参数: 无

返回: 栈的容量

说明: 直接返回 `max`, 即栈可以容纳的元素个数。

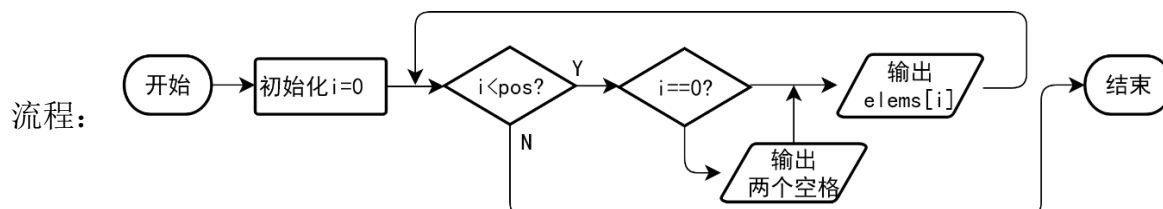


➤ `void STACK::print() const`

参数: 无

返回: 无

说明: 遍历成员数组, 并打印其中存放的元素, 每两个元素间有两个空格。



QUEUE 部分

QUEUE 使用两个 STACK 来实现。QUEUE 类继承 STACK 类, 并在 QUEUE 类中有一个 STACK 成员, 即在实例化后 QUEUE 示例内含有一个继承而来的 STACK 和一个持有的 STACK `s2`, 以下分别称为 `STACKin` 和 `STACKout`。STACK0 用于入队列, `STACKout` 用于出队列。即在尝试将元素加入队列尾时队列总是尝试将其压入 `STACKin`, 而在尝试出队列时总是尝试从 `STACKout` 弹出元素。队列的结构如图 4 所示。

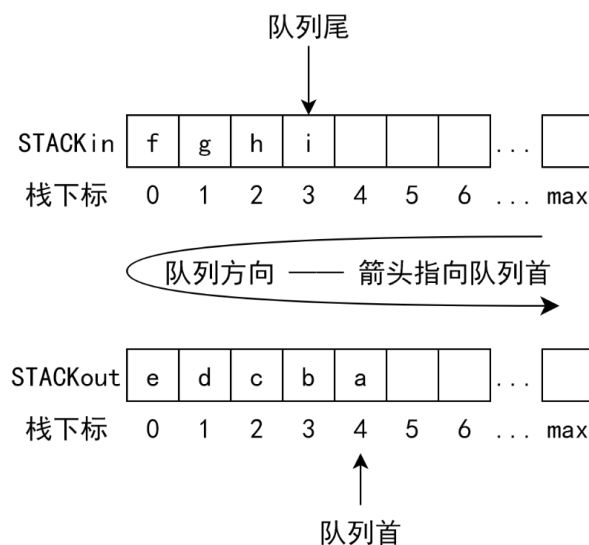


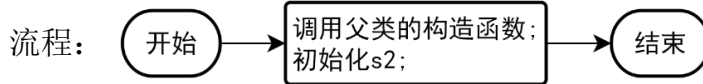
图 4 队列结构

➤ `QUEUE::QUEUE(int m)`

参数: `m`: 队列中栈的大小 (队列大小为 $2m$)

返回: 构造函数无返回值

说明: 构造函数, 构造一个队列并初始化其继承而来的栈和其持有的栈大小均为 `m`, 这样此队列最多能存放 $2m$ 个元素, 初始化栈时在初始化列表中直接调用栈的构造函数。

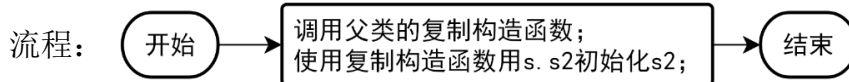


➤ `QUEUE::QUEUE(const QUEUE &s)`

参数: `s`: 要复制的队列

返回: 构造函数无返回值

说明: 复制构造函数, 构造一个大小和内容与 `s` 均相同的队列。直接在初始化列表中调用栈的复制构造函数分别构造本队列继承而来的栈和持有的栈。

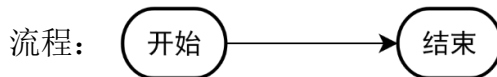


➤ `QUEUE::~~QUEUE()`

参数: 析构函数无参数

返回: 析构函数无返回值

说明: 析构函数, 销毁当前队列并释放构造时所申请的空间。对于持有的栈和继承而来的栈而言, 在销毁本队列时会自动为成员变量以及基类调用其析构函数, 因此不用手动销毁, 直接返回即可。

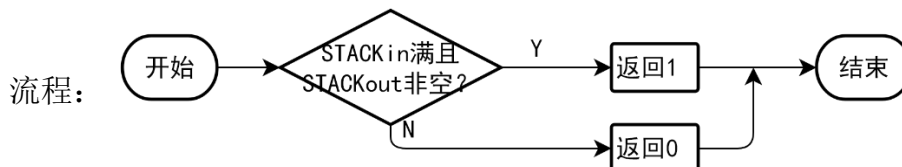


➤ `int QUEUE::full() const`

参数: 无

返回: 若当前队列已满则返回 1, 否则返回 0

说明: 由于队列使用两个栈实现, 用于判断队列已满的依据并不是直接判断两个栈是否分别为满, 而是队列是否还能压入元素。由于 `STACKin` 用于入队列, 而 `STACKout` 用于出队列, 那么当继承的为满且 `STACKout` 为空的时候不能再向队列尾部添加元素。因为此时不能将继承的栈的元素转移到 `STACKout` 中。除此以外的情况均可以再向队列尾部添加元素。



