

华中科技大学

# 课程实验报告

课程名称：C++程序设计

实验名称：面向对象的整型队列编程

院 系：计算机科学与技术学院

专业班级：计算机科学与技术 2001 班

学 号：U202011641

姓 名：刘景宇

指导教师：马光志

2021 年 12 月 12 日

## 一、需求分析

### 1. 题目要求

整型队列是一种先进先出的存储结构，对其进行的操作通常包括：向队列尾部添加一个整型元素、从队列首部移除一个整型元素等。整型循环队列类 **QUEUE** 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如下函数采用 **C++** 语言编程，然后写一个 **main** 函数对队列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。

```
#define _CRT_SECURE_NO_WARNINGS
#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
}
```

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) 用 **QUEUE(int m)** 对队列初始化时，为其 **elems** 分配 **m** 个整型元素内存，并初始化 **max** 为 **m**，以及初始化 **head=tail=0**。

(2) 对于 **QUEUE(const QUEUE& q)** 深拷贝构造函数，在用已经存在的对象 **q** 深拷贝构造新对象时，新对象不能共用已经存在的对象 **q** 的 **elems** 内存，新对象的 **elems** 需要分配和 **q** 为 **elems** 分配的同样大小的内存，并且将 **q** 的 **elems** 的内容深拷贝至新对象分配的内存；新对象的 **max**、**head**、**tail** 应设置成和 **q** 的对应值相同。

(3) 对于 **QUEUE(QUEUE&& q)** 移动构造函数，在用已经存在的对象 **q** 移动构造新对象时，新对象接受使用对象 **q** 为 **elems** 分配的内存，并且新对象的 **max**、**head**、**tail** 应设置成和对象 **q** 的对应值相同；

然后对象 **q** 的 **elems** 设置为空指针以表示内存被移走，同时其 **max**、**head**、**tail** 均应设置为 0。

(4) 对于 **QUEUE& operator=(const QUEUE& q)** 深拷贝赋值函数，在用等号右边的对象 **q** 深拷贝赋值等号左边的对象时，等号左边的对象若为 **elems** 分配了内存，则应先释放内存以避免内存泄漏。等号左边的对象不能共用对象 **q** 的 **elems** 的同一块内存，应为其 **elems** 分配和 **q** 为 **elems** 分配的同样大小的内存，并且将 **q** 的 **elems** 存储的内容拷贝至等号左边对象分配的内存；等号左边对象的 **max**、**head**、**tail** 应设置成和 **q** 的对应值相同。

(5) 对于 **QUEUE& operator=(QUEUE&& q)noexcept** 移动赋值函数，在用已经存在的对象 **q** 移动赋值给等号左边的对象时，等号左边的对象若为 **elems** 分配了内存，则应先释放内存以避免内存泄漏。等号左边的对象接受使用 **q** 为 **elems** 分配的内存，并且等号左边的对象的 **max**、**head**、**tail** 应设置成和对象 **q** 的对应值相同；对象 **q** 的 **elems** 然后设置为空指针以表示内存被移走，同时其 **max**、**head**、**tail** 均应置为 0。

(6) 队列应实现为循环队列，当队尾指针 **tail** 快要追上队首指针 **head** 时，即如果满足  $(tail+1)\%max=head$ ，则表示表示队列已满，故队列最多存放 **max-1** 个元素；而当 **head=tail** 时，则表示队列为空。队列空取出元素或队列满放入元素均应抛出异常，并且保持其内部状态不变。

(7) 打印队列时从队首打印至队尾，打印的元素之间以逗号分隔。

## 2. 需求分析

队列在日常生活中十分常见，例如：银行排队办理业务、食堂排队打饭等等，这些都是队列的应用。排队的原则就是先来后到，排在前面的人就可以优先办理业务，那么队列也一样，队列遵循先进先出的原则。

