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：面向过程的整型队列编程

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 14 日

一、需求分析

1.题目要求

整型循环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：向循环队列尾部添加一个整型元素、从循环队列首部移除一个整型元素等。整型循环队列类型 Queue 及其操作函数采用非面向对象的 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 函数对循环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>
struct Queue{
   int* const  elems;       //elems 申请内存用于存放循环队列的元素
   const  int  max;      //elems 申请的最大元素个数 max
   int   head, tail;     //队列头 head 和尾 tail,队空 head=tail;初始 head=tail=0
};
void initQueue(Queue *const p, int m);    //初始化 p 指队列：最多申请 m 个元素
void initQueue(Queue *const p, const Queue&s); //用 s 深拷贝初始化 p 指队列
void initQueue(Queue *const p, Queue&&s); //用 s 移动初始化 p 指队列
int  number (const Queue *const p);   //返回 p 指队列的实际元素个数
int  size(const Queue *const p);          //返回 p 指队列申请的最大元素个数 max
Queue*const enter(Queue*const p, int e);  //将 e 入队列尾部，并返回 p
Queue*const leave(Queue*const p, int &e); //从队首出元素到 e，并返回 p
Queue*const assign(Queue*const p, const Queue&q); //深拷贝赋 s 给队列并返回 p
Queue*const assign(Queue*const p, Queue&&q); //移动赋 s 给队列并返回 p
char*print(const Queue *const p, char*s);//打印 p 指队列至 s 并返回 s
void destroyQueue (Queue *const p);     //销毁 p 指向的队列
```

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 initQueue(Queue *const p, int m)对 p 指向的循环队列初始化时， 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 initQueue(Queue *const p, const Queue& q)初始化，用已经存在的对象 q 深拷贝构造新对象*p 时,新对象*p 不能和对象 q 的 elems 共用同一块内存,新对象*p 的 elems 需要分配和 q 为 elems 分配的同样大小的内存，并且将已经存在 q 的 elems 的内容深拷贝至新分配的内存；新对象*p 的 max、head、tail 应设置成和已经存在的对象 s 相同。

（3）对于 initQueue(Queue *const p, Queue&& q)初始化，用已经存在的对象 q 移动构

造新对象，新对象使用对象 q 为 elems 分配的内存快，并将其 max、head、tail 设置成和 s 的对应值相同，然后将 s 的 elems 设置为空表示内存已经移动给新对象，将 s 的 max、head、tail 设置为 0。

（4）对于 Queue*const assign(Queue*const p, const Queue&q)深拷贝赋值，用等号右边的对象 q 深拷贝赋值给等号左边的对象，等号左边的对象如果已经有内存则应先释放以避免内存泄漏，然后分配和对象 q 为 elems 分配的同样大小的内存，并且设置其 max、head、tail 和 q 的对应值相同。

（5）对于 Queue*const assign(Queue*const p, Queue&&q)移动赋值，若等号左边的对象为 elems 分配了内存，则先释放改内存一面内存泄漏，然后使用等号右边对象 q 为 elems 分配的内存，并将其 max、head、tail 设置成和对象 q 的对应值相同；对象 q 的 elems 设置为空指针以表示内存被移走给等号左边的对象，同时其 max、head、tail 均应设置为 0。

（6）对于循环队列，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，则表示表示循环队列已满，故循环队列最多存放 max-1 个元素；而当 head=tail 时则表示循环队列为空。循环队列空取出元素或循环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）打印循环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 语言实现对循环队列的基本操作，循环队列相关信息用结构体储存，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

本次实验用结构体来表示循环队列，结构体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：

```
void initQueue(Queue *const p, int m);     //初始化 p 指队列：最多申请 m 个元素
void initQueue(Queue *const p, const Queue&s); //用 s 深拷贝初始化 p 指队列
void initQueue(Queue *const p, Queue&&s); //用 s 移动初始化 p 指队列
int  number (const Queue *const p);    //返回 p 指队列的实际元素个数
```

```
int  size(const Queue *const p);          //返回p指队列申请的最大元素个数max
Queue*const enter(Queue*const p, int e);  //将e入队列尾部，并返回p
Queue*const leave(Queue*const p, int &e); //从队首出元素到e，并返回p
Queue*const assign(Queue*const p, const Queue&q); //深拷贝赋s给队列并返回p
Queue*const assign(Queue*const p, Queue&&q); //移动赋s给队列并返回p
char*print(const Queue *const p, char*s);//打印p指队列至s并返回s
void destroyQueue (Queue *const p);     //销毁p指向的队列
```

总体流程图为：

图 2-1 总体流程图

2. 详细设计

①循环队列数据结构：

