|  |
| --- |
| hust1 |

|  |
| --- |
| **课 程 实 验 报 告** |

|  |
| --- |
| **课程名称： 面向对象程序设计** |
| **实验名称： 面向过程的整形栈编** |

|  |  |
| --- | --- |
| **专业班级：** | **计算机科学与技术201601** |
| **学 号：** | **U201614515`** |
| **姓 名：** | **吴阳民** |
| **指导教师：** | **马光志** |
| **报告日期：** | **2018年 11 月 23 日** |

|  |
| --- |
| **计算机科学与技术学院** |

目 录

[一、需求分析 1](#_Toc530339246)

[1.题目要求 1](#_Toc530339247)

[2.需求分析 2](#_Toc530339248)

[二、系统设计 2](#_Toc530339249)

[1.概要设计 2](#_Toc530339250)

[2.详细设计 2](#_Toc530339251)

[a.构造函数 2](#_Toc530339252)

[b.构造函数 2](#_Toc530339253)

[c.重载运算符() 3](#_Toc530339254)

[d.重载运算符[] 3](#_Toc530339255)

[e.重载运算符<< 3](#_Toc530339256)

[f.重载运算符>> 4](#_Toc530339257)

[g.重载运算符= 4](#_Toc530339258)

[h.打印 4](#_Toc530339259)

[三、软件开发 4](#_Toc530339260)

[四、软件测试 5](#_Toc530339261)

[五、特点与不足 7](#_Toc530339262)

[1.技术特点 7](#_Toc530339263)

[2.不足 7](#_Toc530339264)

[七、源码和说明 7](#_Toc530339265)

[1.文件清单及其功能说明 7](#_Toc530339266)

[2.用户使用说明书 8](#_Toc530339267)

[3.源代码 8](#_Toc530339268)

[a.queue\_ex\_6.h 8](#_Toc530339269)

[b.queue\_ex\_6.cpp 9](#_Toc530339270)

[c.U201614515\_4.h 14](#_Toc530339271)

[d.U201614515\_4.cpp 15](#_Toc530339272)

# 一、需求分析

## 1.题目要求

整型队列是一种先进后出的存储结构，对其进行的操作通常包括判断队列是否为空、向队列顶添加一个整型元素、出队列等。整型队列类型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函数采用C++编程。然后写一个main函数对队列的所有操作函数进行测试，要求main按照《关于C++实验自动验收系统说明》给定的方式工作。注意，请用实验三的SATCK继承形成新的类QUEUE。分析说明除构造函数以外的函数，加virtual说明与不加virtual说明有无区别。并说明为什么不将s2也作为基类。

整型队列是一种先进后出的存储结构，对其进行的操作通常包括判断队列是否为空、向队列顶添加一个整型元素、出队列等。整型队列类型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函数采用C++编程。然后写一个main函数对队列的所有操作函数进行测试，要求main按照《关于C++实验自动验收系统说明》给定的方式工作。注意，请用实验三的SATCK继承形成新的类QUEUE。分析说明除构造函数以外的函数，加virtual说明与不加virtual说明有无区别。并说明为什么不将s2也作为基类。

1. **class** QUEUE：**public** STACK{
2. STACK  s2;
3. **public**:
4. QUEUE(**int** m); //每个栈最多m个元素，要求实现的队列最多能入2m个元素
5. QUEUE(**const** QUEUE&s);           //用队列s拷贝初始化队列
6. **virtual** operator **int** ( ) **const**;         //返回队列的实际元素个数
7. **virtual** **int** full ( ) **const**;            //返回队列是否已满，满返回1，否则返回0
8. **virtual** **int** operator[ ](**int** x)**const**;   //取下标为x的元素，第1个元素下标为0
9. **virtual** QUEUE& operator<<(**int** e);  //将e入队列,并返回队列
10. **virtual** QUEUE& operator>>(**int** &e);    //出队列到e,并返回队列
11. **virtual** QUEUE& operator=(**const** QUEUE&s); //赋s给队列,并返回被赋值的队列
12. **virtual** **void** print( ) **const**;            //打印队列
13. **virtual** ~QUEUE( );                  //销毁队列
14. };