这里要求对整型队列进行 C++ 实现，实现队列的先进先出，并且将其功能进行模块化，编写成函数，为了避免用户异常调用函数，函数参数多采用常引用。采取面向对象的整型队列编程，初始化采用构造函数完成，销毁队列采用析构函数完成。初始化队列时可以是生成空队列，也可以是深拷贝一份，或者浅拷贝一份，其中对于浅拷贝得到的队列，要在释放队列前判断空间是否已经释放。能够实现队列的复制，并且可以完成从尾部入队、从首部出队等基本操作，同时能够对于空队列时出队、满队列时入队等不合法操作抛出异常，作为提示，能够随时查看队列能存放的最大整数个数以及当前存放整数个数。

## 二、系统设计

### 1. 概要设计

面向对象的整型队列，能够很好的实现模块化以及代码的复用性。

设计相应的数据结构，包括存放队列元素的部分、最大元素，以及队列的头和尾的序号。其中 **elems** 为指针，指向 **Queue** 队列对应的首地址，**max** 为整型常量，存储 **QUEUE** 中的最大存储元素的个数，整型变量 **head** 和 **tail** 分别存放队列中首元素和尾元素在队列中的序号位置。（如图 1 所示）

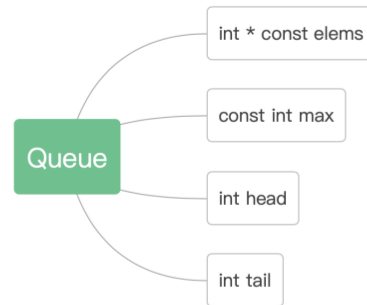


图 1 QUEUE 数据结构

将队列功能的实现进行函数化，包括利用构造函数初始化队列，通过重载等 C++ 特性实现通过操作符返回实际元素个数、最大元素、进队、出队、队列的复制等功能，通过 `print` 成员函数实现队列元素的输出显示等，通过析构函数实现队列的销毁，同时将其普通成员函数声明为虚函数，便于派生出其他类，实现多态。（如图 2 所示）

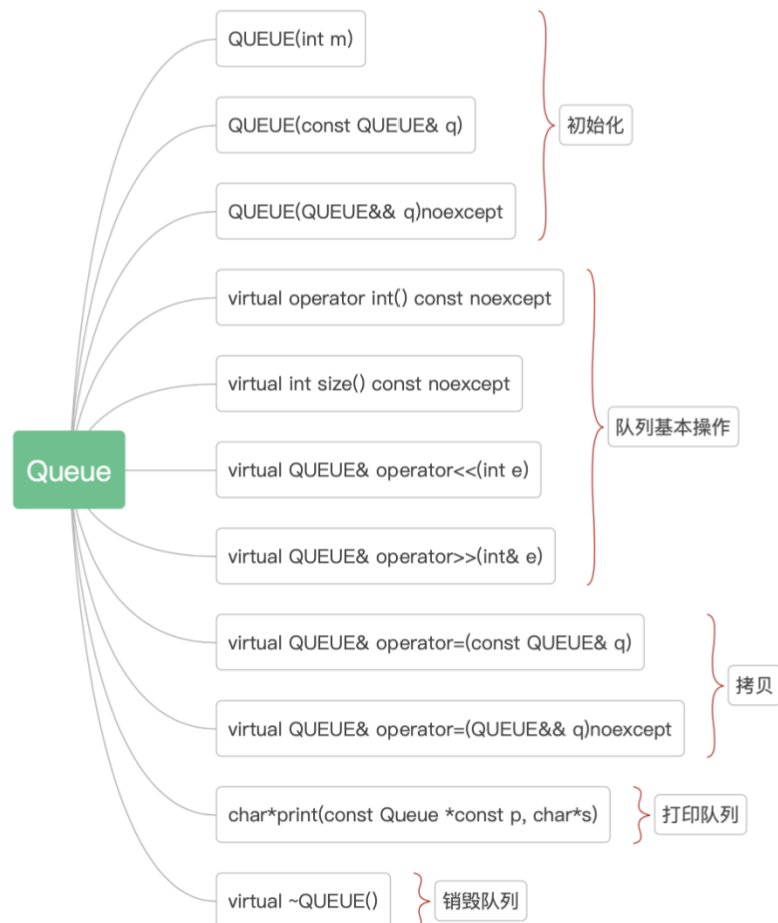


图 2 QUEUE 实现

## 2. 详细设计

### 2.1 QUEUE 初始化

### 2.1.1 QUEUE(int m)

功能：初始化队列，最多申请 m 个元素

入口参数：队列最多存放元素个数 m

出口参数：无

流程：对于传入参数 m 进行判断，如果 m 小于等于 0，不合法则抛出异常。否则分配 m 个整数大小的空间，并将首地址赋值给 elems，max 设置为 m，队列头 head 和队列尾都初始化为 0。

