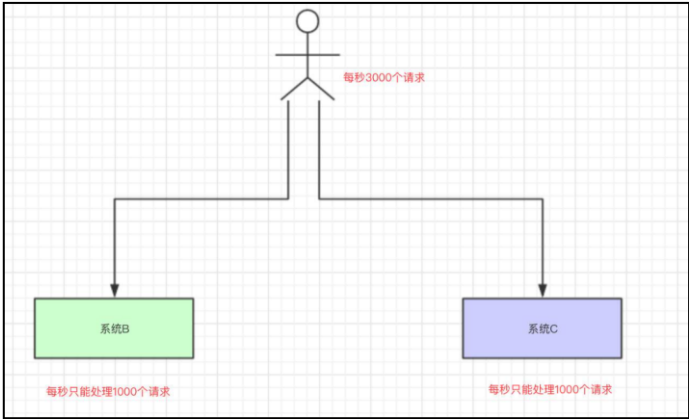
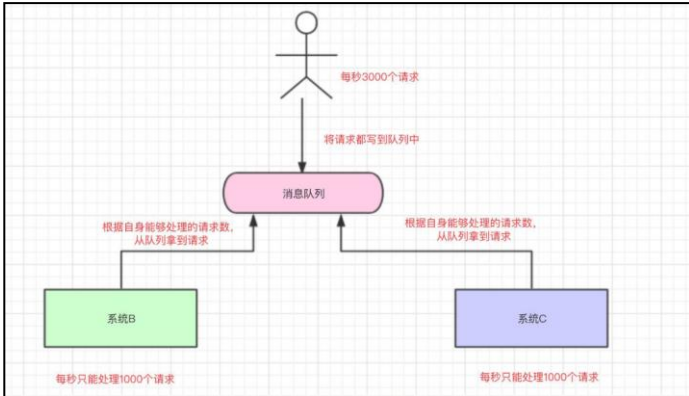


《数据结构》上机报告

2022 年 10 月 2 日

姓名：郑博远 学号：2154312 班级：计科1班 得分：_____

实验题目	消息队列
实验目的	1. 掌握队列的结构和基本操作； 2. 了解消息队列的使用场景（解耦、异步通信、限流消峰）
问题描述	<p>某宝平台定期要搞一次大促，大促期间的并发请求可能会很多，比如每秒 3000 个请求。假设我们现在有两台机器处理请求，并且每台机器只能每次处理 1000 个请求。如图（1）所示，那多出来的 1000 个请求就被阻塞了（没有系统响应）。</p>  <p>图 1 展示了无消息队列的场景。一个用户（人形图标）每秒发送 3000 个请求。这些请求被直接分发到两个系统：系统 B 和系统 C。系统 B 和系统 C 每秒只能处理 1000 个请求。由于请求量超过了系统的处理能力，多余的 1000 个请求被阻塞，无法得到系统响应。</p> <p>图 2 有消息队列的流程图</p>  <p>图 2 展示了有消息队列的场景。用户每秒发送 3000 个请求，这些请求首先被写入一个消息队列（粉色圆角矩形）。然后，系统 B 和系统 C 根据各自能够处理的请求数，从消息队列中取出请求进行处理。这样，即使每秒有 3000 个请求，系统也能有序地处理它们，不会出现阻塞的情况。</p> <p>请你实现一个消息队列，如图（2）所示。系统 B 和系统 C 根据自己能够处理的请求数去消息队列中拿数据，这样即便每秒有</p>

	<p>8000 个请求，也只是把请求放在消息队列中。如何去拿消息队列中的消息由系统自己去控制，甚至可以临时调度更多的机器并发处理这些请求，这样就不会让整个系统崩溃了。</p> <p>注：现实中互联网平台的每秒并发请求可以达到千万级。</p>	
基本要求	<p>定义顺序队列类型，使其具有如下功能：</p> <p>(1) 建立一个空队列；</p> <p>(2) 释放队列空间，将队列销毁；</p> <p>(3) 将队列清空，变成空队列；</p> <p>(4) 判断队列是否为空；</p> <p>(5) 返回队列内的元素个数；</p> <p>(6) 将队头元素弹出队列（出队）；</p> <p>(7) 在队列中加入一个元素（入队）；</p> <p>(8) 从队头到队尾将队列中的元素依次输出。</p>	
选做要求	本小题为选做	
	已完成选做内容（序号）	
数据结构设计	<p>本题探究消息队列在实际生活当中的应用。具体包括解耦、异步通信、限流消峰三个方面的知识，题目是对限流消峰的体现。所涉及到的数据结构队列我使用模板类进行封装，设计如下：</p> <pre> //循环队列 template <class QElemType> class SqQueue { private: QElemType* base; int MAXQSIZE; int front; int rear; public: SqQueue(int maxqsize); //建立空队列（构造函数） ~SqQueue(); //销毁队列（析构函数） int QueueLength(); //获取队列长度 Status EnQueue(QElemType e); //新元素入队 Status DeQueue(QElemType& e); //队首元素出队 Status GetHead(QElemType& e); //获取队首元素 Status ClearQueue(); //清空队列 bool QueueEmpty(); //判断队列是否为空 </pre>	