➤ `QUEUE::operator int() const`

参数: 无

返回: 队列中元素的个数

说明: 重载 `int()` 运算符, 返回队列中元素的个数, 即两个栈中的元素个数之和。



➤ `QUEUE &QUEUE::operator<<(int e)`

参数: `e`: 插入队列尾部的元素

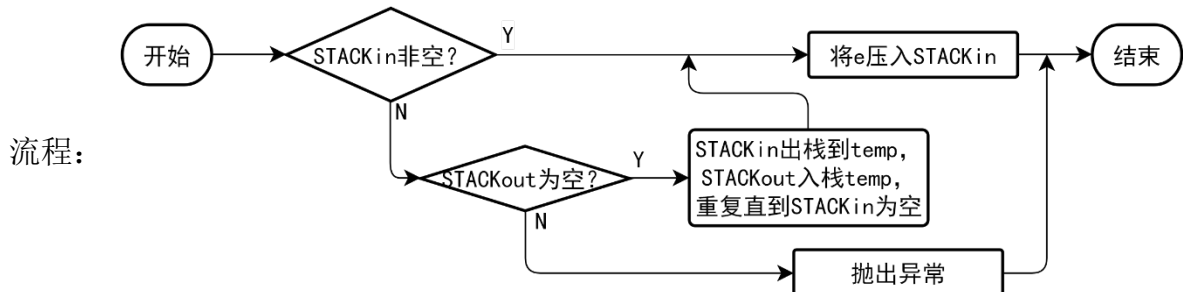
返回: 队列本身

说明: 重载<<运算符, 将 `e` 插入队列的尾部, 并返回队列本身。在尝试入队列时, `STACKin` 和 `STAK1` 只可能处于如下 3 种状态:

- 1) `STACKin` 未滿
- 2) `STACKin` 滿且 `STACKout` 空
- 3) `STACKin` 滿且 `STACKout` 非空

第一种情况下可以直接将元素压入 `STACKin`。第二种情况下可先将 `STACKin` 中的元素依次全部转移到 `STACKout` 中, 然后将元素压入 `STACKin`。第三种情况下队列将被判断为滿的, 不能入栈。

异常: 当队列滿时将抛出 `logic_error` 异常。



➤ `QUEUE &QUEUE::operator>>(int &e)`

参数: `e`: 弹出队列的元素

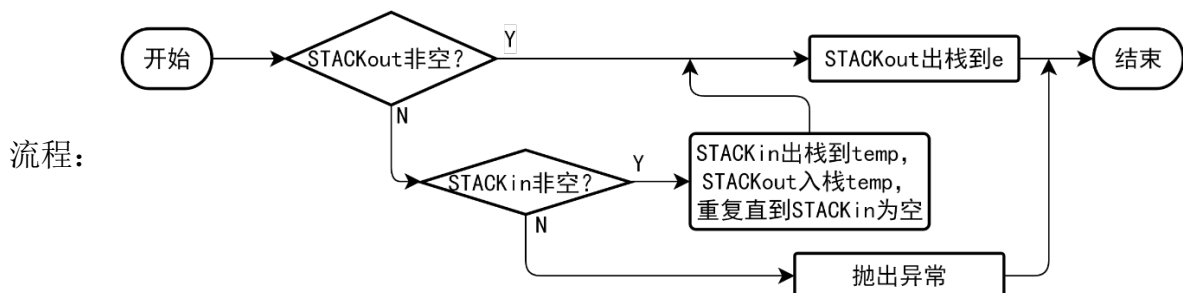
返回: 队列本身

说明: 重载>>运算符, 从队列首弹出一个元素并赋值给 `e`, 然后返回队列本身。在尝试出队列时, `STACKin` 和 `STACKout` 只可能处于如下 3 种状态:

- 1) `STACKout` 非空
- 2) `STACKout` 空且 `STACKin` 非空
- 3) `STACKout` 空且 `STACKin` 空

第一种情况下可以直接从 `STACKout` 中弹出元素。第二种情况下可先将 `STACKin` 中的元素依次全部转移到 `STACKout` 中, 然后从 `STACKout` 中弹出元素。第三种情况下队列将被判断为空, 不能弹出。

异常: 当队列为空时将抛出 `logic_error` 异常。

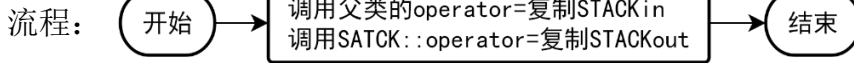


➤ `QUEUE &QUEUE::operator=(const QUEUE &s)`

参数: `s`:被复制的队列

返回: 队列本身

说明: 重载`=`运算符, 将本队列的大小更改为与队列 `s` 的大小相同, 然后将队列 `s` 的内容赋值给本队列, 并返回本队列。直接调用 `STACK::operator=()` 将 `STACKin` 和 `STACKout` 分别复制一份即可。