```
struct Queue{
   int* const  elems;       //elems申请内存用于存放循环队列的元素
   const  int  max;      //elems申请的最大元素个数max
   int  head, tail;      //队列头head和尾tail,队空head=tail;初始head=tail=0
};
```

②函数原型：void initQueue(Queue *const p, int m);

函数功能：初始化指定长度的循环队列

入口参数：循环队列指针 Queue *const p，最多申请的元素个数 int m

出口参数：无

流程:给 p->elems 分配 m 大小的空间,将 m 赋值给 p->max,将 0 赋值给 p->head 和 p->tail

③函数原型：void initQueue(Queue* const p, const Queue& s);

函数功能：用已知循环队列 s 深拷贝初始化 p 循环队列

入口参数：循环队列指针 Queue *const p，已知循环队列引用 const Queue& s

出口参数：无

流程：给 p->elems 分配 s.max 大小的空间，利用循环将 s->elems 的所有元素赋值给 p->elems 的所有元素，将 s 中的其余成员变量赋值给 p 的对应的成员变量。

④函数原型：void initQueue(Queue* const p, Queue&& s);

函数功能：用已知循环队列 s 移动初始化 p 循环队列

入口参数：循环队列指针 Queue *const p，已知循环队列引用 Queue&& s

出口参数：无

流程：将 s 中的所有成员变量直接赋值给 p 的对应成员变量，并将 s 中所有的成员变量置为 0。

⑤函数原型：int number (const Queue *const p);

函数功能：返回 p 所指的循环队列的实际元素个数

入口参数：循环队列指针 const Queue *const p

出口参数：循环队列实际元素个数

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

⑥函数原型：int size(const Queue* const p);

函数功能：返回 p 指队列申请的最大元素个数 max

入口参数：循环队列指针 const Queue *const p

出口参数：循环队列最大元素个数

流程：返回 p->max

⑦函数原型：Queue* const enter(Queue*const p, int e);

函数功能：将 e 入队列尾部，并返回 p

入口参数：循环队列指针 Queue *const p，入队元素 int e

出口参数：队列指针

流程图：

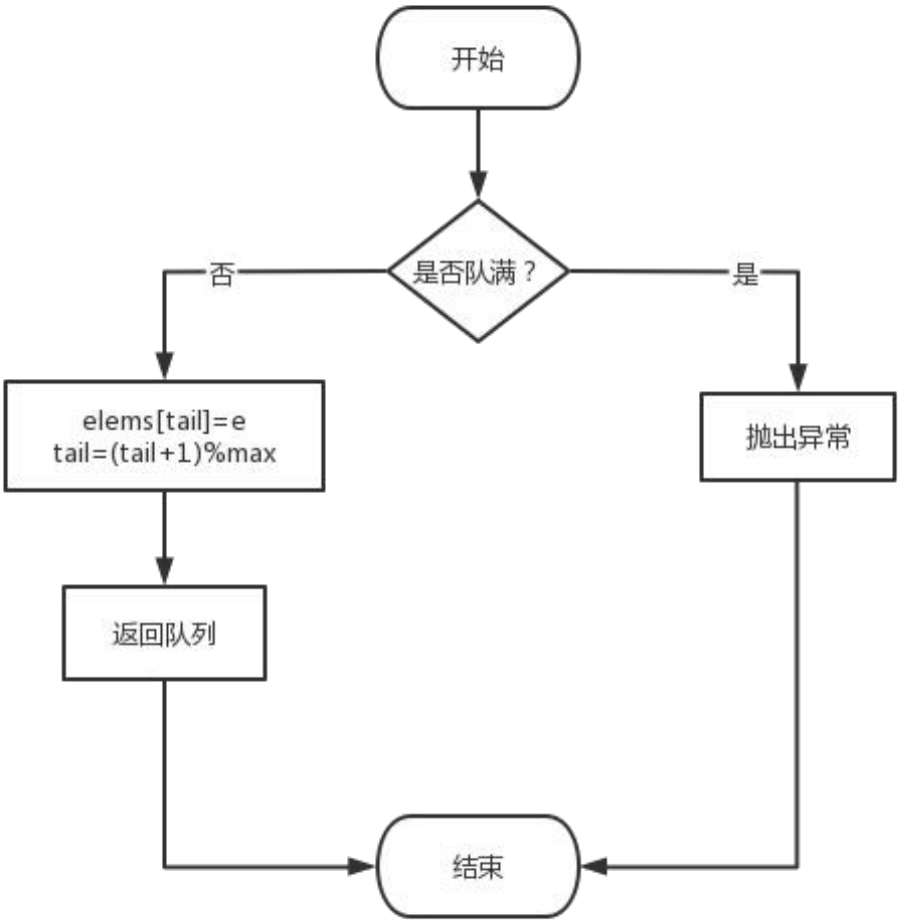

图 2-2 入队函数流程图

⑧函数原型：Queue* const leave(Queue* const p, int& e)；
函数功能：从队首出元素到 e，并返回 p
入口参数：循环队列指针 Queue *const p，出队元素引用 int& e
出口参数：队列指针
流程图：

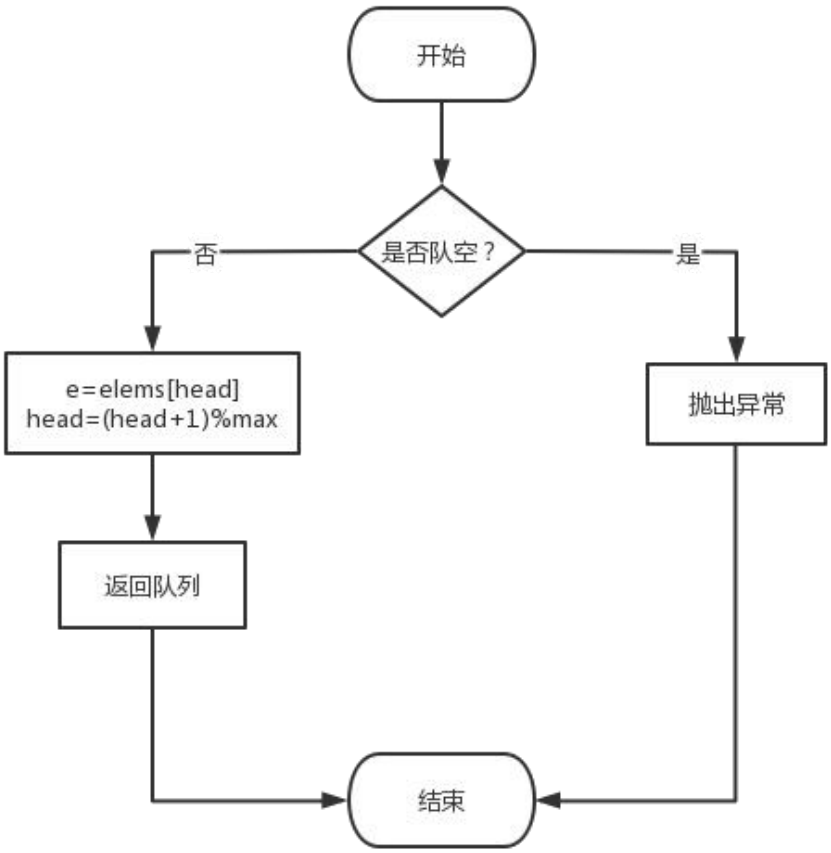

图 2-3 出队函数流程图