	<pre> Status PrintQueue(ostream& out); //从队首逐一输出元素 }; //本次队列存放元素的结构体 记录商品信息 struct Data { char CommodityName[128]; //商品名称 float Price; //商品价格 int RequestTime; //请求的时间 };</pre>
功能(函数) 说明	<pre>1. 队列类的成员函数 /* * SqQueue(int maxqsize) * @param maxqsize 队列的最大长度 */ template <class QElemType> SqQueue<QElemType>::SqQueue(int maxqsize); //空队列的建立（构造函数） /* * SqQueue() */ template <class QElemType> SqQueue<QElemType>::~~SqQueue(); //销毁已有的队列（析构函数） /* * Status ClearQueue() */ template <class QElemType> Status SqQueue<QElemType>::ClearQueue(); //把现有队列置空队列 /* * Status GetHead(QElemType& e) * @param e 取到的队首元素 */ template <class QElemType> Status SqQueue<QElemType>::Top(QElemType& e); //取队首元素 /*</pre>

```

* Status DeQueue(QElemType& e)
* @param e 取出的队首元素
*/
template <class QElemType>
Status SqQueue<QElemType>::DeQueue(QElemType& e);
//弹出队首元素

/*
* Status EnQueue(QElemType e)
* @param e 入队列的元素
*/
template <class QElemType>
Status SqQueue<QElemType>::EnQueue(QElemType e);
//新元素入队列

/*
* bool QueueEmpty()
*/
template <class QElemType>
bool SqQueue<QElemType>::QueueEmpty();
//当前队列是否为空队列

/*
* int QueueLength ()
*/
template <class QElemType>
int SqQueue<QElemType>::QueueLength();
//返回当前队列的长度

/*
* Status SqQueue<QElemType>::PrintQueue(ostream& out)
* @param out 输出流对象(可输出到屏幕、文件等)
*/
template <class QElemType>
Status SqQueue<QElemType>::PrintQueue(ostream& out)
//输出从队首到队尾的所有元素


```

2. 存放请求信息的结构体友元函数

```

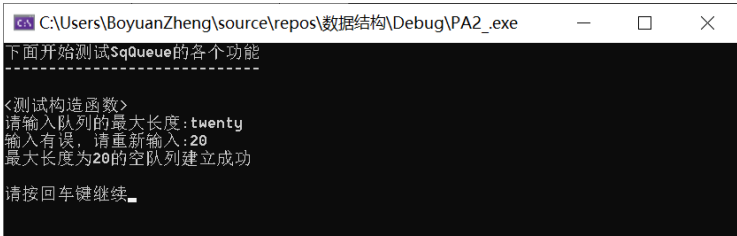
/*
* ostream& operator<<(ostream& out, const Data& that)
* @param out 输出流
* @param that 准备输出的商品信息
*/
ostream& operator<<(ostream& out, const Data& that)