### 2.1.2 QUEUE(const QUEUE& q)

功能：用 q 深拷贝初始化队列

入口参数：队列对象常引用 q

出口参数：无

流程：给当前队列分配同样大小的空间，首地址赋值给 elems，将队列 q 中 elems 中的每个元素复制一份到当前队列的 elems 中，并将队列 q 的其他参数对应赋值给当前队列。

### 2.1.3 QUEUE(QUEUE&& q)noexcept

功能：用 q 移动初始化队列

入口参数：队列常引用 q

出口参数：无

流程：将队列 q 中的所有元素赋值给当前队列的对应参数，并且将 q 队列中的每个元素进行初始化，将 elems 赋值为空，将 max、head、tail 都赋值为 0。

## 2.2 QUEUE 基本操作

### 2.2.1 virtual int int() const noexcept

功能：重载 int() 返回当前队列中实际元素的个数

入口参数：无

出口参数：队列中实际元素个数

流程：利用队列的队首和队尾的编号，以及队列能存放的最大元素个数。利用以下计算元素个数的公式  $(\text{tail} - \text{head} + \text{max}) \% \text{max}$  可以得到队列存放元素的个数。

### 2.2.2 virtual operator size() const noexcept

功能：重载 size() 返回队列申请的最大元素个数 max

入口参数：无

出口参数：队列中能存放的最大元素个数

流程：返回队列中的 max。

### 2.2.3 QUEUE& operator<<(int e)

功能：重载<< 将 e 入队列尾部，并返回当前队列

