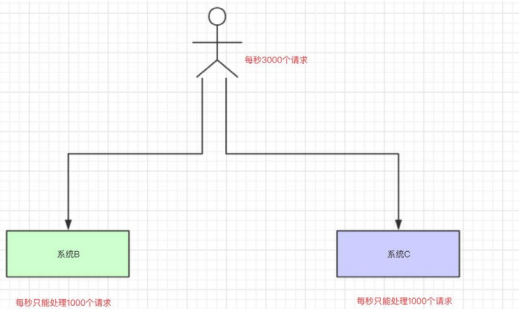
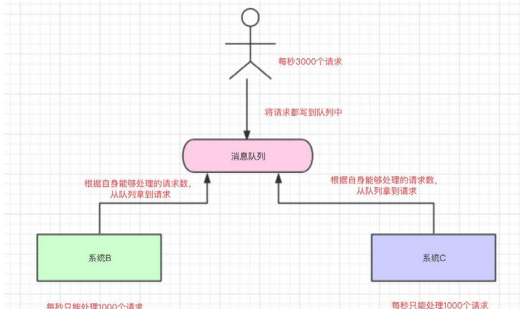


《数据结构》上机报告

2020 年 10 月 27 日

姓名： 林日中 学号： 1951112 班级： 10072602 得分： _____

实验题目	队列的应用实验报告 - 消息队列	
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> <div style="display: flex; justify-content: space-around;">   </div> <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 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> <p>实验目的：</p> <ol style="list-style-type: none"> 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） 	
基本要求	<p>定义顺序队列类型，使其具有如下功能：</p> <ol style="list-style-type: none"> (1) 建立一个空队列； (2) 释放队列空间，将队列销毁； (3) 将队列清空，变成空队列； (4) 判断队列是否为空； (5) 返回队列内的元素个数； (6) 将队头元素弹出队列（出队）； (7) 在队列中加入一个元素（入队）； (8) 从队头到队尾将队列中的元素依次输出。 	
	已完成基本内容（序号）：	(1)(2)(3)(4)(5)(6)(7)(8)
选做要求	<p>已完成选做内容（序号）：</p> <p>无</p>	

<p>数据结构设计</p>	<p>在本次上机实验中，我设计了两个数据结构：struct MyMessage 和 struct MyQueue，分别表示队列中存储的基础数据类型“信息”和队列。</p> <p>struct MyMessage 存储了指向通知内容的字符指针 char *content，表示创建通知的时间的字符串 char time_str[]，存储从 1970 年到创建通知时经过了多少秒的 time_t t。</p> <p>struct MyStack 存储了指向动态分配的基础数据类型数组 QElemType *base；栈的容量 int _size，队头队尾指针 front 和 rear；队列最大容量 max_queue_size。</p>
<p>功能(函数)说明</p>	<div data-bbox="352 667 971 1202" data-label="Text"> <pre>typedef struct MyMessage { private: char *content = nullptr; char time_str[32] = {'\0'}; time_t t; time_t get_time()... friend istream &operator>>(istream &, MyMessage &); friend ostream &operator<<(ostream &, const MyMessage &); public: MyMessage() {} MyMessage(const char *c) ... MyMessage(const MyMessage &m) ... ~MyMessage() ... MyMessage &set(const char *c) ... MyMessage &operator=(const MyMessage &m) ... const char *get_content() const { return content; } operator bool() const { return content != nullptr; } } QElemType; // struct MyMessage istream &operator>>(istream &in, MyMessage &m) ... ostream &operator<<(ostream &out, const MyMessage &m) ...</pre> </div> <p>struct MyMessage 的构造函数实现了构造空 MyMessage 对象、由字符串或已有的 MyMessage 对象构造新对象的功能；析构函数对 content 进行 delete 操作。set() 函数依据传入的字符串给对象的 content 赋新值，并重新设置生成时间。对运算符=的重载函数实现了根据已有对象给本对象赋值的功能。get_content() 和 operator bool() 函数分别返回指向通知内容的字符指针和 content 是否为空指针的布尔值。对运算符>>和<<的重载函数实现了便捷输入、输出对象内容的功能。</p> <div data-bbox="352 1464 1302 1982" data-label="Text"> <pre>struct MyQueue { protected: QElemType *base = nullptr; int _size = 0; int front = 0; int rear = 0; int max_queue_size = 0; public: MyQueue(const int s = 100) : base(new QElemType[s]), max_queue_size(s) {} ~MyQueue() { delete[] base; } int size() const { return _size; } // = (rear - front + max_queue_size) % max_queue_size; bool push(const QElemType &e) ... QElemType pop() ... int pop(MyQueue &q, int n = -1) ... QElemType top() const { return front == rear ? QElemType() : base[front]; } bool empty() const { return front == rear; } bool full() const { return _size == max_queue_size; } void traverse() ... }; // struct MyQueue</pre> </div>