```

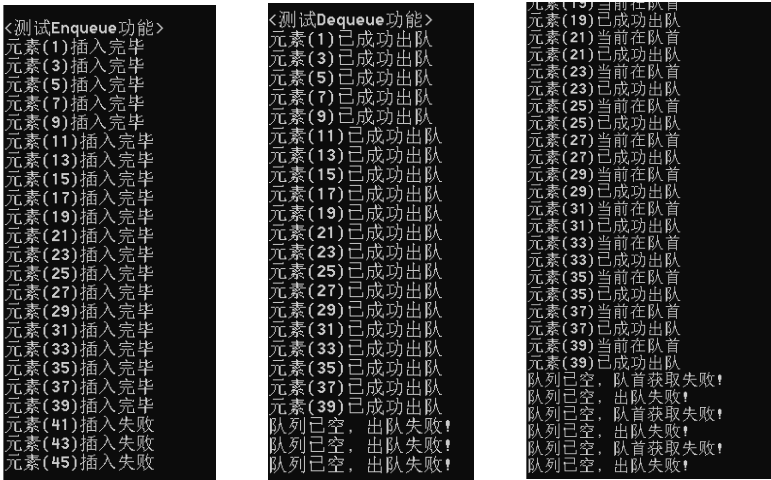
	<pre>//重载<<方便 cout 输出 3. 菜单项及各功能对应函数 /* * int Menu() */ int Menu(); //打印可视化菜单界面 返回所选项 /* * void TestBasicFunction() */ void TestBasicFunciton(); //测试队列基本功能的使用 /* * void Simulation1() */ void Simulation1(); //模拟无消息队列的情况 /* * void Simulation2() */ void Simulation2(); //模拟有消息队列的情况 /* * void wait_for_enter() */ void wait_for_enter(); //等待键盘读入回车</pre>
界面设计和使用说明	<p>程序开始，进入菜单界面进行功能选择。若输入错误的数字或其他非法字符，则程序不进行响应。重复读入直至选项正确：</p> 

选择菜单项 1，队列基础功能的测试：

输入队列最大长度，测试队列的构造。若输入长度错误、非法，则程序给出错误提示，反复读入直至正确。具体测试如下图：

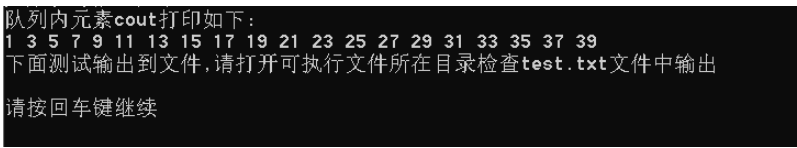


测试元素的入队、出队以及队首元素的读取，即 EnQueue、DeQueue、GetHead 函数的使用：

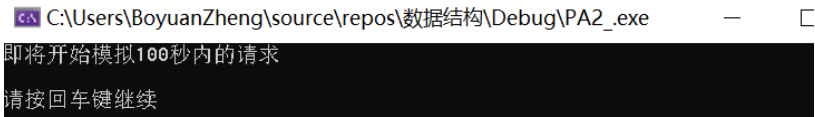


依次测试各个功能函数的使用，包含判断队列是否为空的 QueueEmpty，返回队列长度的 QueueLength，清空已有队列的 ClearQueue 等函数，不一一具体展示。