入口参数：入队整数 e

出口参数：入队后队列的常引用

流程：首先判断队列是否满，如果满，则抛出异常，否则将 e 存放到队尾在队列中的下一个元素，并更新队列队尾的编号，返回更新后的队列的引用。

#### 2.2.4 QUEUE& operator>>(int& e)

功能：重载>> 从队首出元素到 e

入口参数：用来接收队首元素的整数变量 e

出口参数：入队后队列的常引用

流程：首先判断队列是否空，如果空，则抛出异常，否则取出队列中的第一个元素赋值给 e，并更新队首在队列中的编号，返回更新后的队列的引用。

### 2.3 拷贝 QUEUE

#### 2.3.1 QUEUE& operator=(const QUEUE& q)

功能：深拷贝赋 q 给队列

入口参数：队列常引用 q

出口参数：深拷贝后当前队列的引用

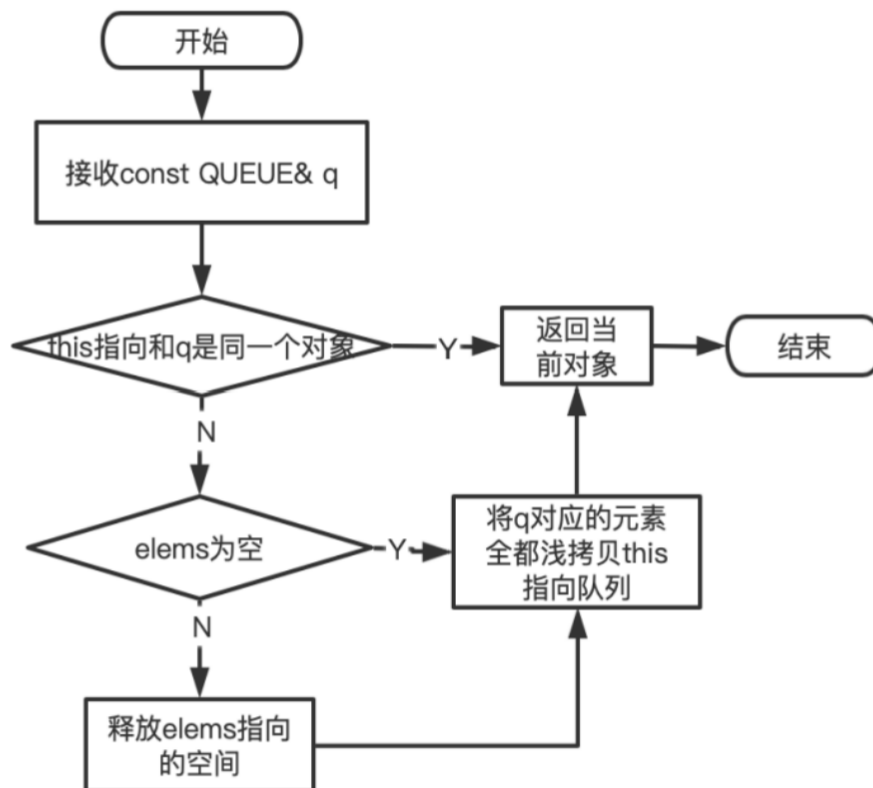


图 3 深拷贝赋值流程图

#### 2.3.2 Queue\*const assign(Queue\*const p, Queue&&q)

功能：移动赋 s 给队列并返回 p

入口参数：队列指针 p，队列常引用 s

出口参数：无



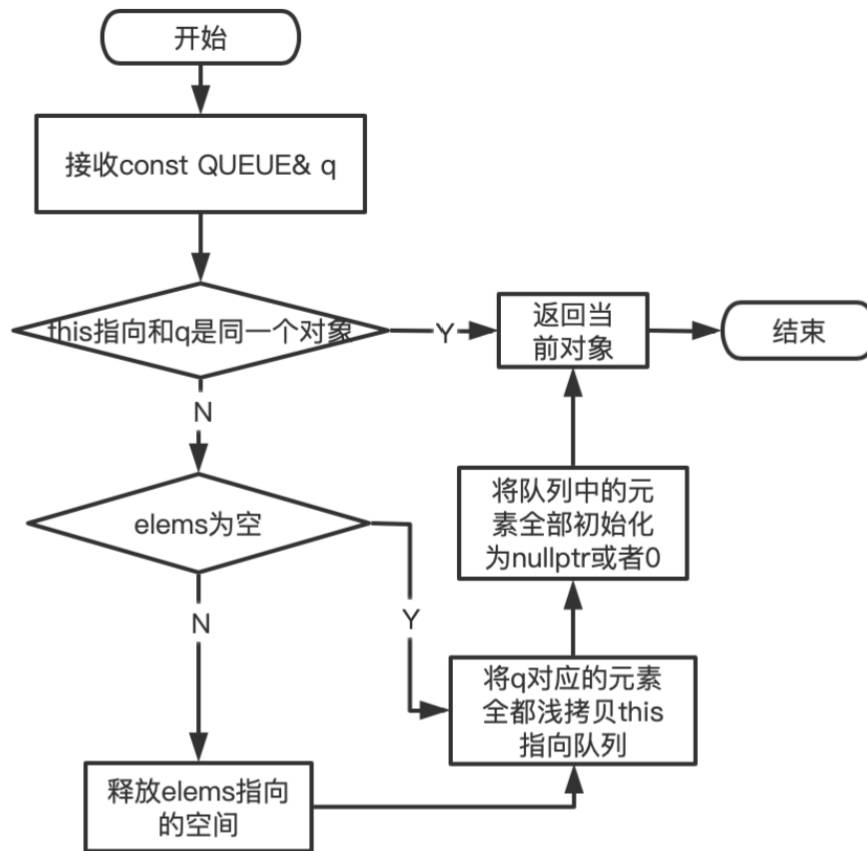


图 4 移动赋值流程图

## 2.4 打印 Queue

`char* print(char* s) const noexcept`

功能：打印当前指队列至 s 并返回 s

入口参数：存放队列元素字符的字符指针 s

出口参数：队列元素字符串对应的字符指针