	<p><code>struct MyQueue</code> 的构造函数能根据传入参数构造一个 <code>MyQueue</code> 对象，其最大容量为(传输参数的值 + 1)；析构函数能对 <code>base</code> 指针指向的内容进行 <code>delete</code> 操作。<code>size()</code> 函数能够返回队列中的元素个数。<code>push()</code> 和 <code>pop(void)</code> 函数分别实现了在队尾压入元素和弹出队头元素的功能；<code>pop(MyQueue &q, int)</code> 函数实现了从本对象中弹出元素插入到第一个参数的尾部，第二个参数表示要提取的数量，若为-1 则提取的元素个数为 <code>min(*this</code> 中剩余元素个数, <code>q</code> 中剩余空位个数)。 <code>top()</code> 函数能够返回队头元素。<code>empty()</code> 和 <code>full()</code> 分别返回表示队列是否为空、是否为满的布尔值。</p>
开发环境	Windows 10, C++ language, Visual Studio Code with g++
调试分析	<pre>int test_myqueue(const int total, int cap_a, int cap_b) { cout << "\n<<< This is a test function >>>\n\n"; int message_capacity = 60000, messages_per_sec = 6000, seq = 0, count = total; MyQueue notice_queue(message_capacity + 1), a(cap_a + 1), b(cap_b + 1); clock_t start = clock(); do... cout << "The process ended in " << (double)(clock() - start) / CLOCKS_PER_SEC << " second(s). " << endl; return 0; }</pre> <p><code>test_myqueue()</code> 函数封装了测试队列功能的操作：根据用户键入的两个较小消息队列的容量，模拟两个消息队列从主队列中提取消息并处理的过程。在此过程中，程序会输出每秒弹出的第一个元素的信息。</p> <p>经过测试，本程序功能完好。</p> <p>本程序输出的所有文本如下：</p> <pre>Welcome! This program is committed to realizing the process of two sub-queues grabbing messages from a main notification queue. There is a main notification queue generating 6,000 messages per second, totally 60,000. Please enter the capacities of the two queues within [2000..3000]: 3000 3000< <<< This is a test function >>> a.pop() : Content : hello world - 00000 Time : Tuesday 11/03/20 19:04:26 b.pop() : Content : hello world - 03000 Time : Tuesday 11/03/20 19:04:26 a.pop() : Content : hello world - 06000 Time : Tuesday 11/03/20 19:04:27 b.pop() : Content : hello world - 09000 Time : Tuesday 11/03/20 19:04:27 a.pop() : Content : hello world - 12000 Time : Tuesday 11/03/20 19:04:28 b.pop() : Content : hello world - 15000 Time : Tuesday 11/03/20 19:04:28 a.pop() : Content : hello world - 18000 Time : Tuesday 11/03/20 19:04:29 b.pop() : Content : hello world - 21000 Time : Tuesday 11/03/20 19:04:29 a.pop() : Content : hello world - 24000 Time : Tuesday 11/03/20 19:04:30 b.pop() : Content : hello world - 27000 Time : Tuesday 11/03/20 19:04:30</pre>

```
a.pop() :  
Content : hello world - 30000  
Time   : Tuesday 11/03/20 19:04:31  
b.pop() :  
Content : hello world - 33000  
Time   : Tuesday 11/03/20 19:04:31  
  
a.pop() :  
Content : hello world - 36000  
Time   : Tuesday 11/03/20 19:04:32  
b.pop() :  
Content : hello world - 39000  
Time   : Tuesday 11/03/20 19:04:32  
  
a.pop() :  
Content : hello world - 42000  
Time   : Tuesday 11/03/20 19:04:33  
b.pop() :  
Content : hello world - 45000  
Time   : Tuesday 11/03/20 19:04:33  
  
a.pop() :  
Content : hello world - 48000  
Time   : Tuesday 11/03/20 19:04:34  
b.pop() :  
Content : hello world - 51000  
Time   : Tuesday 11/03/20 19:04:34  
  
a.pop() :  
Content : hello world - 54000  
Time   : Tuesday 11/03/20 19:04:36  
b.pop() :  
Content : hello world - 57000  
Time   : Tuesday 11/03/20 19:04:36  
  
The process ended in 10.842 second(s).
```