⑨函数原型：Queue* const assign(Queue* const p, const Queue& q);

函数功能：深拷贝赋 s 给队列并返回 p

入口参数：循环队列指针 Queue *const p，已知循环队列引用 const Queue& q

出口参数：循环队列指针

流程：如果 p==&q，直接返回 p，如果 p->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* const assign(Queue* const p, Queue&& q);

函数功能：移动赋 s 给队列并返回 p

入口参数：循环队列指针 Queue *const p，已知循环队列引用 Queue&& q

出口参数：循环队列指针

流程：如果 p==&q，直接返回 p，如果 p->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* print(const Queue* const p, char* s)；

函数功能：打印 p 指队列至 s 并返回 s

入口参数：循环队列指针 const Queue *const p，输出字符串 char* s

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：void destroyQueue(Queue* const p)；

函数功能：销毁循环队列

入口参数：循环队列指针：Queue* const p

出口参数：无

流程：如果 p->elems 非空，将 p->elems 申请的空间释放，并将 p 的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

采用实验一测试库，测试结果如图 4-1 所示。

图 4-1 实验一测试图

五、特点与不足

1. 技术特点

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 p->elems 非空的时候才可以销毁队列。

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

主要问题：在编写深拷贝赋值函数和移动赋值函数的时候，没有考虑将对象本身的引用

传入函数的情况，导致测试不通过。

解决方法：在进入函数的时候判断一下，如果 this 与传入的变量的地址相同，就直接返回*this，避免了接下来的错误操作。

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，先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。

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

ex1_queue_c.h 是头文件（结构体和函数声明）

ex1_queue_c.cpp 是源文件（函数定义）

ex1_test.cpp 是源文件（main 函数）

ex1_queue_c.sln 用来打开项目

“实验一测试库”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 ex1_queue_c.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

ex1_queue_c.h：