流程：从队首到队尾遍历一遍队列中的元素，每读到一个元素就将其与一个逗号分隔符拼接到字符串 s 中，返回队列元素字符串的字符指针。

## 2.5 销毁 Queue

`~QUEUE()`

功能：销毁当前队列

入口参数：无

出口参数：无

流程：判断当前队列中的 elems 是否为空，如果不为空，就将 elems 指向的空间释放，并赋值为 nullptr，同时将 max 赋值为 0。

# 三、软件开发

硬件环境：PC 机，Intel Core i5 四核 CPU 1.4GHz，8G 内存

使用操作系统: MacOS Big Sur11.6

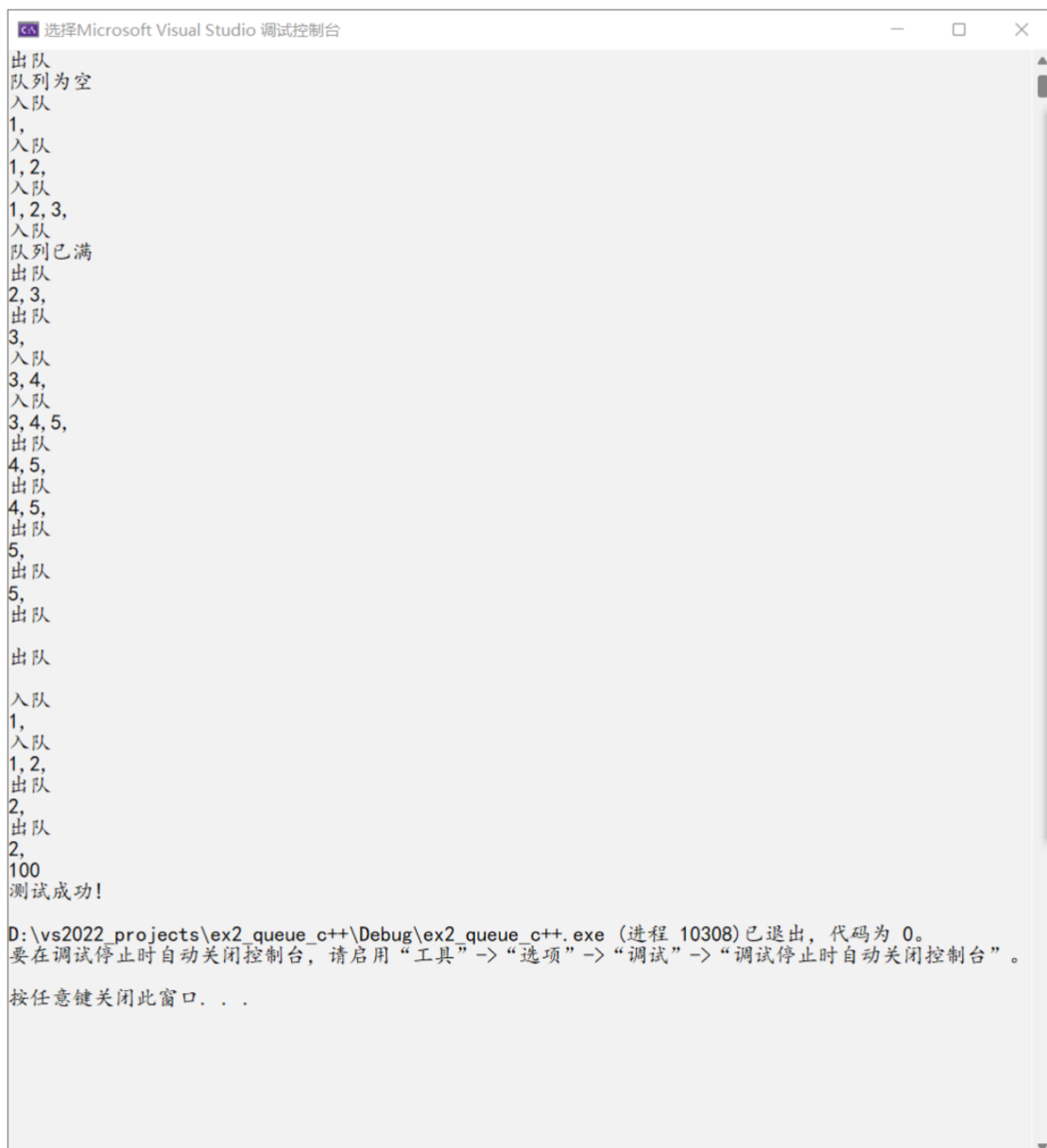
程序运行平台: VS2019

编译模式: x86

### 四、软件测试

测试结果以及得分如下图所示。

可以看到所有功能基本实现。



```
选择Microsoft Visual Studio 调试控制台
出队
队列为空
入队
1,
入队
1, 2,
入队
1, 2, 3,
入队
队列已满
出队
2, 3,
出队
3,
入队
3, 4,
入队
3, 4, 5,
出队
4, 5,
出队
4, 5,
出队
5,
出队
5,
出队
出队
入队
1,
入队
1, 2,
出队
2,
出队
2,
100
测试成功!

D:\vs2022_projects\ex2_queue_c++\Debug\ex2_queue_c++.exe (进程 10308) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。

按任意键关闭此窗口...
```