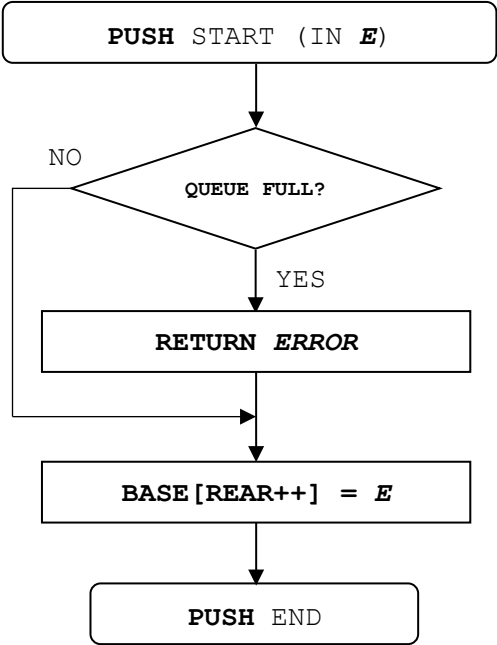
综上，本实验设计的数据结构很好地完成了题目对队列的功能的要求，功能完好，使用方便，并且能够较好地处理非法数据的输入并反馈相关错误提示；程序界面友好，具有比较恰当的人性化提醒，具有一定的鲁棒性。

一、实验总结

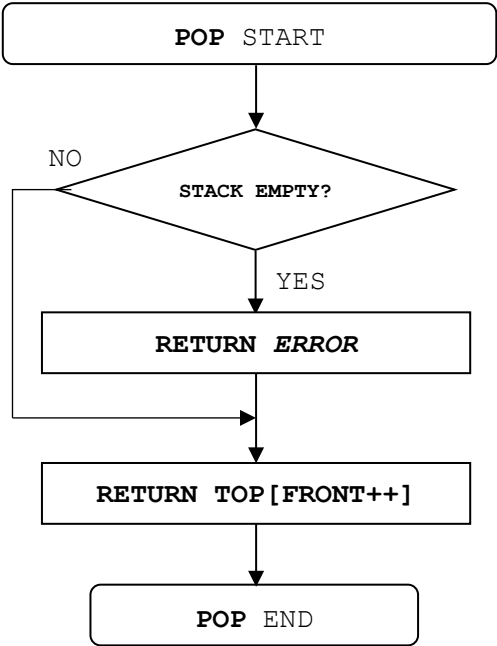
在本次上机实验中，我复习了理论课上学到的队列的定义和作用，将课本上队列的顺序存储结构和与其相关的建立、压入元素、弹出元素等函数样例封装成了 `struct MyQueue` 结构体。