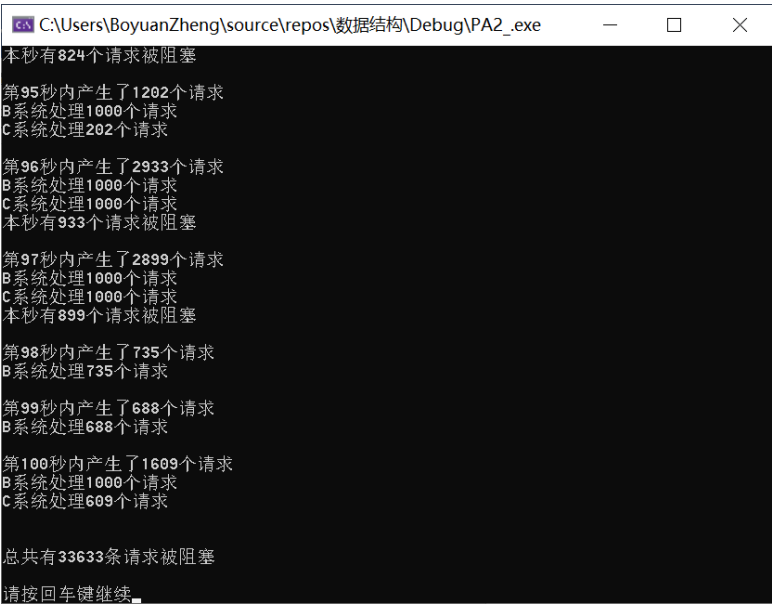
下面展示 PrintQueue 函数到 cout 到屏幕和到文件中的测试：



选择菜单项 2，模拟无消息队列情况下的场景：



程序模拟 100 秒内每秒的请求情况以及系统 B、C 的相应处理情况，具体如下图：



```
C:\Users\BoyuanZheng\source\repos\数据结构\Debug\PA2_exe
本秒有824个请求被阻塞

第95秒内产生了1202个请求
B系统处理1000个请求
C系统处理202个请求

第96秒内产生了2933个请求
B系统处理1000个请求
C系统处理1000个请求
本秒有933个请求被阻塞

第97秒内产生了2899个请求
B系统处理1000个请求
C系统处理1000个请求
本秒有899个请求被阻塞

第98秒内产生了735个请求
B系统处理735个请求

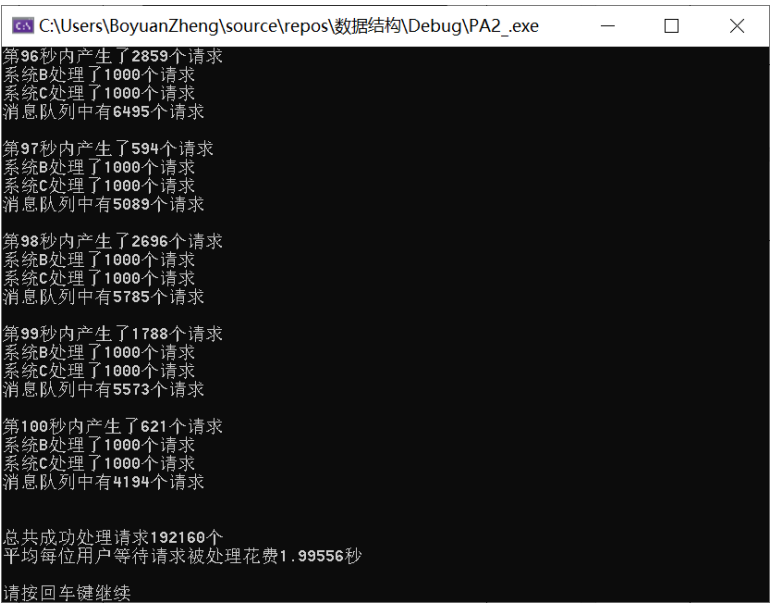
第99秒内产生了688个请求
B系统处理688个请求

第100秒内产生了1609个请求
B系统处理1000个请求
C系统处理609个请求

总共有33633条请求被阻塞
请按回车键继续
```

可以观察到，100 秒内总共有上万条请求被阻塞得不到响应，过多的请求也会导致系统 B、C 的崩溃。

选择菜单项 3，模拟有消息队列情况下的场景：



```
C:\Users\BoyuanZheng\source\repos\数据结构\Debug\PA2_exe
第96秒内产生了2859个请求
系统B处理了1000个请求
系统C处理了1000个请求
消息队列中有6495个请求

第97秒内产生了594个请求
系统B处理了1000个请求
系统C处理了1000个请求
消息队列中有5089个请求

第98秒内产生了2696个请求
系统B处理了1000个请求
系统C处理了1000个请求
消息队列中有5785个请求

第99秒内产生了1788个请求
系统B处理了1000个请求
系统C处理了1000个请求
消息队列中有5573个请求

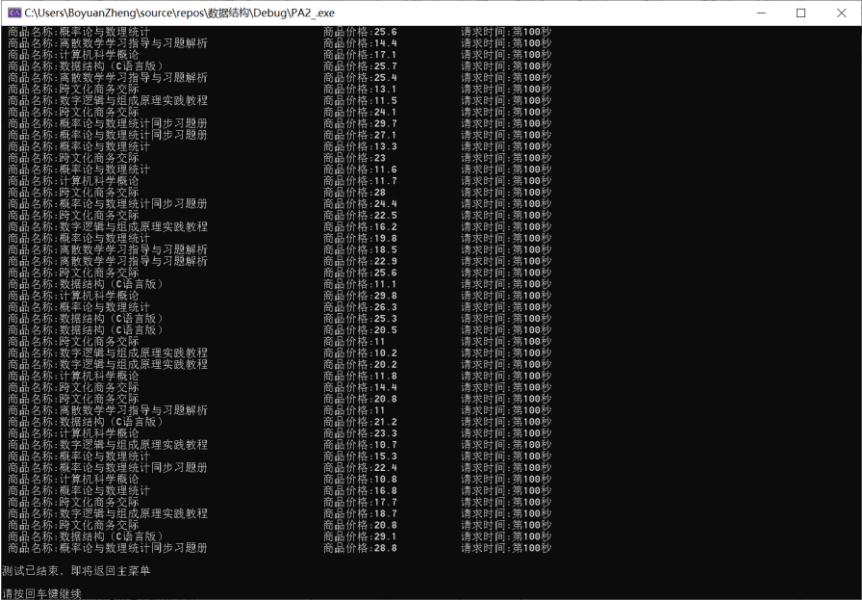
第100秒内产生了621个请求
系统B处理了1000个请求
系统C处理了1000个请求
消息队列中有4194个请求

总共成功处理请求192160个
平均每位用户等待请求被处理花费1.99556秒

请按回车键继续
```