图 5 测试结果 1



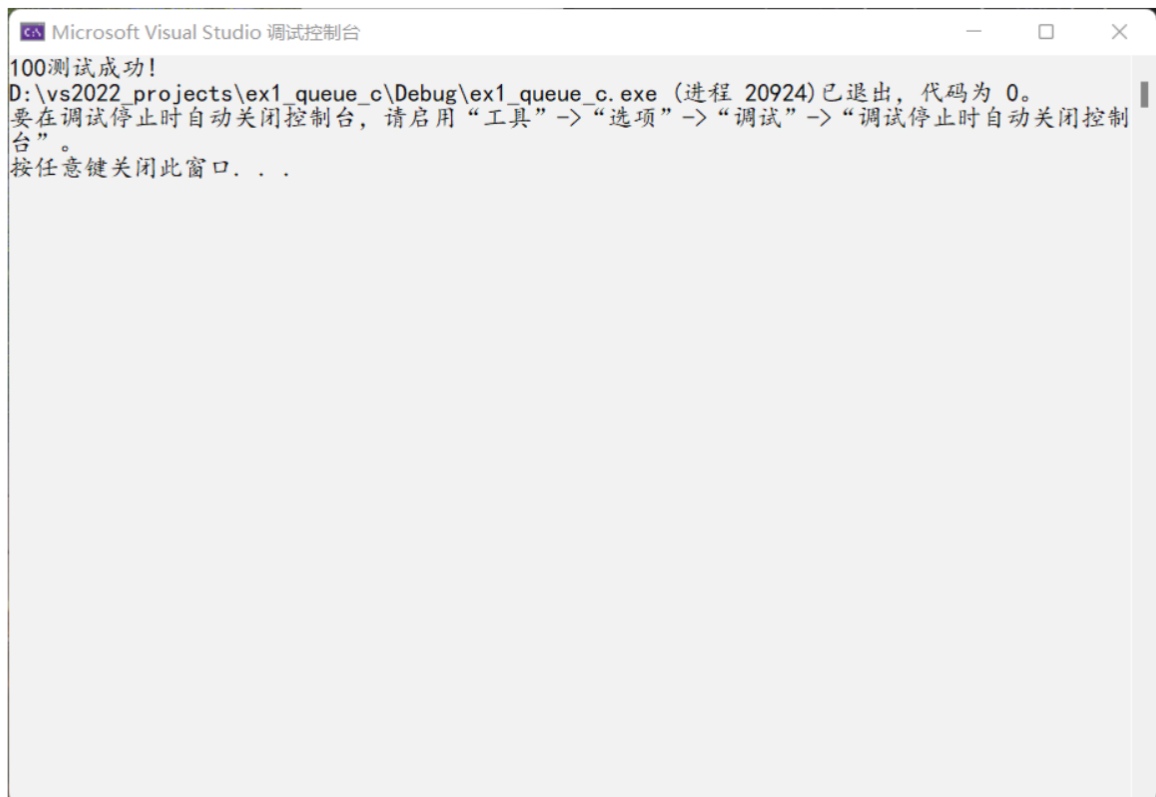


图 6 测试得分

## 五、特点与不足

### 1. 技术特点

通过数据结构的设计以及函数封装以及 C++ 的类等实现了模块化, 增加了代码复用性, 函数化、模块化的程序使得调试过程更容易找到 bug 所在。

对重要数据类型声明为常量, 函数接口也设计为常量, 增加了程序运行过程中的安全性。同时类的成员函数声明为虚函数, 方便派生出其他类, 实现多态。

同时通过操作符重载等使操作符具有新功能, 使得编写代码是代码简洁明了, 增强了代码的可读性。

### 2. 不足和改进的建议

队列的最大存放数据元素个数一旦经过初始化, 就无法修改, 对于开始对于数据量不定的问题, 不能够保证取得一个合适的队列最大存放元素个数。

可以增加函数来实现队列的增加容量的功能, 使得队列能够根据数据量持续增加自身所能存放的数据元素的个数, 从而增加该程序的适应性。

## 六、过程和体会

### 1. 遇到的主要问题和解决方法

无法直接修改常量内容, 通过强制类型转换, 来改变常量中的值。

对于浅拷贝的考虑不周到, 使得销毁队列的时候可能会出现再次释放已经被释放掉的内容, 从而出现

错误，在销毁队列函数中对 `elems` 是否为空进行单独的判断，避免重复释放。

## 2. 课程设计的体会

通过课程设计，对于 C++ 课程中学到的原理进行了实践，实现了队列的初始化、入队、出队等基本功能，对于队列的实现原理有了更深一步的理解。

对于浅拷贝、深拷贝过程中出现错误的情况进行 debug，从而提高了逻辑的严谨性。通过函数封装等，进一步理解了模块化编程。

对于操作符重载的意义有了更深层次的理解，操作符重载对于代码编写很有帮助。

更深层次地理解了面向对象过程中的继承和多态等。

通过 `const` 关键字的使用，更加理解安全性的用意。通过对整个程序的设计、编写、调试等强化了对程序的整体性的把握。

## 七、源码和说明

### 1. 文件清单及其功能说明

`ex2_queue.h`、`ex2_queue.cpp`、`ex2_test.cpp`、`ex2_queue_c++.vcxproj`、`ex2_queue_c++.vcxproj.filters`、`ex2_queue_c++.vcxproj.user`、`QUEUE.exe`。