➤ `int QUEUE::operator[](int x) const`

参数: `x`:要取出的元素的下标

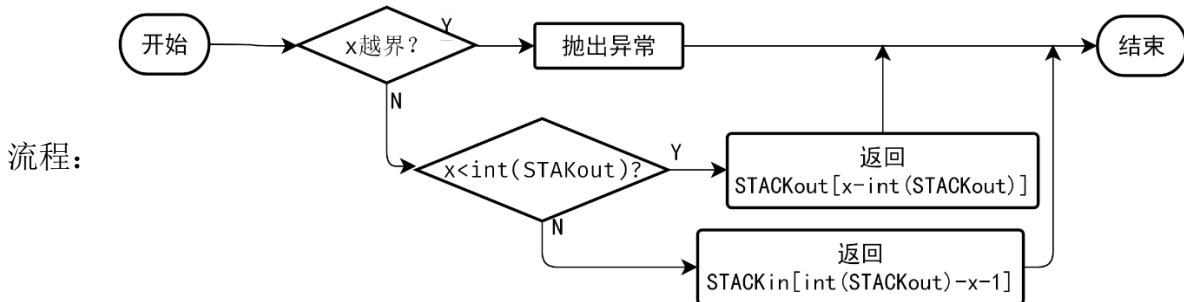
返回: 队列中下标为 `x` 的元素值

说明: 重载`[]`运算符, 取出从队列首开始计数的下标为 `x` 的元素值并将其返回。由于队列是由两个栈实现的, 因此不能够直接从栈的数组中取元素, 而要进行计算。尝试取元素时, 有 3 种可能:

- 1) `x` 小于 `STACKout` 中元素的个数
- 2) `x` 大于或等于 `STACKout` 中元素的个数
- 3) `x` 越界(`x < 0` 或 `x ≥ 队列中的元素个数`)

由于队列首是从 `STACKout` 开始计算的, 因此在第一种情况下, 要取出的元素在 `STACKout` 中, 直接从 `STACKout` 中取出第 `x-int(STACKout)` 个元素即可。第二种情况下要取出的元素在 `STACKin` 中, 需要取出 `STACKin` 中的第 `int(STACKout)-x-1` 个元素。第三种情况直接抛出异常。

异常: 当 `x` 越界(`x < 0` 或 `x ≥ 队列中的元素个数`)时抛出 `std::logic_error` 异常。

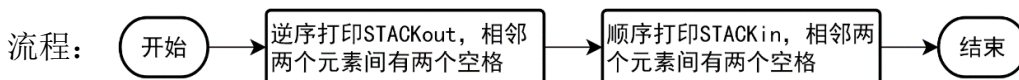


➤ `void QUEUE::print() const`

参数: 无

返回: 无

说明: 从队列首遍历到队列尾, 并打印每一个存放的元素, 每两个元素间有两个空格。由于队列首位于 `STACKout` 的栈顶, 而队列尾位于 `STACKin` 的栈底, 因此从 `s2` 开始反向遍历, 完成后正向遍历 `s1` 中的元素。



Utils 部分

Utils 命名空间提供了两个工具函数: `convertToInt` 和 `isPrime`, 供 `main` 函数使用。

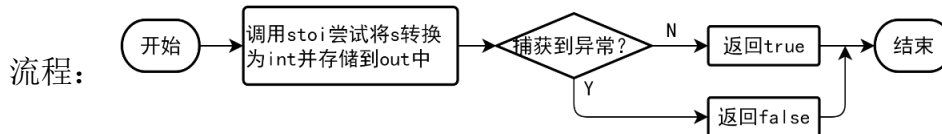
➤ `bool Utils::convertToInt(const std::string &s, const &out)`

参数: s:输入字符串

out:转换结果

返回: 转换是否成功, 成功则返回 true, 否则返回 false

说明: 此函数用于命令行处理, 尝试将字符串 s 的内容转换为整型并存储于 out 中。调用标准库的 stoi 进行实现并尝试捕获 invalid_argument 和 out_of_range 异常。若捕获到异常则说明转换出现错误, 返回 false, 否则返回 true。

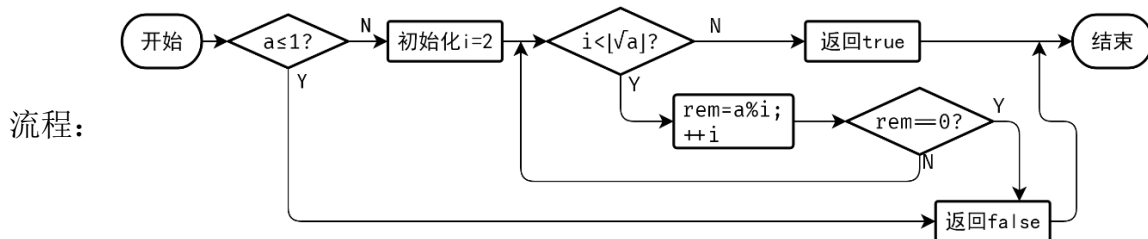


➤ bool Utils::isPrime(const int a)

参数: a:用于判断的数

返回: a 是否为质数, 是则返回 true, 否则返回 false

说明: 此函数用于判断输入是否为质数, 通过计算 a 对从 2 到 \sqrt{a} 每一个数的模来判断 a 是否为质数。



三、软件开发

开发平台: Archlinux x86-64

测试平台: Archlinux x86-64, Mingw64(Windows)

文本编辑工具: vim 版本 8.0

编译链接工具: gcc 版本 7.2.0 (Mingw64 下 gcc6.3.0)

逻辑调试工具: cgdb 版本 0.7.0

内存分析工具: valgrind 版本 3.13.0 (Mingw64 下不存在)

文档生成工具: doxygen 版本 1.8.13

Makefile 处理工具: GNU Make 版本 4.2.1

编译参数: -std=c++14 -Wall -Werror -static -static-libstdc++

编译命令: g++ U201514898_6.cpp \${CXXFLAGS} -o U201514898_6.exe

编译命令中 \${CXXFLAGS} 为编译参数

四、软件测试

自动测试部分

这一部分测试当 `argc!=1` 时的程序功能。包括了对于每一个类方法的正确性测试。通过修改命令行参数并比较程序运行结果和标准程序运行结果来判断程序是否运行正确。为了节省篇幅, 直接将如图 5 所示的测试结果的输出部分复制到输出部分, 而不再单独给出截图。