注意：虽然实现的队列 最多能入2m个元素，但是由于模拟中入出操作序列的限制，即操作不能违背先进先出的原则，可能还没入2m个元素队列就“满”了，所以要注意full( )函数的实现：不能简单判断队列是否装了2m个元素。

在完成上述程序及测试无误后，请使用队列解决如下舞伴问题，此时main用非命令行的方式工作。假定在一次舞会上，男士排成一队，女士排成另一队。每次舞曲响起时，从男队和女队的队头各出一人，配成舞伴跳完此曲，跳完后各自进入自己队列的尾部。若男女两队的初始人数分别为M和F（M和F均为素数，且M!=F），男队中排在位置m（m<=M）的男士，非常想和女队位置f（f<=F）的女士跳舞，问他在第几支曲舞曲才能和该女士跳舞?请编程解决上述问题。

## 2.需求分析

问题：为什么不将s2也作为基类。

如果将s2也作为基类，s1与s2属于同一对象类型，最终继承结果是只有一个栈，而再该程序中需要的是两个栈。

# 二、系统设计

## 1.概要设计

整型栈是一种先进后出的存储结构，对其进行的入队操作通常在队尾进行，对其进行的出队操作通常在队头进行。队列可以使用数组或者链表或者栈模拟来实现。判断队列是否为空、向队列顶添加一个整型元素、出队列等。

## 2.详细设计

### a.构造函数

QUEUE

入口参数：int m（m表示每个栈初始容量大小）

出口参数：无

返回值：无



### b.构造函数

QUEUE

入口参数：const QUEUE &s（待拷贝的队列的引用）

出口参数：无

返回值：无



### c.重载运算符()

int QUEUE:: operator int() const

入口参数：无

出口参数：无

返回值：队列中实际元素个数



### d.重载运算符[]

int QUEUE::operator[](int x) const

入口参数：下标x

出口参数：无

返回值：下标x处的元素



### e.重载运算符<<

QUEUE& QUEUE::operator<< (int e)

入口参数：e

出口参数：无

返回值：this



### f.重载运算符>>

QUEUE& QUEUE::operator>> (int &e)

入口参数：无

出口参数：e

返回值：this



### g.重载运算符=

QUEUE& QUEUE::operator= (const QUEUE&s)

入口参数：s

出口参数：无

返回值：无



### h.打印

void QUEUE::print() const

入口参数：无

出口参数：无

返回值：无



# 三、软件开发

软件开发环境：visual studio Enterprise 2017

软件复测环境：C++ Builder 2009

软件测试工具：checksystem\_v3.0

本地随机测试工具：cmd

# 四、软件测试

正在测试C:/Users/Tortoise/Desktop/U201614515\_4.exe.......

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 5 -I 1 2 3 4 -O 2 -O 2

用户输出:S 5 I 1 2 3 4 O 3 4 O

标准输出:S 5 I 1 2 3 4 O 3 4 O

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 5 -O 0 -I 1 2 3 4 -O 5 -I 1

用户输出:S 5 O I 1 2 3 4 O E

标准输出:S 5 O I 1 2 3 4 O E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 5 -I 1 2 3 4 -O 2 -I 5 6 7 -I 8

用户输出:S 5 I 1 2 3 4 O 3 4 I 3 4 5 6 7 I 3 4 5 6 7 8

标准输出:S 5 I 1 2 3 4 O 3 4 I 3 4 5 6 7 I 3 4 5 6 7 8

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 5 -I 1 2 3 4 -O 2 -A 4 -I 5 6 -I 7 -I 8

用户输出:S 5 I 1 2 3 4 O 3 4 A 3 4 I 3 4 5 6 I 3 4 5 6 7 I 3 4 5 6 7 8

标准输出:S 5 I 1 2 3 4 O 3 4 A 3 4 I 3 4 5 6 I 3 4 5 6 7 I 3 4 5 6 7 8

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 5 -I 1 2 3 4 -O 2 -C -I 5 6 -A 2