程序展示每秒内的请求个数，以及系统 B、C 的处理情况和消息队列中等待被处理的请求数量如上图。

可以观察到，100 秒内大部分用户的请求能在 2 秒内被解决，除截止到 100 秒时积压在消息队列中的请求，其他请求均成功及时地得到了响应。程序输出 100 秒结束时在队列中的请求信息如下：



各个菜单项的功能执行完毕后，按回车键均能够返回主菜单进行下一轮功能项的选择。

选择菜单项 0，程序退出结束。



<p>调试分析</p>	<p>1. 循环队列不同于链表的实现方式，需要指定动态内存申请时数组申请的最大长度。本题中，我一开始选择用宏定义的方式来指定数组的长度。由于循环队列中需要空出一个元素的位置来区分队列的空、满两种不同状态，因此实际上能使用的队列有效最大长度与动态内存申请时的数组长度应该大小差一。为了程序的可读性，我将宏定义写作如“<code>#define MAXSIZEQ 3000 + 1</code>”的形式。但由于宏定义是在预编译阶段进行替换，<code>3000 + 1</code>这一表达式的优先级并非最高，导致了动态内存申请时的一系列错误。考虑到队列的有效长度在不同情况下由使用者构造时给定会更为合理方便，最后将 <code>MAXSIZE</code> 作为变量放入 <code>SqQueue</code> 队列的成员当中，并且在 <code>SqQueue</code> 队列构造时作为一个参数传入，这便解决了该问题；</p> <p>2. 由于想要让队列在代码精简的情况下适应更多不同的使用场景，我使用了模板类封装队列这一数据结构。起初，我想要将成员函数的定义、声明分别放在“<code>queue.h</code>”、“<code>queue.cpp</code>”两个文件中，使得模板类声明头文件和实现文件分离，但会在运行时出现编译错误。实际上这是因为，C++中每一个对象所占用的空间大小都必须在编译阶段被唯一的确定。但由于模板类的数据元素不确定，在它被真正定义使用前，编译器都无法确切地知道模板类中使用模板类型的对象的所占用的空间大小。只有模板被真正使用的时候，编译器才知道模板套用的类型以及其所一共被分配的空间大小。因此，我想要分头文件的方式是无法实现的。最后，我将 <code>cpp</code> 中的函数定义部分也全部移入 <code>.h</code> 的头文件当中，将其改名为“<code>queue.hpp</code>”。由于在编译时编译器能够确切知晓每一个对象的空间大小，因此就不存在编译错误了。</p>
<p>心得体会</p>	<p>本次实验中，我尝试用 C++方式实现队列，将操作其的函数都封装在类中。同时为了使封装之后的类能够适应不同的数据类型，我使用模板类，使其能够以诸如本题中“用户请求信息”（包含商品名称、商品价格、请求发起的时间等）的结构体之类的不同数据类型为其队列中的元素。</p>

	<p>为了更好的理解和完成本题，我了解了解耦、异步、限流消峰的具体含义，理解了消息队列在其中的作用。解耦指的是，当各个系统耦合度很高的情况下，每个系统之间的输入与输出交叉耦合使得各个系统的设计变的繁琐。如果系统之间通过消息队列进行信息传递，即将其他系统所需的信息放入消息队列中，对应的系统再在队列中取得信息，便可以实现解耦。在应用异步之前，若一个请求所带来的系列流程很长，那么用户就要等待相应长的时间，这样的体验是不可以容忍的。引入消息队列之后，便可以减少用户的等待时间，让对应的系统在队列里取请求，而无需用户等待。限流消峰与本题的题目比较类似，即通过队列防止大量的请求堵塞或损坏服务器，在本题的程序中体现的很清楚。</p> <p>本题中的消息队列的实现，使用的就是队列这一数据结构经典的实现方式。其中队列的入队、出队、读队首元素时间复杂度均为 $O(1)$；获取队列长度、判断是否为空时间复杂度也为 $O(1)$；逐个打印队列中的所有元素时间复杂度为 $O(n)$。</p>
--	---