图 5 输出界面样例

➤ 设定队列大小测试

输入: -S 1

用户输出: S 1

标准输出: S 1

输入: -S 100

用户输出: S 100

标准输出: S 100

输入: -S 0

用户输出: S 0

标准输出: S E

可以看出，在输入大于 0 的情况下程序正确地创建了指定大小的栈，并正确地进行了输出。而当 S 后面为 0 的时候程序输出 S 0，标准程序输出 S E。由于题目要求中没有规定队列大小为 0 时程序的行为，也没有规定不允许出现大小为 0 的队列，因此认为是未定义行为，这种边界情况不予考虑。

➤ 入队列测试

输入: -S 3 -I 1 2 3

用户输出: S 3 I 1 2 3

标准输出: S 3 I 1 2 3

输入: -S 3 -I 1 2 3 4 5 6 7

用户输出: S 3 I E

标准输出: S 3 I E

可以看出，对于正确的插入和错误的插入程序都能够正确的处理，插入完成则输出队列中的元素，否则输出 E。

➤ 出队列测试

输入：-S 3 -O 1

用户输出：S 3 O E

标准输出：S 3 O E

输入：-S 3 -I 1 2 -O 2

用户输出：S 3 I 1 2 O

标准输出：S 3 I 1 2 O

输入：-S 3 -I 1 2 -O 3

用户输出：S 3 I 1 2 O E

标准输出：S 3 I 1 2 O E

对于出队列而言，若队列是空的，那么程序能够正确输出 E，否则程序将-O 后所接数字个数的元素出栈，输出结果表明程序能够正常运行。

➤ 深拷贝构造测试

输入：-S 1 -C

用户输出：S 1 C

标准输出：S 1 C

输入：-S 3 -I 1 2 -C

用户输出：S 3 I 1 2 C 1 2

标准输出：S 3 I 1 2 C 1 2

从输出可以看出，无论队列中是否有元素深拷贝构造都能够正确地复制构造整个队列。

➤ 深拷贝赋值测试

输入：-S 3 -I 1 2 3 -A 6

用户输出：S 3 I 1 2 3 A 1 2 3

标准输出：S 3 I 1 2 3 A 1 2 3

输入：4 -I 1 2 3 -A 2

用户输出：S 4 I 1 2 3 A 1 2 3

标准输出：S 4 I 1 2 3 A 1 2 3

从输出可以看出，无论-A 所指定的参数是大于原队列大小还是小于原队列大小，程序表现都与标准程序一致。

➤ 随机取值测试

输入：-S 4 -I 1 2 3 4 -G 1

用户输出: S 4 I 1 2 3 4 G 2
标准输出: S 4 I 1 2 3 4 G 2

输入: -S 4 -I 1 2 3 4 -G 7
用户输出: S 4 I 1 2 3 4 G E
标准输出: S 4 I 1 2 3 4 G E

对于范围在数组存储元素个数范围内的取值而言,程序能够正确地取出下标为-G 参数后的数字所指定的下标的元素。在越界情况下能够正确输出 E。

➤ 大小显示测试

输入: -S 4 -I 1 2 -N
用户输出: S 4 I 1 2 N 2
标准输出: S 4 I 1 2 N 2

输入: -S 6 -N
用户输出: S 6 N 0
标准输出: S 6 N 0

从输出可以看出,当程序参数带有-N 时,程序会正确的打印队列现存元素的大小。

➤ 综合测试

输入: -S 3 -I 1 2 3 -O 2 -G 2 -A 10 -C -N
用户输出: S 3 I 1 2 3 0 3 G E
标准输出: S 3 I 1 2 3 0 3 G E

输入: -S 3 -I 1 2 3 -O 1 -G 1 -A 1 -C -N
用户输出: S 3 I 1 2 3 0 2 3 G 3 A 2 3 C 2 3 N 2
标准输出: S 3 I 1 2 3 0 2 3 G 3 A 2 3 C 2 3 N 2

输入: -S 14 -I 1 2 3 4 5 -O 1 -G 1 -A 3 -C -C
用户输出: S 14 I 1 2 3 4 5 0 2 3 4 5 G 3 A 2 3 4 5 C 2 3 4 5 C 2
3 4 5
标准输出: S 14 I 1 2 3 4 5 0 2 3 4 5 G 3 A 2 3 4 5 C 2 3 4 5 C 2
3 4 5

对于连续的输入,程序能够正确的处理并没有返回错误,至此自动测试部分完成,程序的功能全部实现且没有错误出现。

舞伴问题部分

这一部分测试当 `argc==1` 时程序的功能,即对于舞伴问题解决程序的测试。直接在命终端(此次测试为在 `MSYS2 mingw64` 终端)下运行 `./U201514898_6.exe`,然后输入进行测试。测试包含了程序的正确性测试和健壮性测试:在输入非法时,应该提示用户重新输入数据并继续执行程序。输入正确时,程序的输出结果应该与理论值相比较以保证其

正确性。理论值为 $\min(\{x \mid x \equiv m \pmod{M} \text{ and } x \equiv f \pmod{F}\})$ 。

➤ M=2 F=3 m=1 f=1

理论值: 1

运算结果: 1

```
fish /home/HuSixu/Project/CppExperiment
HuSixu@Ruby-Win10 ~/P/CppExperiment (master *)
> ./U201514898_6.exe
Please enter the number of gentlemen: 2
Please enter the number of laydies: 3
Please enter m (1 <= m <= number of gentlemen): 1
Please enter f (1 <= f <= number of ladies): 1
==> m and f met at round 1
HuSixu@Ruby-Win10 ~/P/CppExperiment (master *)
>
```

