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：面向对象的整型队列编程

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

本次实验要求用 C++实现对循环队列的基本操作，循环队列相关信息用类储存，基本操作包括初始化循环队列（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、获取循环队列实际长度、获取循环队列最大长度、入队、出队、用深拷贝和移动构造方法实现赋值、打印循环队列中的元素、销毁循环队列等功能。

二、系统设计

1. 概要设计

本次实验用类来表示循环队列，类的成员变量包括指向循环队列的指针、所能包含的最多的元素个数、队首队尾指针。所实现的对循环队列的操作包括初始化循环队列（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、获取循环队列实际长度、获取循环队列最大长度、入队、出队、用深拷贝和移动构造方法实现赋值、打印循环队列中的元素、销毁循环队列等。

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：

```
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
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
```

总体流程图如下：

图 2-1 总体流程图

2. 详细设计

①循环队列类：

```
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
}
```

②函数原型：QUEUE::QUEUE(int m);
函数功能：初始化指定长度的循环队列
入口参数：循环队列的长度 int m（隐含参数 this）
出口参数：无
流程：给 elems 分配 m 大小的空间，将 m 赋值给 max，将 0 赋值给 head 和 tail

③函数原型：QUEUE::QUEUE(const QUEUE& q);
函数功能：用 q 深拷贝初始化队列
入口参数：已知循环队列引用 const Queue& q（隐含参数 this）
出口参数：无
流程：给 elems 分配 q.max 大小的空间，利用循环将 q->elems 的所有元素赋值给 elems 的所有元素，将 q 中的其余成员变量赋值给对应的成员变量。

④函数原型：QUEUE::QUEUE(QUEUE&& q)noexcept;
函数功能：用 q 移动初始化队列
入口参数：已知循环队列引用 Queue&& q（隐含参数 this）
出口参数：无
流程：将 q 中的所有成员变量直接赋值给对应成员变量，并将 q 中所有的成员变量置为 0。

⑤函数原型：QUEUE::operator int() const noexcept;
函数功能：运算符重载，返回循环队列的实际元素个数
入口参数：无（隐含参数 this）
出口参数：循环队列实际元素个数

流程：如果 elems 非空，返回队列长度(tail - head + max) % max，否则返回 0。

⑥函数原型：int QUEUE::size() const noexcept;
函数功能：返回队列申请的最大元素个数 max
入口参数：无（隐含参数 this）
出口参数：循环队列最大元素个数
流程：返回 max

⑦函数原型：QUEUE& QUEUE::operator<<(int e);
函数功能：运算符重载，将 e 入队列尾部，并返回队列
入口参数：入队元素 int e（隐含参数 this）
出口参数：队列引用
流程图：

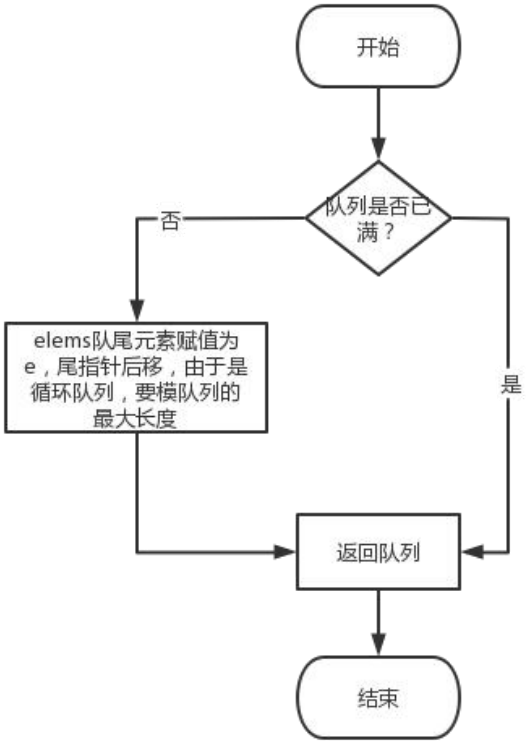

图 2-2 入队函数流程图

⑧函数原型：QUEUE& QUEUE::operator>>(int& e);
函数功能：运算符重载，从队首出元素到 e，并返回队列
入口参数：出队元素引用 int& e（隐含参数 this）
出口参数：队列引用
流程图：