附.完整代码

1.用模板类实现队列的 hpp 文件 (queue.hpp)

```
#pragma once
#include <iostream>
using namespace std;

#define OK 0
#define QERROR -1
#define QOVERFLOW -2
typedef int Status;

template <class QElemType>
class SqQueue {
private:
    QElemType* base;
```

```

    int MAXQSIZE;
    int front;
    int rear;
public:
    SqQueue(int maxqsize);        //建立空队列（构造函数）
    ~SqQueue();                  //销毁队列（析构函数）
    int QueueLength();           //获取队列长度
    Status EnQueue(QElemType e); //新元素入队
    Status DeQueue(QElemType& e); //队首元素出队
    Status GetHead(QElemType& e); //获取队首元素
    Status ClearQueue(); //清空队列
    bool QueueEmpty();           //判断队列是否为空
    Status PrintQueue(ostream& out); //从队首逐一输出元素
};

template <class QElemType>
SqQueue <QElemType>::SqQueue(int maxqsize)
{
    MAXQSIZE = maxqsize + 1; //空一个位置以区分满和空
    base = new(nothrow) QElemType[MAXQSIZE];
    if (!base)
        exit(QOVERFLOW);
    front = rear = 0;
}

template <class QElemType>
SqQueue <QElemType>::~~SqQueue()
{
    if (base)
        delete base;
    front = rear = 0;
}

template <class QElemType>
Status SqQueue <QElemType>::ClearQueue()
{
    front = rear = 0;
    return OK;
}

template <class QElemType>
int SqQueue <QElemType> ::QueueLength()
{

```

```

        return (rear - front + MAXQSIZE) % MAXQSIZE;
    }

template <class QElemType>
Status SqQueue <QElemType> ::EnQueue(QElemType e)
{
    if ((rear + 1) % MAXQSIZE == front)
        return QERROR;
    base[rear] = e;
    rear = (rear + 1) % MAXQSIZE;
    return OK;
}

template <class QElemType>
Status SqQueue <QElemType> ::DeQueue(QElemType& e)
{
    if (front == rear)
        return QERROR;
    e = base[front];
    front = (front + 1) % MAXQSIZE;
    return OK;
}

template <class QElemType>
Status SqQueue <QElemType> ::GetHead(QElemType& e)
{
    if (front == rear)
        return QERROR;
    e = base[front];
    return OK;
}

template <class QElemType>
bool SqQueue <QElemType>::QueueEmpty()
{
    return front == rear;
}

template <class QElemType>
Status SqQueue<QElemType>::PrintQueue(ostream& out)
{
    if (front == rear) {
        out << "Queue is Empty." << endl;
    }
}

```

```

        return QERROR;
    }

    for (int i = front; i != rear; i = (i + 1) % MAXQSIZE)
        out << base[i] << " ";
    out << endl;
    return OK;
}

```

2.测试程序的主源文件 (main.cpp)

```

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>
#include <Windows.h>
#include <iomanip>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include "../queue.hpp"
using namespace std;
//模拟的商品名称
const char* CommodityNameList[] = { "数据结构（C语言版）\t", "离散数学学习指导与习题解析", "数字逻辑与组成原理实践教程", "概率论与数理统计\t", "概率论与数理统计同步习题册", "跨文化商务交际\t", "计算机科学概论\t" };
struct Data {
    char CommodityName[128];    //商品名称
    float Price;                //商品价格
    int RequestTime;            //请求的时间
    friend ostream& operator<<(ostream& out, const Data& that);
};

ostream& operator<<(ostream& out, const Data& that)
{
    cout << "商品名称:" << that.CommodityName << "\t\t\t商品价格:" << that.Price << "\t\t\t请求时间:第" << that.RequestTime << "秒" << endl;
    return out;
}

```

```

void wait_for_enter()
{
    cout << endl << "请按回车键继续";
    while (getchar() != '\n')
        ;
    cout << endl << endl;
}

int Menu()
{
    system("mode con: cols=83 lines=30");
    cout << endl << endl << endl << endl;
    cout << "\t\t\t\t PA2*-队列的应用" << endl;

    cout << endl << endl;
    cout << "\t\t\t\t 菜单选择" << endl;
    cout << '\t' << setw(65) << setfill('-') << "" << endl;
    cout << "\t|\t\t|\t\t|\t\t|\t\t|\t\t|" << endl;
    cout << "\t|\t1\t|\t2\t|\t3\t|\t0\t|" << endl;
    cout << "\t|\t\t|\t\t|\t\t|\t\t|" << endl;
    cout << '\t' << setw(65) << setfill('-') << "" << endl;
    cout << "\t|\t\t|\t\t|\t\t|\t\t|" << endl;
    cout << "\t| 测试队列\t| 无消息队列\t| 有消息队列\t| 退出\t|"
<< endl;
    cout << "\t| 基础功能\t| 场景模拟\t| 场景模拟\t| QUIT\t|"
<< endl;
    cout << "\t|\t\t|\t\t|\t\t|\t\t|" << endl;
    cout << '\t' << setw(65) << setfill('-') << "" << endl << endl <<
endl;
    cout << "\t\t\t\t [请按对应数字选择功能]" << endl;
    cout << "\t\t\t\t\t";
    while (char ch = _getch())
        if (ch >= '0' && ch <= '3')
            return ch - '0';
    return 0;
}

void TestBasicFunction()
{
    system("cls");
    int e;
    cout << "下面开始测试SqQueue的各个功能" << endl;
    cout << "-----" << endl;