➤ M=2 F=3 m=2 f=3

理论值: 6

运算结果: 6

```
fish /home/HuSixu/Project/CppExperiment
HuSixu@Ruby-Win10 ~/P/CppExperiment (master *)
> ./U201514898_6.exe
Please enter the number of gentlemen: 2
Please enter the number of laydies: 3
Please enter m (1 <= m <= number of gentlemen): 2
Please enter f (1 <= f <= number of ladies): 3
==> m and f met at round 6
HuSixu@Ruby-Win10 ~/P/CppExperiment (master *)
>
```

➤ M=1009 N=1013 m=512 f=513

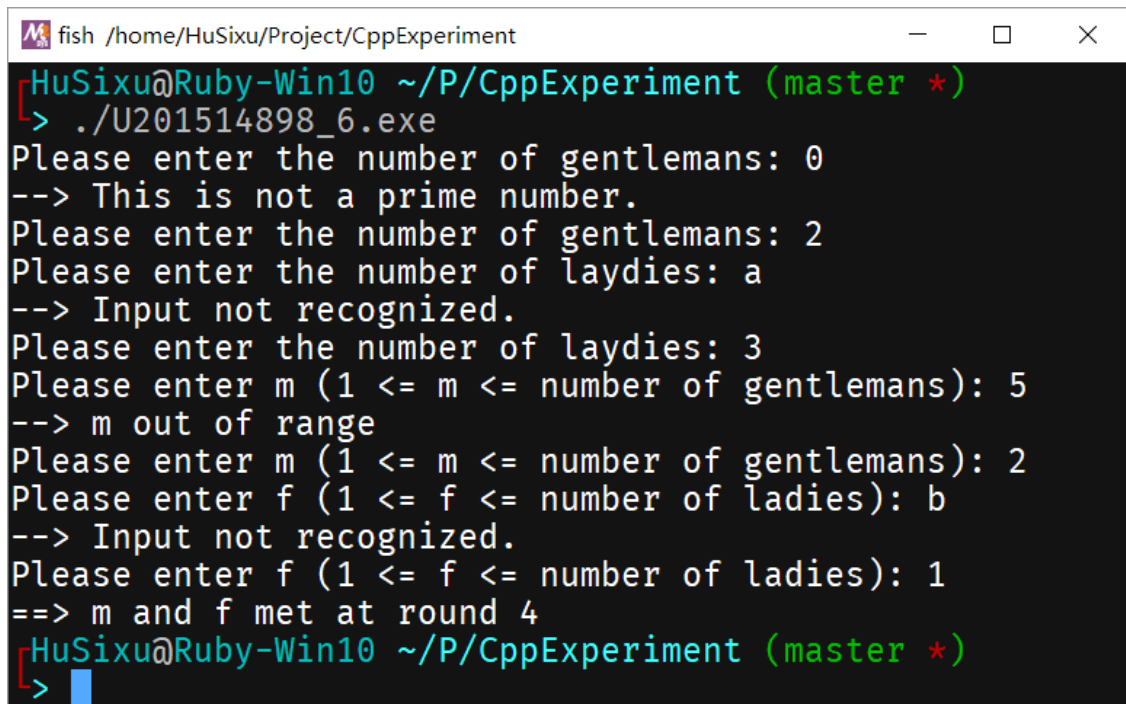
理论值: 255789

运算结果: 255789

```
fish /home/HuSixu/Project/CppExperiment
HuSixu@Ruby-Win10 ~/P/CppExperiment (master *)
> ./U201514898_6.exe
Please enter the number of gentlemen: 1009
Please enter the number of laydies: 1013
Please enter m (1 <= m <= number of gentlemen): 512
Please enter f (1 <= f <= number of ladies): 513
==> m and f met at round 255789
HuSixu@Ruby-Win10 ~/P/CppExperiment (master *)
>
```

➤ 错误输入测试

尝试输入非质数的人数，越界的 m 和 f ，以及非数（字母），并观察程序的行为。从下图可以看出，对于不合法的输出，程序提示出错以及错误的原因，并要求重新输入，这与期望的结果一致。



```
fish /home/HuSixu/Project/CppExperiment
HuSixu@Ruby-Win10 ~/P/CppExperiment (master *)
> ./U201514898_6.exe
Please enter the number of gentlemen: 0
--> This is not a prime number.
Please enter the number of gentlemen: 2
Please enter the number of laydies: a
--> Input not recognized.
Please enter the number of laydies: 3
Please enter m (1 <= m <= number of gentlemen): 5
--> m out of range
Please enter m (1 <= m <= number of gentlemen): 2
Please enter f (1 <= f <= number of ladies): b
--> Input not recognized.
Please enter f (1 <= f <= number of ladies): 1
==> m and f met at round 4
HuSixu@Ruby-Win10 ~/P/CppExperiment (master *)
>
```

至此所有测试完成。程序功能的正确性和程序的健壮性都得到了验证，程序的行为与与预想的一致。

五、特点与不足

1. 技术特点

- 文档全面。使用了 doxyblocks 使用注释在接口部分添加了文档，可以直接使用 doxygen 生成文档以供其他的贡献者和使用者查询。
- 提供 Makefile，编译简便。
- 未使用平台依赖的库，程序具有较高的跨平台性。
- 自行编写了一个新的测试系统¹以弥补现存测试系统的不足。