用户输出:S 5 I 1 2 3 4 O 3 4 C 3 4 I 3 4 5 6 A 3 4 5 6

标准输出:S 5 I 1 2 3 4 O 3 4 C 3 4 I 3 4 5 6 A 3 4 5 6

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 5 -I 1 2 3 4 -O 2 -N

用户输出:S 5 I 1 2 3 4 O 3 4 N 2

标准输出:S 5 I 1 2 3 4 O 3 4 N 2

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 5 -I 1 2 3 4 -G 3 -G 7

用户输出:S 5 I 1 2 3 4 G 4 G E

标准输出:S 5 I 1 2 3 4 G 4 G E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 5 -I 1 2 3 4 -G 3 -I 5 6 7 8 -O 3 -I 9 0 -G 6 -I 1

用户输出:S 5 I 1 2 3 4 G 4 I 1 2 3 4 5 6 7 8 O 4 5 6 7 8 I 4 5 6 7 8 9 0 G 0 I E

标准输出:S 5 I 1 2 3 4 G 4 I 1 2 3 4 5 6 7 8 O 4 5 6 7 8 I 4 5 6 7 8 9 0 G 0 I E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 3 -I 1 2 3 -O 1 -I 5 6 -G 1 -G 6

用户输出:S 3 I 1 2 3 O 2 3 I 2 3 5 6 G 3 G E

标准输出:S 3 I 1 2 3 O 2 3 I 2 3 5 6 G 3 G E

答案正确！

执行命令：C:/Users/Tortoise/Desktop/U201614515\_4.exe -S 3 -I 1 2 3 4 -G 1 -I 5 6 -G 5 -O 6 -O 1

用户输出:S 3 I 1 2 3 4 G 2 I 1 2 3 4 5 6 G 6 O O E

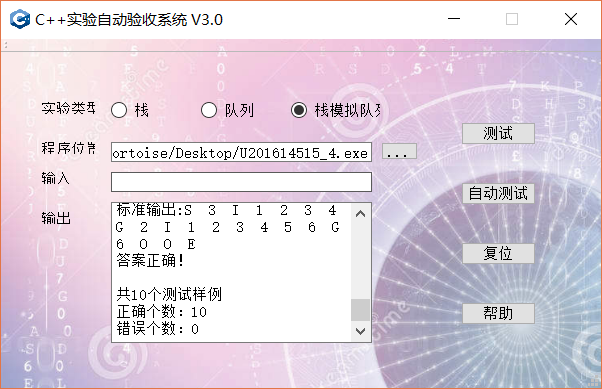
标准输出:S 3 I 1 2 3 4 G 2 I 1 2 3 4 5 6 G 6 O O E

答案正确！

共10个测试样例

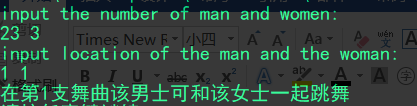
正确个数：10

错误个数：0

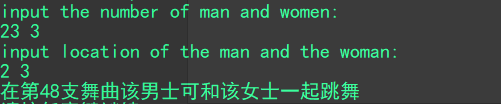


舞伴问题测试

M=23 F=3 m=1 f=1 理论值：1 运算结果：1



M=23 F=3 m=2 f=3 理论值：48 运算结果：48



# 五、特点与不足

## 1.技术特点

各个模块独立编写，在之后的程序中可以直接模块复用。

使用vs开发完成再到c++builder复测，保证本地环境对开发结果无影响。

## 2.不足

无

# 七、源码和说明

## 1.文件清单及其功能说明

* command\_line\_parser.h与command\_line\_parser.cpp为命令行解析模块
* file\_writer.h与file\_writer.cpp为文件写入模块
* stack\_operator\_overloading.h与stack\_operator\_overloading.cpp为栈及功能定义模块
* U201614515\_2.h与U201614515\_2.cpp提供程序检测运行入口(main函数)