二、性能分析

- (1) 队列元素的压入
时间复杂度为 $O(1)$ 。
流程图如下：



- (2) 队列元素的弹出
时间复杂度为 $O(1)$ 。
流程图如下：



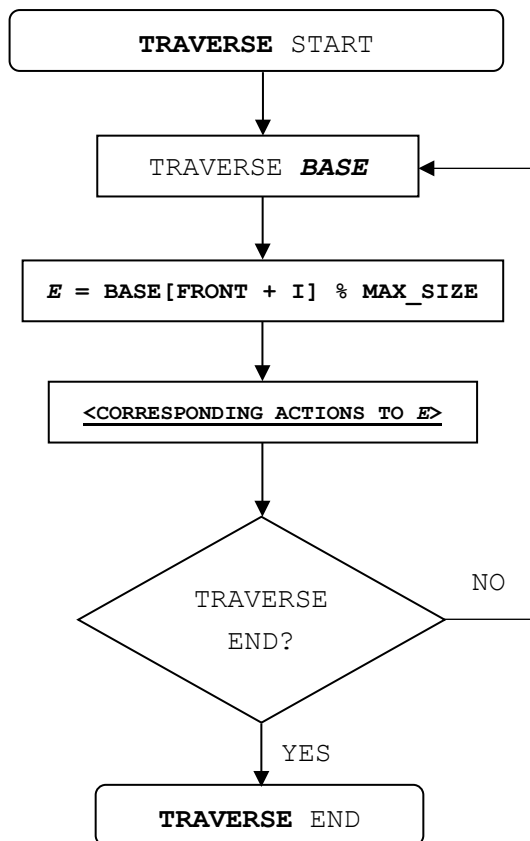
心得体会

而一次压入、弹出多个元素的操作，本质是 n 个 **push** 操作和 n 个 **pop** 操作。其时间复杂度为 $O(n)$ 。

(3) 队列的遍历

时间复杂度为 $O(n)$ 。

流程图如下：

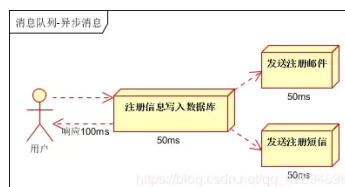
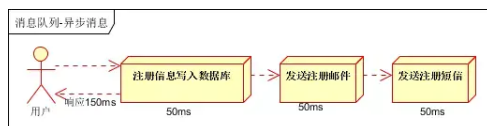


三、 消息队列的使用场景

(1) 异步处理

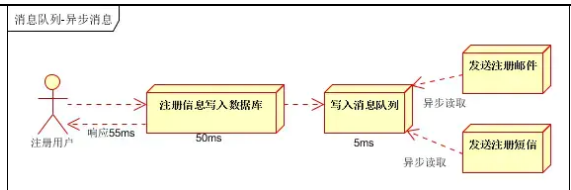
用户注册后，需要发注册邮件和注册短信。传统的做法有两种：串行的方式和并行方式。

串行方式：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户。



并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间。

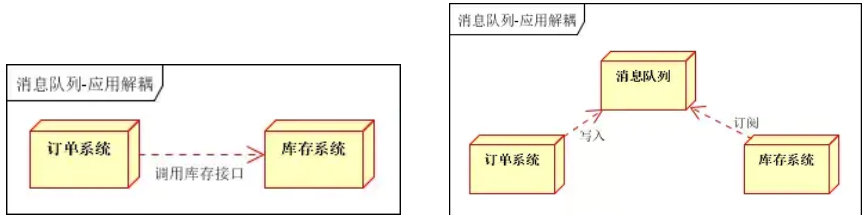
假设三个业务节点每个使用 50 毫秒，不考虑网络等其他开销，则串行方式的时间是 150 毫秒，并行的时间可能是 100 毫秒。那么，串行方式下系统的吞吐量是每秒 7 QPS，串行方式下系统的吞吐量是每秒 10 QPS。改造后的**异步处理**架构如下：



按照以上约定，用户的响应时间相当于是注册信息写入数据库的时间，也就是 50 毫秒。注册邮件，发送短信写入消息队列后，直接返回，因此写入消息队列的速度很快，基本可以忽略，因此用户的响应时间可能是 50 毫秒。因此架构改变后，系统的吞吐量提高到每秒 20QPS。比串行提高了 3 倍，比并行提高了两倍！