2. 不足和改进的建议

- 命令行解析不健壮。本程序对于部分特殊的命令行处理会输出与标准程序不一样的错误结果。可以考虑使用 getopt 等库进行参数解析。
- 线程不安全。若要做到线程安全需要在部分内存操作的地方加锁。
- 功能不完备。对于栈和队列这种线性容器可以考虑加入迭代器。
- 自动测试样例覆盖不全面，不能保证程序的健壮性。可考虑使用 Google Test 或者 Qt Test 来编写单元测试以在整个程序的开发过程中实现全面而广泛的自动化测试。

¹ <https://gitlab.com/husixu1/HUSTCppExperimentCheckSystemPlusPlus>

六、过程和体会

1. 遇到的主要问题和解决方法

在实现 `STACK::operator=` 时遇到了 `max` 为 `const` 成员不可更改的问题。在查阅资料后使用 `const_cast` 构造了一个非 `const` 指针成功的修改了 `const` 成员变量的值。在实现队列的过程中由于需要调用两个栈的重载运算符造成了一定的混淆，通过 `gdb` 调试定位后成功的为继承而来的栈和持有的栈分别调用了正确的重载运算符函数。

2. 课程设计的体会

通过这次课程设计，我更巩固并加深了对于 C++ 部分知识的理解，包括继承和多态，`const` 成员的修改，重载运算符，标准异常库等等，并成功的进行了实践。此外，还对于以前并不熟悉的命令行解析进行了深入的探究与实践。六次渐进性的实验也让我更深入的了解了 C++ 与 C 之间、面向对象与面向过程之间的联系与区别，为今后的学习与实践生活打下了基础。

七、源码和说明

1. 文件清单及其功能说明

`U201514898_6.cpp`: 源代码，主函数、文档、所有的类定义和实现都在这个文件中。

`U201514898_6.exe`: 可执行文件，需在 64 位 Windows 平台上执行，全静态链接。

`Makefile`: 使用 GNU Make 或者 NMake 可快速编译。

`Doxyfile`: doxygen 配置文件，用于生成文档。

2. 用户使用说明书

编译方法:

Windows 平台:

1. 直接使用二进制文件。
2. `cmd` 下使用 `g++ U201514898_6.cpp -std=c++14 -O3 -o U201514898.exe` 进行编译（需要确保 `g++` 在系统 `PATH` 中并且能够正确找到标准 `c++` 动态链接库）。
3. 使用 `MSYS2` 平台安装 `mingw toolchain`。

Linux 平台:

1. 在确保安装了 `gnu toolchain` 后（至少需要包含 `gcc`, `make`）在 `Makefile` 所在的文件夹下运行 `make` 命令。
2. 直接使用 `g++ U201514898_6.cpp -std=c++14 -O3 -o U201514898.exe` 进行编译。（同样需要 `gnu toolchain`）

运行方法:

1. 在终端/命令行下直接运行可执行文件，加参数则为自动测试，不加参数则为解决舞伴问题。
2. 使用调试工具（如 `gdb`）运行调试。
3. 使用测试系统运行。

3. 源代码

```

/**
 * @file U201514898_6.cpp
 * @brief object-oriented queue implementation, two stack inherited version.
 *
 * @details
 * This file implemented a int queue using two stacks, inheriting one of them
 * and some overloaded operators to operate it,
 * developed using object oriented method. A main function is
 * added to test the correctness and robustness of the stack/queue.
 *
 * @date 2017-11-07
 * @author Sixu Hu
 *
 * @copyright
 * Copyright 2017 Sixu Hu <husixu1@hotmail.com>
 * <br><br>
 * This file is part of C++ Experiment.
 * <br><br>
 * C++ Experiment free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <br><br>
 * C++ Experiment is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with C++ Experiment. If not, see <http://www.gnu.org/licenses/>.
 **/

#include<stdexcept>
#include<iostream>
#include<fstream>
#include<limits>

/// @brief the stack class
class STACK {
private:
    /// @brief the array used to store stack elements
    int *const elems;
    /// @brief stack capacity

```

```

const int max;
/// @brief stack size
int pos;

public:
    /// @brief constructor, initialize the stack with the size of m
    /// @param[in] m the size of the stack
    STACK(int m);

    /// @brief copy-constructor, initialize the stack with another stack
    /// @param[in] s the stack to copy
    STACK(const STACK &s);

    /// @brief get the capacity of the stack
    /// @return stack capacity
    virtual int size() const;

    /// @brief get the size of the stack
    /// @return stack size
    virtual operator int() const;

    /// @brief get the element with index x
    /// @param[in] x the element index to get
    /// @return the element with index x
    virtual int operator [](int x) const;

    /// @brief push an element to the stack
    /// @param[in] e the element to push
    /// @return the stack after pushing
    virtual STACK &operator <<(int e);

    /// @brief pop an element from the stack
    /// @param[out] e the popped element
    /// @return the stack after popping
    virtual STACK &operator >>(int &e);

    /// @brief copy a stack to this stack
    /// @param s the stack to copy
    /// @return the copied stack
    /// @note the current stack's element will be deleted
    virtual STACK &operator =(const STACK &s);

    /// @brief print the elements of the thread
    virtual void print() const;

```

```

    /// @brief de-structor, destroy the stack
    virtual ~STACK();
};

namespace Utils {
    /// @brief convert string to int
    /// @param[in] s the string to convert
    /// @param[out] out the output int
    /// @return if the conversion is successful
    bool convertToInt(const std::string &s, int &out);

    /// @brief judge if a int is a prime number
    /// @param[in] a the int to judge
    /// @return is a a prime number
    bool isPrime(const int a);
}

/// @brief the queue class, using two stack
class QUEUE : public STACK {
private:
    /// @brief the two stack used to construct the queue
    STACK s2;

public:
    /// @brief constructor, initialize the queue with the size of m
    /// @param[in] m the size of the queue
    QUEUE(int m);

    /// @brief copy-constructor, initialize the queue with another queue
    /// @param[in] s the queue to copy
    QUEUE(const QUEUE &s);

    /// @brief get the capacity of the queue
    /// @return queue capacity
    virtual operator int() const;

    /// @brief get the size of the queue
    /// @return queue size
    virtual int full() const;

    /// @brief get the element with index x
    /// @param[in] x the element index to get
    /// @return the element with index x