其中 `QUEUE.exe` 为可执行程序，`ex2_queue.h`、`ex2_queue.cpp`、`ex2_test.cpp` 为源代码，`ex2_.vcxproj`、`ex2_queue_c++.vcxproj.filters`、`ex2_queue_c++.vcxproj.user` 为 VS 工程文件。

### 2. 用户使用说明书

下载 `QUEUE.exe`，执行 `QUEUE.exe`。

### 3. 源代码

`ex2_queue.h`

1. `#define _CRT_SECURE_NO_WARNINGS`
2. `#include <iostream>`
3. `#include <string.h>`
4. `using namespace std;`
5. `class QUEUE{`
6.     `int* const elems; //elems 申请内存用于存放队列的元素`
7.     `const int max; //elems 申请的最大元素个数为 max`
8.     `int head, tail; //队列头 head 和尾 tail，队空 head=tail;初始 head=tail=0`
9. `public:`
10.     `QUEUE(int m); //初始化队列：最多申请 m 个元素`
11.     `QUEUE(const QUEUE& q); //用 q 深拷贝初始化队列`
12.     `QUEUE(QUEUE&& q)noexcept; //用 q 移动初始化队列`
13.     `virtual operator int() const noexcept; //返回队列的实际元素个数`
14.     `virtual int size() const noexcept; //返回队列申请的最大元素个数 max`

```
15.     virtual QUEUE& operator<<(int e); //将 e 入队列尾部, 并返回当前队列
16.     virtual QUEUE& operator>>(int& e); //从队首出元素到 e, 并返回当前队列
17.     virtual QUEUE& operator=(const QUEUE& q); //深拷贝赋值并返回被赋值队列
18.     virtual QUEUE& operator=(QUEUE&& q)noexcept; //移动赋值并返回被赋值队列
19.     virtual char * print(char *s) const noexcept; //打印队列至 s 并返回 s
20.     virtual ~QUEUE(); //销毁当前队列
21. };
```