```

```

cout << endl << "<测试构造函数>" << endl;
cout << "请输入队列的最大长度:";
int maxsize;
cin >> maxsize;
while (!cin.good()) {
    cout << "输入有误, 请重新输入:";
    cin.clear();
    cin.ignore(65536, '\n');
    cin >> maxsize;
}
getchar();
{
    SqQueue<int> queue(maxsize);
    cout << "最大长度为" << maxsize << "的空队列建立成功" << endl;
    wait_for_enter();

#if 1
    cout << endl << "<测试Enqueue功能>" << endl;
    for (int i = 0; i <= maxsize + 2; i++) {
        if (queue.Enqueue(i * 2 + 1) == OK)
            cout << "元素(" << i * 2 + 1 << ")插入完毕" << endl;
        else
            cout << "元素(" << i * 2 + 1 << ")插入失败" << endl;
    }
    wait_for_enter();
#endif

#if 1
    cout << endl << "<测试Dequeue功能>" << endl;
    for (int i = 0; i <= maxsize + 2; i++) {
        if (queue.DeQueue(e) == OK)
            cout << "元素(" << e << ")已成功出队" << endl;
        else
            cout << "队列已空, 出队失败!" << endl;
    }
    wait_for_enter();
#endif

#if 1

```

```

cout << endl << "<测试GetHead功能>" << endl;
cout << "先将队列重新填充" << endl;
for (int i = 0; i <= maxsize + 2; i++) {
    if (queue.Enqueue(i * 2 + 1) == OK)
        cout << "元素(" << i * 2 + 1 << ")插入完毕" << endl;
    else
        cout << "元素(" << i * 2 + 1 << ")插入失败" << endl;
}

cout << endl << "填充完毕, 下面开始测试" << endl;
for (int i = 0; i <= maxsize + 2; i++) {
    if (queue.GetHead(e) == OK)
        cout << "元素(" << e << ")当前在队首" << endl;
    else
        cout << "队列已空, 队首获取失败!" << endl;
    if (queue.DeQueue(e) == OK)
        cout << "元素(" << e << ")已成功出队" << endl;
    else
        cout << "队列已空, 出队失败!" << endl;
}
wait_for_enter();
#endif

#if 1
cout << endl << "<测试PrintQueue功能>" << endl;
cout << "先将队列重新填充" << endl;
for (int i = 0; i <= maxsize + 2; i++) {
    if (queue.Enqueue(i * 2 + 1) == OK)
        cout << "元素(" << i * 2 + 1 << ")插入完毕" << endl;
    else
        cout << "元素(" << i * 2 + 1 << ")插入失败" << endl;
}
cout << "队列内元素cout打印如下:" << endl;
queue.PrintQueue(cout);
cout << "下面测试输出到文件,";
ofstream out;
out.open("test.txt", ios::out);
out << "队列内元输出到文件打印如下:" << endl;
queue.PrintQueue(out);
cout << "请打开可执行文件所在目录检查test.txt文件中输出" << endl;
out.close();
wait_for_enter();