(2) 应用解耦

用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。如左下图。



假如库存系统无法访问，则订单减库存将失败，从而导致订单失败，订单系统与库存系统耦合。引入应用消息队列后的方案，如右上图。

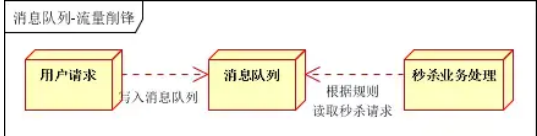
订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功

库存系统：订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作

假如在下单时库存系统不能正常使用，正常下单也不会被影响。因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。这样就实现了订单系统与库存系统的应用解耦。

(3) 限流消峰

电商秒杀活动的 APP 有可能会因为流量暴增挂掉。为解决这个问题，一般需要在应用前端加入消息队列，可以缓解短时间内高流量压垮应用。



用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面。系统会根据消息队列中的请求信息再做后续处理。

源代码

```
1.  #ifdef __GNUC__
2.  #include <bits/stdc++.h>
3.  #endif // __GNUC__
4.  #ifdef _MSC_VER
5.  #define _CRT_SECURE_NO_WARNINGS
6.  #include <iostream>
7.  #include <cstdio>
8.  #include <cstdlib>
```

```

9.  #include <cstring>
10. #include <ctime>
11. #include <queue>
12. #endif // _MSC_VER
13. #include <windows.h>
14. using namespace std;
15.
16. typedef struct MyMessage
17. {
18.     private:
19.         char *content = nullptr;
20.         char time_str[32] = {'\0'};
21.         time_t t;
22.         time_t get_time()
23.         {
24.             t = time(nullptr);
25.             // locale::global(locale("en.US-utf8"));
26.             std::strftime(time_str, sizeof(time_str), "%A %c", std::localtime(&t));
27.             return t;
28.         }
29.         friend istream &operator>>(istream &, MyMessage &);
30.         friend ostream &operator<<(ostream &, const MyMessage &);
31.
32.     public:
33.         MyMessage() {}
34.         MyMessage(const char *c)
35.         {
36.             if (c)
37.             {
38.                 content = new char[strlen(c) + 1];
39.                 strcpy(content, c);
40.             }
41.             get_time();
42.         }
43.         MyMessage(const MyMessage &m)
44.         {
45.             if (m.content)
46.             {
47.                 content = new char[strlen(m.content) + 1];
48.                 strcpy(content, m.content);
49.             }
50.             get_time();
51.         }
52.         ~MyMessage()

```



```

53.     {
54.         if (content)
55.         {
56.             delete[] content;
57.             content = nullptr;
58.         }
59.     }
60.     MyMessage &set(const char *c)
61.     {
62.         delete[] content;
63.         content = nullptr;
64.         if (c)
65.         {
66.             content = new char[strlen(c) + 1];
67.             strcpy(content, c);
68.         }
69.         get_time();
70.         return *this;
71.     }
72.     MyMessage &operator=(const MyMessage &m)
73.     {
74.         if (content)
75.         {
76.             delete[] content;
77.             content = nullptr;
78.         }
79.         if (m.content)
80.         {
81.             content = new char[strlen(m.content) + 1];
82.             strcpy(content, m.content);
83.         }
84.         get_time();
85.         return *this;
86.     }
87.     const char *get_content() const { return content; }
88.     operator bool() const { return content != nullptr; }
89. } QElemType; // struct MyMessage
90. istream &operator>>(istream &in, MyMessage &m)
91. {
92.     if (m.content)
93.     {
94.         delete[] m.content;
95.         m.content = nullptr;
96.     }