```
#pragma once
#include <stdio.h>
#include <string.h>
struct 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 max
    int    head, tail;    //队列头 head 和尾 tail，队空 head=tail;初始 head=tail=0
};
void initQueue(Queue* const p, int m);    //初始化 p 指队列：最多申请 m 个元素
void initQueue(Queue* const p, const Queue& s); //用 s 深拷贝初始化 p 指队列
void initQueue(Queue* const p, Queue&& s); //用 s 移动初始化 p 指队列
int  number(const Queue* const p);  //返回 p 指队列的实际元素个数
```

```
int    size(const Queue* const p);              //返回 p 指队列申请的最大元素个数 max
Queue* const enter(Queue* const p, int e);   //将 e 入队列尾部，并返回 p
Queue* const leave(Queue* const p, int& e); //从队首出元素到 e，并返回 p
Queue* const assign(Queue* const p, const Queue& q); //深拷贝赋 s 给队列并返回 p
Queue* const assign(Queue* const p, Queue&& q); //移动赋 s 给队列并返回 p
char* print(const Queue* const p, char* s);//打印 p 指队列至 s 并返回 s
void destroyQueue(Queue* const p);    //销毁 p 指向的队列
```

ex1_queue_c.cpp：

```
#include"ex1_queue_c.h"
void initQueue(Queue* const p, int m)
{
    *(int**)&p->elems = new int[m];
    *(int*)&p->max = m;
    p->head = p->tail = 0;
}//初始化 p 指队列：最多申请 m 个元素
void initQueue(Queue* const p, const Queue& s)
{
    *(int**)&p->elems = new int[s.max]; //深拷贝要求重新申请空间
    *(int*)&p->max = s.max;
    p->head = s.head;
    p->tail = s.tail;
    for (int i = 0; i < s.max; i++)
        p->elems[i] = s.elems[i];
} //用 s 深拷贝初始化 p 指队列
void initQueue(Queue* const p, Queue&& s)
{
    *(int*)&p->max = s.max;
    *(int**)&p->elems = s.elems;
    p->tail = s.tail;
    p->head = s.head;
```

```
    *(int**)&s.elems = 0; //移动构造要求赋值后将原已知队列的元素置为 0
    *(int*)&s.max = 0;
    s.tail = s.head = 0;
}//用 s 移动初始化 p 指队列
int number(const Queue* const p)
{
    if(p->elems)
        return (p->tail - p->head + p->max) % p->max; //循环队列长度的公式
    return 0;
}//返回 p 指队列的实际元素个数
int  size(const Queue* const p)
{
    return p->max;
}//返回 p 指队列申请的最大元素个数 max
Queue* const enter(Queue* const p, int e)
{
    if ((p->tail + 1) % p->max == p->head) //判断是否已满
        throw"Queue is full!";
    p->elems[p->tail] = e;
    p->tail = (p->tail + 1) % p->max; //循环队列注意模 max
    return p;
}//将 e 入队列尾部，并返回 p
Queue* const leave(Queue* const p, int& e)
{
    if (p->head == p->tail) //判断是否为空
        throw"Queue is empty!";
    e = p->elems[p->head];
    p->head = (p->head + 1) % p->max; //循环队列注意模 max
    return p;
}//从队首出元素到 e，并返回 p
```

```
Queue* const assign(Queue* const p, const Queue& q)
{
    if (p == &q) return p; //直接返回，不能出现自己赋值给自己的情况
    if (p->elems)
    {
        delete[] p->elems;
        *(int**)&p->elems = NULL;
    }
    *(int**)&p->elems = new int[q.max];
    *(int*)&p->max = q.max;
    p->head = q.head;
    p->tail = q.tail;
    for (int i = 0; i < q.max; i++)
        p->elems[i] = q.elems[i];
    return p;
}//深拷贝赋 s 给队列并返回 p
Queue* const assign(Queue* const p, Queue&& q)
{
    if (p==&q) return p; //直接返回，不能出现自己赋值给自己的情况
    if (p->elems) {
        delete[] p->elems;
        *(int**)&p->elems = NULL;
    }
    *(int**)&p->elems = q.elems;
    *(int*)&p->max = q.max;
    p->tail = q.tail;
    p->head = q.head;
    *(int**)&q.elems = NULL;
    *(int*)&q.max = 0;
    q.head = q.tail = 0;
```

```
    return p;
}//移动赋 s 给队列并返回 p
char* print(const Queue* const p, char* s)
{
    for (int i = 0; i < number(p); i++)
        s += sprintf(s, "%d,", p->elems[(p->head + i) % p->max]); //移动头指针实现遍历
    return s;
}//打印 p 指队列至 s 并返回 s
void destroyQueue(Queue* const p)
{
    if (p->elems) //当 elems 非空时才需要释放空间
    {
        delete[] p->elems;
        *(int**)&p->elems = 0;
        p->head = p->tail = *(int*)&p->max = 0;
    }
}//销毁
```

ex1_test.cpp:

```
#include<iostream>
#include"ex1_queue_c.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;
}
```