```



```

#endif

#if 1
    cout << endl << "<测试ClearQueue功能>" << endl;
    cout << "下面开始清空队列" << endl;
    queue.ClearQueue();
    cout << "队列清空完毕，队列内元素打印如下：" << endl;
    queue.PrintQueue(cout);
    wait_for_enter();
#endif

#if 1
    cout << endl << "<测试QueueEmpty功能>" << endl;
    cout << "当前队列状态是：" << (queue.QueueEmpty() ? "空" : "非空") << endl;
    cout << "将队列重新填充" << endl;
    for (int i = 0; i <= maxsize + 2; i++) {
        if (queue.Enqueue(i * 2 + 1) == OK)
            cout << "元素(" << i * 2 + 1 << ")插入完毕" << endl;
        else
            cout << "元素(" << i * 2 + 1 << ")插入失败" << endl;
    }
    cout << "当前队列状态是：" << (queue.QueueEmpty() ? "空" : "非空") << endl;
    wait_for_enter();
#endif

#if 1
    cout << endl << "<测试QueueLength功能>" << endl;
    cout << "当前队列长度是：" << queue.QueueLength() << endl;
    for (int i = 0; i <= maxsize + 2; i++) {
        if (queue.DeQueue(e) == OK)
            cout << "元素(" << e << ")已成功出队" << endl;
        else
            cout << "队列已空，出队失败!" << endl;
        cout << "当前队列长度是：" << queue.QueueLength() << endl;
    }
    wait_for_enter();
#endif

```

```

        cout << endl << "<测试析构函数>" << endl;
    }
    cout << "Queue队列已经被销毁成功" << endl;
    wait_for_enter();

    cout << "队列功能测试完毕，按任意键返回主菜单";
    getchar();
}

void Simulation1()
{
    system("cls");
    int TotalMiss = 0;
    cout << "即将开始模拟100秒内的请求" << endl;
    wait_for_enter();
    for (int i = 1; i <= 100; i++) {
        int request = rand() % 3000 + 500;
        cout << endl << "第" << i << "秒内产生了" << request << "个请求"
<< endl;
        if (request >= 1000) {
            request -= 1000;
            cout << "B系统处理1000个请求" << endl;
            if (request >= 1000) {
                request -= 1000;
                cout << "C系统处理1000个请求" << endl;
            }
            else {
                cout << "C系统处理" << request << "个请求" << endl;
                request = 0;
            }
        }
        else {
            cout << "B系统处理" << request << "个请求" << endl;
            request = 0;
        }
        if(request)
            cout << "本秒有" << request << "个请求被阻塞" << endl;
        TotalMiss += request;
        Sleep(100);
    }
    cout << endl << endl << "总共有" << TotalMiss << "条请求被阻塞" << endl;
    wait_for_enter();
}

```

```

}

void Simulation2()
{
    system("cls");
    SqQueue<Data> MessageQueue(10000);
    int RequestTot = 0, WaitSum = 0;
    cout << "即将开始模拟100秒内的请求" << endl;
    wait_for_enter();
    for (int i = 1; i <= 100; i++) {
        int request = rand() % 3000 + 500;
        cout << endl << "第" << i << "秒内产生了" << request << "个请求"
        << endl;
        Data e;
        for (int j = 0; j < request; j++) {
            strcpy(e.CommodityName, CommodityNameList[rand() % 7]);
            //随机模拟商品名称
            e.Price = (rand() % 200 + 100) / 10.0f;
            //随机模拟订单价格
            e.RequestTime = i; //请求的时间点
            MessageQueue.Enqueue(e);
        }

        //系统开始处理请求
        int j;
        for (j = 1; j <= 1000; j++) {
            if (MessageQueue.DeQueue(e) != OK)
                break;
            //队列已经为空
            RequestTot++;
            WaitSum += i - e.RequestTime; //等待时间
        }
        cout << "系统B处理了" << j - 1 << "个请求" << endl;

        for (j = 1; j <= 1000; j++) {
            if (MessageQueue.DeQueue(e) != OK)
                break;
            //队列已经为空
            RequestTot++;
            WaitSum += i - e.RequestTime; //等待时间
        }
        cout << "系统C处理了" << j - 1 << "个请求" << endl;
        cout << "消息队列中有" << MessageQueue.QueueLength() << "个请求"

```

```

<< endl;
    Sleep(100);
}

cout << endl << endl << "总共成功处理请求" << RequestTot << "个" << endl;
cout << "平均每位用户等待请求被处理花费" << WaitSum * 1.0f / RequestTot
<< "秒" << endl;
wait_for_enter();

system("mode con: cols=150 lines=50");
cout << "截止到第100秒, 还有" << MessageQueue.QueueLength() << "个
请求未被处理, 具体如下:" << endl;
wait_for_enter();
MessageQueue.PrintQueue(cout);

cout << "测试已结束, 即将返回主菜单" << endl;
wait_for_enter();
}

int main()
{
    int ret;
    srand((unsigned)time(NULL));
    while (ret = Menu()) {
        switch (ret){
            case 1:
                TestBasicFunction();
                break;
            case 2:
                Simulation1();
                break;
            case 3:
                Simulation2();
                break;
        }
    }

    return 0;
}

```