```

```

97.     char tmp[1024] = {'\0'};
98.     in >> tmp;
99.     m.content = new char[strlen(tmp) + 1];
100.    strcpy(m.content, tmp);
101.    return in;
102.}

103.ostream &operator<<(ostream &out, const MyMessage &m)
104.{
105.    return out << "Content : " << (m.content ? m.content : "<empty>") << std::endl
106.        << "Time      : " << m.time_str;
107.}
108.
109.struct MyQueue
110.{
111.protected:
112.    QElemType *base = nullptr;
113.    int _size = 0;
114.    int front = 0;
115.    int rear = 0;
116.    int max_queue_size = 0;
117.
118.public:
119.    MyQueue(const int s = 100) : base(new QElemType[s]), max_queue_size(s) {}
120.    ~MyQueue() { delete[] base; }
121.    int size() const { return _size; } // = (rear - front + max_queue_size) % max_queue_size;

122.    bool push(const QElemType &e)
123.    {
124.        if ((rear + 1) % max_queue_size == front)
125.        {
126.            return false;
127.        }
128.        base[rear] = e;
129.        rear = (rear + 1) % max_queue_size;
130.        _size++;
131.        return true;
132.    }
133.    QElemType pop()
134.    {
135.        QElemType result;
136.        if (front == rear)
137.        {
138.            return QElemType();
139.        }

```

```

140.         result = base[front];
141.         front = (front + 1) % max_queue_size;
142.         _size--;
143.         return result;
144.     }
145.     int pop(MyQueue &q, int n = -1)
146.     {
147.         if (n == -1 || n > (q.max_queue_size - q.size() - 1) || n > this->size())
148.         {
149.             n = min(this->size(), q.max_queue_size - q.size() - 1);
150.         }
151.         for (int i = n; i; i--)
152.         {
153.             if (this->top())
154.             {
155.                 q.push(this->pop());
156.             }
157.         }
158.         return n;
159.     }
160.     QElemType top() const { return front == rear ? QElemType() : base[front]; }
161.     bool empty() const { return front == rear; }
162.     bool full() const { return _size == max_queue_size; }
163.     void traverse()
164.     {
165.         int len = size();
166.         for (int i = 0; i < len; i++)
167.         {
168.             cout << base[(front + i) % max_queue_size] << endl;
169.         }
170.     }
171. }; // struct MyQueue
172.
173. int test_myqueue(const int total, int cap_a, int cap_b)
174. {
175.     cout << "\n<<< This is a test function >>>\n\n";
176.     int message_capacity = 60000, messages_per_sec = 6000, seq = 0, count = total;
177.     MyQueue notice_queue(message_capacity + 1), a(cap_a + 1), b(cap_b + 1);
178.     clock_t start = clock();
179.
180.     do
181.     {
182.         if (seq < total)
183.         { // generate elements of notice queue

```

```
184.         char str[] = "hello world - ????" ;
185.         for (int i = 0; i < messages_per_sec && seq < total; i++, seq++)
186.         {
187.             sprintf(str, "hello world - %05i", seq);
188.             notice_queue.push(str);
189.         }
190.     }
191.     if (!notice_queue.empty())
192.     {
193.         notice_queue.pop(a);
194.     }
195.     if (!notice_queue.empty())
196.     {
197.         notice_queue.pop(b);
198.     }
199.     if (!a.empty())
200.     {
201.         cout << "a.pop() : \n"
202.              << a.pop() << endl;
203.         count--;
204.     }
205.     if (!b.empty())
206.     {
207.         cout << "b.pop() : \n"
208.              << b.pop() << endl
209.              << endl;
210.         count--;
211.     }
212.     while (!a.empty())
213.     {
214.         a.pop();
215.         count--;
216.     }
217.     while (!b.empty())
218.     {
219.         b.pop();
220.         count--;
221.     }
222.     Sleep(1000);
223. } while (count > 0);
224. cout << "The process ended in " << (double)(clock() - start) / CLOCKS_PER_SEC << " second
(s). " << endl;
225. return 0;
226. }
```

```
227.  
228. int main()  
229. {  
230.     cout << "Welcome! This program is committed to realizing the process of two sub-  
        queues grabbing messages from a main notification queue. " << endl  
231.         << "There is a main notification queue generating 6,000 messages per second, totally  
        60,000. Please enter the capacities of the two queues within [2000..3000]: ";  
232.  
233.     int cap_a, cap_b;  
234.     cin >> cap_a >> cap_b;  
235.  
236.     test_myqueue(60000, cap_a, cap_b);  
237.     system("pause");  
238.     return 0;  
239. }
```