```

```

virtual int operator[](int x)const;

/// @brief push an element to the queue
/// @param[in] e the element to push
/// @return the queue after pushing
virtual QUEUE &operator<<(int e);

/// @brief pop an element from the queue
/// @param[out] e the popped element
/// @return the queue after popping
virtual QUEUE &operator>>(int &e);

/// @brief copy a queue to this queue
/// @param s the queue to copy
/// @return the copied queue
/// @note the current queue's element will be deleted
virtual QUEUE &operator=(const QUEUE &s);

/// @brief print the elements of the thread
virtual void print() const;

/// @brief de-structor, destroy the queue
virtual ~QUEUE();
};

int main(int argc, char *argv[]) {
    using namespace std;
    using namespace Utils;

    // if argc is 1, run dancer problem, else run tests
    if (argc == 1) {
        int menNum, womenNum, m, f;
        //> get men number
        while (true) {
            cout << "Please enter the number of gentlemen: " << flush;
            cin >> menNum;
            if (cin.eof() || cin.bad()) {
                continue;
            } else if (cin.fail()) {
                cout << "--> Input not recognized." << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
            } else if (!isPrime(menNum)) {
                cout << "--> This is not a prime number." << endl;
            }
        }
    }
}

```

```

        } else
            break;
    }
    //> get women number
    while (true) {
        cout << "Please enter the number of laydies: " << flush;
        cin >> womenNum;
        if (cin.eof() || cin.bad()) {
            continue;
        } else if (cin.fail()) {
            cout << "--> Input not recognized." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        } else if (!isPrime(womenNum)) {
            cout << "--> This is not a prime number." << endl;
        } else if (womenNum == menNum) {
            cout << "--> The number of laydies must be unequal with the
number of gentlemans." << endl;
        } else
            break;
    }
    //> get m
    while (true) {
        cout << "Please enter m (1 <= m <= number of gentlemans): " <<
flush;
        cin >> m;
        if (cin.eof() || cin.bad()) {
            continue;
        } else if (cin.fail()) {
            cout << "--> Input not recognized." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        } else if (m <= 0 || m > menNum) {
            cout << "--> m out of range" << endl;
        } else
            break;
    }
    //> get f
    while (true) {
        cout << "Please enter f (1 <= f <= number of ladies): " << flush;
        cin >> f;
        if (cin.eof() || cin.bad()) {
            continue;
        } else if (cin.fail()) {

```

```

        cout << "--> Input not recognized." << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    } else if (f <= 0 || f > womenNum) {
        cout << "--> f out of range" << endl;
    } else
        break;
}

// create QUEUE
QUEUE queueMen(2 * menNum + 1), queueWomen(2 * womenNum + 1);
for (int i = 0; i < menNum; ++i)
    queueMen << i + 1;
for (int i = 0; i < womenNum; ++i)
    queueWomen << i + 1;

int currentMan, currentWoman, counter = 1;
while (1) {
    queueMen >> currentMan;
    queueWomen >> currentWoman;
    if (currentMan == m && currentWoman == f) {
        cout << "=> m and f met at round " << counter << endl;
        break;
    }
    queueMen << currentMan;
    queueWomen << currentWoman;
    ++counter;
}
return 0;
} else {
    // arguments
    string arg;
    int number;

    // queue for test
    QUEUE *queue = nullptr ;

    // file to write
    ofstream outFile;
    outFile.open("U201514898_6.TXT");

    auto coutBackup = cout.rdbuf();
    cout.rdbuf(outFile.rdbuf());

```

面向对象程序设计实验报告

```
#define CHECK_IF_LAST_ARG if (i == argc - 1){ \
    cerr << "paramter not found after: " << arg << endl; \
    return 1; \
}

try {
    for (int i = 1; i < argc; ++i) {
        arg = string(argv[i]);
        if (arg == "-S" || arg == "-s") {
            CHECK_IF_LAST_ARG;

            arg = string(argv[++i]);
            if (!convertToInt(arg, number)) {
                cerr << "Error: cannot convert " << arg << " to int" <<
endl;

                return 1;
            }

            queue = new QUEUE(number);
            cout << "S  " << number;
        } else if (arg == "-I" || arg == "-i") {
            cout << "I";

            for (++i; i < argc; ++i) {
                arg = string(argv[i]);
                // until there's noting to covert
                if (!convertToInt(arg, number)) {
                    break;
                }
                (*queue) << number;
            }
            if (i != argc)
                --i;

            if (int(*queue)) {
                cout << "  ";
                queue->print();
            }
        } else if (arg == "-O" || arg == "-o") {
            CHECK_IF_LAST_ARG

            cout << "O";
            arg = string(argv[++i]);
            if (!convertToInt(arg, number)) {
```



```

        cerr << "Error: cannot convert " << arg << " to int" <<
endl;

        return 1;
    }

    int garbage;
    for (int j = 0; j < number; ++j) {
        (*queue) >> garbage;
    }

    if (int(*queue)) {
        cout << " ";
        queue->print();
    }

} else if (arg == "-C" || arg == "-c") {
    QUEUE *newQueue = new QUEUE(*queue);
    queue = newQueue;

    cout << "C";
    if (int(*queue)) {
        cout << " ";
        queue->print();
    }
} else if (arg == "-A" || arg == "-a") {
    CHECK_IF_LAST_ARG

    arg = string(argv[++i]);
    if (! convertToInt(arg, number)) {
        cerr << "Error: cannot convert " << arg << " to int" <<
endl;

        return 1;
    }

    QUEUE *newQueue = new QUEUE(number);
    *newQueue = *queue;
    queue = newQueue;

    cout << "A";
    if (int(*queue)) {
        cout << " ";
        queue->print();
    }
}