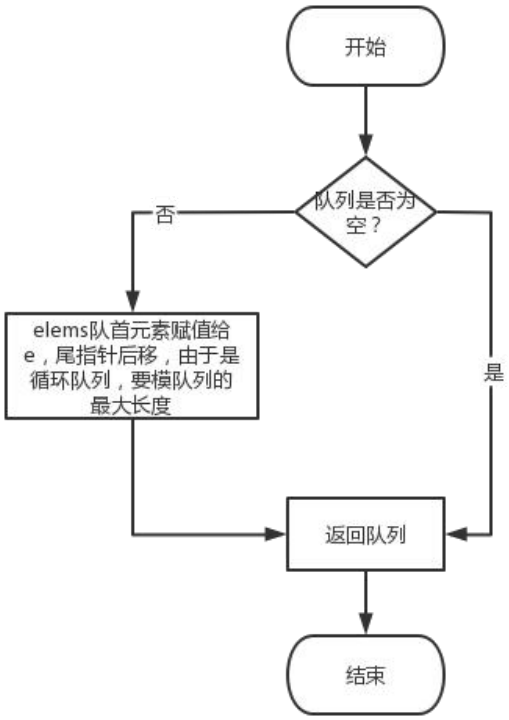

图 2-3 出队函数流程图

⑨函数原型：QUEUE& QUEUE::operator=(const QUEUE& q)；

函数功能：运算符重载，深拷贝赋值并返回被赋值队列

入口参数：已知循环队列引用 const Queue& q（隐含参数 this）

出口参数：循环队列引用

流程：如果 this==&q，直接返回*this，如果 elems 不为空，释放空间，之后与前面定义的深拷贝初始化函数相同。

⑩函数原型：QUEUE& QUEUE::operator=(QUEUE&& q)noexcept；

函数功能：运算符重载，移动赋值并返回被赋值队列

入口参数：已知循环队列引用 Queue&& q（隐含参数 this）

出口参数：循环队列引用

流程：如果 this==&q，直接返回*this，如果 elems 不为空，释放空间，之后与前面定义的移动初始化函数相同。

⑪函数原型：char* QUEUE::print(char* s) const noexcept；

函数功能：打印队列至 s 并返回 s

入口参数：输出字符串 char* s（隐含参数 this）

出口参数：输出字符串

流程：将每一个元素用 sprintf 函数输出到 s 字符串中，返回字符串

⑫函数原型：QUEUE::~QUEUE()；

函数功能：销毁循环队列

入口参数：无（隐含参数 this）

出口参数：无

流程：如果 elems 非空，将 elems 申请的空间释放，并将所有成员变量赋值为 0。

三、软件开发

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

四、软件测试

采用实验二测试库，测试结果如图 4-1 所示。

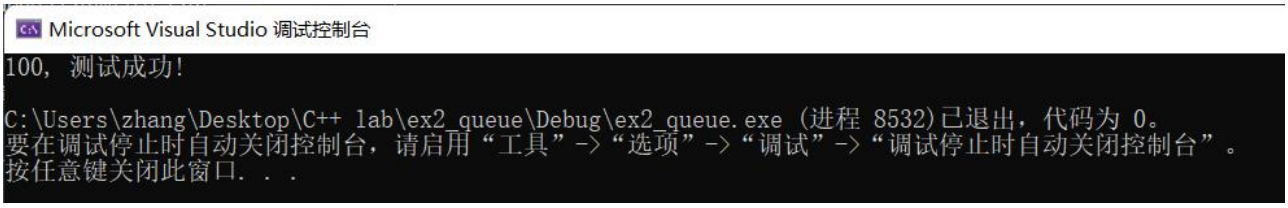

图 4-1　实验二测试图

五、特点与不足

1. 技术特点

使用类封装队列，每次移动头指针或者尾指针的时候，都对队列最大长度进行取模运算，这样就避免了越界问题

2. 不足和改进的建议

代码风格不佳，NULL、nullptr、0 乱用，一些特殊情况考虑不够全面，如只有当 elems 非空的时候才可以析构队列。

之后应该改进自己的代码风格，要做到前后统一，同时考虑问题的时候要细致。

六、过程和体会

1. 遇到的主要问题和解决方法

本次实验与第一次实验比较类似，所以并没有遇到一些问题，编写过程比较顺利。

2. 课程设计的体会

在本次实验中我体会到了分模块逐函数编程的思想，将每一个函数全部实现，最后再考

虑整合，这样在出现问题的时候可以快速定位，减少许多的调试时间。

我体会到C++用类进行封装的思想，封装是面向对象的重要特性，能较好地保证代码的安全性。

同时我体会到特殊情况的重要性，在编写函数的时候应该综合考虑各种可能出现的情况，尽量保证代码的完善，避免后期的调试和修改。

七、源码和说明

1. 文件清单及其功能说明

ex2_queue.h 是头文件（类和函数声明）

ex2_queue.cpp 是源文件（函数定义）

ex2_test.cpp 是源文件（main 函数）

ex2_queue.sln 用来打开项目

“实验二测试库”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 ex2_queue.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

ex2_queue.h：

```
#pragma once
#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
};
```

ex2_queue.cpp：

```
#include"ex2_queue.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,只读成员必须在初始化列表中进行初始化。
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，只读成员必须在初始化列表中进行初始化。
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，只读成员必须在初始化列表中进行初始化。
QUEUE::operator int() const noexcept
{
    if (elems)
        return (tail - head + max) % max; //循环队列长度公式
    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) //判断是否已满
```

```
        throw"QUEUE is full!";
    elems[tail] = e;
    tail = (tail + 1) % max;   //循环队列取模
    return *this;
}//将 e 入队列尾部，并返回当前队列
QUEUE& QUEUE::operator>>(int& e)
{
    if (head == tail)   //判断是否为空
        throw"QUEUE is empty!";
    e = elems[head];
    head = (head + 1) % max;   //循环队列取模
    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;   //避免自己给自己赋值
    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;   //避免自己给自己赋值
    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 < (*this); i++)
        s += sprintf(s, "%d,", elems[(head + i) % max]); //头指针移动实现遍历
    return s;
}//打印队列至 s 并返回 s
QUEUE::~QUEUE()
{
    if (elems)    //elems 非空才可以销毁
    {
        delete[] elems;
        *(int**)&elems = 0;
        head = tail = *(int*)&max = 0;
    }
}//销毁当前队列
```

ex2_test.cpp:

```
#include<iostream>
#include"ex2_queue.h"
extern const char* TestQUEUE(int& s); //测试函数
int main()
{
    int s; //分数
    const char* q = TestQUEUE(s);    //提示信息字符串
    printf("%d, %s\n", s, q);
    return 0;
}
```