#### ex2\_queue.cpp

```
1.  #define _CRT_SECURE_NO_WARNINGS
2.  #include "ex2_queue.h"
3.  #include <malloc.h>
4.  #include <cstring>
5.  #include <iostream>
6.  using namespace std;
7.  QUEUE::QUEUE(int m) : max(m), elems(new int[m]), head(0), tail(0) {}
8.  QUEUE::QUEUE(const QUEUE& q) : max(q.max), elems(new int[q.max]), head(q.head), tail(q.tail) {
9.      //用 q 深拷贝初始化队列
10.     //分配 elems 空间
11.     for (int i = 0; i < max; i++)
12.         elems[i] = q.elems[i];
13. }
14. QUEUE::QUEUE(QUEUE&& q)noexcept :max(q.max), elems(q.elems), head(q.head), tail(q.tail) {
15.     //用 q 移动初始化队列
16.     *(int**)&q.elems = nullptr;
17.     *(int*)&q.max = 0;
18.     q.head = q.tail = 0;
19. }
20. QUEUE::operator int() const noexcept {
21.     //返回队列的实际元素个数
22.     if (!max)
23.         return 0;
24.     return (tail - head + max) % max;
25. }
26. int QUEUE::size() const noexcept {
27.     //返回队列申请的最大元素个数 max
28.     return max;
```

```

29. }
30. QUEUE& QUEUE::operator<<(int e) {
31.     //将 e 入队列尾部, 并返回当前队列
32.     cout<<" 入队"<<endl;
33.     if ((tail + 1) % max == head) {
34.         cout << "队列已满" << endl;
35.         throw "QUEUE is full!";
36.     }
37.     elems[tail] = e;
38.     tail = (tail + 1) % max;
39.     char s[20];
40.     print(s);
41.     return *this;
42. }
43. QUEUE& QUEUE::operator>>(int& e) {
44.     //从队首出元素到 e, 并返回当前队列
45.     cout << "出队" << endl;
46.     if (head == tail) {
47.         cout << "队列为空" << endl;
48.         throw "QUEUE is empty!";
49.     }
50.     e = elems[head];
51.     head = (head + 1) % max;
52.     char s[20];
53.     print(s);
54.     return *this;
55. }
56. QUEUE& QUEUE::operator=(const QUEUE& q) {
57.     //深拷贝赋值并返回被赋值队列
58.     *(int**)&elems = new int[q.max];
59.     for (int i = 0; i < max; i++)
60.         elems[i] = q.elems[i];
61.     *(int*)&max = q.max;
62.     head = q.head;
63.     tail = q.tail;
64.     return *this;

```

```

65. }
66. QUEUE& QUEUE::operator=(QUEUE&& q)noexcept {
67.     //移动赋值并返回被赋值队列
68.     if (elems == q.elems) return *this;
69.     if (elems)
70.         delete[] elems;
71.     *(int**)&elems = q.elems;
72.     *(int**)&q.elems = nullptr;
73.     *(int*)&max = q.max;
74.     *(int*)&q.max = 0;
75.     head = q.head;
76.     q.head = 0;
77.     tail = q.tail;
78.     q.tail = 0;
79.     return *this;
80. }
81. char* QUEUE::print(char* s) const noexcept {
82.     //打印队列至 s 并返回 s
83.     s[0] = 0;
84.     char* pos = s;
85.     for (int i = head; i != tail; i = (i + 1) % max) {
86.         sprintf(pos, "%d,", elems[i]);
87.         pos = s + strlen(s);
88.     }
89.     cout << s << endl;
90.     return s;
91. }
92. QUEUE::~~QUEUE() {
93.     //销毁当前队列
94.     if(elems)
95.         delete[] elems;
96.     *(int**)&elems = nullptr;
97.     *(int*)&max = 0;
98.     head = tail = 0;
99. }

```

ex2\_test.cpp

```
1. #include <iostream>
2. using namespace std;
3. extern const char * TestQUEUE(int &s);
4. int main()
5. {
6.     int s;
7.     string str=TestQUEUE(s);
8.     cout<<str<<endl<<s<<endl;
9.     return 0;
10. }
```