```

```

        } else if (arg == "-N" || arg == "-n") {
            cout << "N  " << int(*queue);

            } else if (arg == "-G" || arg == "-g") {
                CHECK_IF_LAST_ARG

                arg = string(argv[++i]);
                if (! convertToInt(arg, number)) {
                    cerr << "Error: cannot convert " << arg << " to int" <<
endl;

                    return 1;
                }

                cout << "G  " << (*queue)[number];

            } else {
                cerr << "Wrong parameter: " << arg << endl;
                return 1;
            }

            if (i != argc - 1)
                cout << "  ";
        }
    } catch (logic_error error) {
        cout << "  E" << endl;
        return 1;
    }
    cout.rdbuf(coutBackup);
    outFile.close();
    return 0;
}

// STACK implementation
STACK::STACK(int m) : elems(new int [m]), max(m) {
    pos = 0;
}

STACK::STACK(const STACK &s) : elems(new int[s.max]), max(s.max) {
    pos = s.pos;
    for (int i = 0; i < pos; ++i)
        elems[i] = s.elems[i];
}

```

```

int STACK::size() const {
    return max;
}

STACK::operator int() const {
    return pos;
}

int STACK::operator [](int x) const {
    if (x >= pos)
        throw std::logic_error("array index out of range");
    return elems[x];
}

STACK &STACK::operator <<(int e) {
    if (pos == max)
        throw std::logic_error("array is full, cannot push");
    elems[pos] = e;
    ++pos;
    return *this;
}

STACK &STACK::operator >>(int &e) {
    if (pos == 0)
        throw std::logic_error("pop from empty stack");
    e = elems[--pos];
    return *this;
}

STACK &STACK::operator =(const STACK &s) {
    // Do never use this in your code, this is a horrible example
    int *ref = const_cast<int *>(elems);
    delete ref;
    ref = new int [s.max];
    int *num = const_cast<int *>(&max);
    *num = s.max;
    pos = s.pos;

    for (int i = 0; i < pos; ++i)
        elems[i] = s.elems[i];
    return *this;
}

void STACK::print() const {

```

```

using namespace std;
for (int i = 0; i < pos; ++i) {
    if (i != pos - 1)
        cout << elems[i] << " ";
    else
        cout << elems[i];
}
cout << flush;
}

STACK::~STACK() {
    delete []elems;
}

// QUEUE implementation
QUEUE::QUEUE(int m) : STACK(m), s2(m) {
    return;
}

QUEUE::QUEUE(const QUEUE &s) : STACK(static_cast<STACK>(s)), s2(s.s2) {
    return;
}

QUEUE::operator int() const {
    return STACK::operator int() + s2.operator int();
}

int QUEUE::full() const {
    return STACK::operator int() == STACK::size() && !s2.operator int();
}

int QUEUE::operator [](int x) const {
    if (x >= STACK::operator int() + s2.operator int())
        throw std::logic_error("trying to get elem outside of queue");

    // judge where is the element
    if (x >= int(s2))
        return STACK::operator [](x - int(s2));
    else
        return s2[int(s2) - x - 1];
}

QUEUE &QUEUE::operator <<(int e) {
    // judge if s1 is full

```

```

    if (STACK::operator int() == STACK::size()) {
        int temp;
        int origSize = STACK::operator int();
        for (int i = 0; i < origSize; ++i) {
            STACK::operator >>(temp);
            s2 << temp;
        }
    }
    // push into stack
    STACK::operator <<(e);
    return *this;
}

QUEUE &QUEUE::operator >>(int &e) {
    if (int(s2)) {
        s2 >> e;
    } else {
        int temp;
        int origSize = STACK::operator int();
        for (int i = 0; i < origSize - 1; ++i) {
            STACK::operator >>(temp);
            s2 << temp;
        }
        STACK::operator >>(e);
    }
    return *this;
}

QUEUE &QUEUE::operator =(const QUEUE &s) {
    STACK::operator=(static_cast<STACK>(s));
    s2 = s.s2;
    return *this;
}

void QUEUE::print() const {
    using namespace std;

    // print elements in s2
    for (int i = 0; i < int(s2); ++i) {
        if (i)
            cout << " ";
        cout << s2[int(s2) - i - 1];
    }
}

```

```

if (int(s2) && STACK::operator int())
    cout << " ";

// print elements in s1
for (int i = 0; i < STACK::operator int(); ++i) {
    if (i)
        cout << " ";
    cout << STACK::operator [](i);
}

cout << flush;
}

QUEUE::~~QUEUE() {
    return;
}

bool Utils::convertToInt(const std::string &s, int &out) {
    using namespace std;

    try {
        out = stoi(s);
    } catch (invalid_argument i) {
        return false;
    } catch (out_of_range o) {
        return false;
    }
    return true;
}

bool Utils::isPrime(const int a) {
    if (a <= 1)
        return false;

    int i;
    for (i = 2; i * i <= a; i++)
        if (a % i == 0)
            return false;
    return true;
}

```