## 2.用户使用说明书

可执行程序为U201614515\_2.exe程序运行结果保存在U201614515\_2.txt中。源代码在任何环境下可直接建立工程进行编译。

## 3.源代码

### a.queue\_ex\_6.h

1. #pragma once
2. #define QUEUE\_EX\_6
4. #ifndef STACK\_OPERATOR\_OVERLOADING\_H
6. #include"stack\_operator\_overloading.h"
8. #endif // !STACK\_OPERATOR\_OVERLOADING\_H
10. **using** **namespace** std;
12. **class** QUEUE: **public** STACK {
13. STACK s2;
14. **public**:
15. QUEUE(**int** m);       //初始化队列：每个栈最多m个元素
16. QUEUE(**const** QUEUE&s);       //用队列s拷贝初始化队列
17. operator **int**() **const**;           //返回队列的实际元素个数
18. QUEUE& operator<<(**int** e);         //将e入队列,并返回队列
19. QUEUE& operator>>(**int** &e);    //出队列到e,并返回队列
20. **int** operator[](**int** x);
21. QUEUE& operator=(**const** QUEUE&s); //赋s给队列,并返回被赋值的队列
22. **void** print() **const**;         //打印队列
23. ~QUEUE();                   //销毁队列
24. };
25. **void** run\_queue\_ex\_6(**int** argc, **char** \* argv[]);
26. **bool** isPrime(**int** n)
27. {
28. **for** (**int** i = 2; i <= sqrt((**double**)n); i++) {
29. **if** (n%i == 0) {
30. **return** **false**;//不是素数
31. }
32. }
33. **return** **true**;//是素数
34. }

### b.queue\_ex\_6.cpp

1. #include "queue\_ex\_6.h"

4. QUEUE::QUEUE(**int** m) :s2(m), STACK(m) {}
6. QUEUE::QUEUE(**const** QUEUE & s) : s2(s.s2), STACK(s.s2.size()) {}
8. QUEUE::operator **int**() **const**
9. {
10. //   cout << "[info] breakpoint in class:QUEUE function: operator int message:pos of queue: " << STACK::operator int() + (int)(this->s2) ;
11. **return** STACK::operator **int**() + (**int**)(**this**->s2);
12. }
14. QUEUE & QUEUE::operator<<(**int** e)
15. {
17. **if** (STACK::operator **int**() >=  **this**->size() && (**int**)(**this**->s2) == 0) {
18. //   cout << "[info] breakpoint in class:QUEUE function: operator << message: stack 1 status: full ,transfer to stack 2";
20. **while** (STACK::operator **int**() > 0) {
21. **int** tmp = 0;
22. STACK:: operator >> (tmp);
23. **this**->s2 << tmp;
24. }
25. }
26. **if** (STACK::operator **int**() >=  **this**->size()) {
27. //   cout << "[error] breakpoint in class:QUEUE function: operator << message: queue full ";
28. **this**->error\_flag = **true**;
29. **return** \***this**;
30. }
31. STACK:: operator << (e);
32. //   cout << "[info] breakpoint in class:QUEUE function: operator << message: push " << e;
33. **return** \***this**;
34. }
36. QUEUE & QUEUE::operator>>(**int** & e)
37. {
38. **if** ((**int**)(**this**->s2) > 0) {
39. **this**->s2 >> e;
40. }
41. **else** **if** (STACK::operator **int**() >= 1) {
42. **while** (STACK::operator **int**() > 1) {
43. **int** tmp = 0;
44. STACK::operator>> (tmp);
45. **this**->s2 << tmp;
46. }
47. STACK::operator >> (e);
48. **while** ((**int**)(**this**->s2) > 0) {
49. **int** tmp = 0;
50. **this**->s2 >> tmp;
51. STACK::operator << (tmp);
52. }
53. }
54. **else** {
55. //   cout << "[error] breakpoint in class:QUEUE function: operator >> message: empty queue ";
56. **this**->error\_flag = **true**;
57. }
58. //   cout << "[info] breakpoint in class:QUEUE function: operator >> message: pop " << e;
59. **return** \***this**;
60. }
62. **int** QUEUE::operator[](**int** x)
63. {
64. **if** (x >= (**int**)(**this**->s2)) {
65. x -= (**int**)(**this**->s2);
66. }
67. **else** {
68. //   cout << "[info] breakpoint in class:QUEUE function: operator[] message:get elem in stack 2" << endl;
69. **return** (**this**->s2)[x];
70. }
71. **if** (x + 1 >(STACK::operator **int**())) {
72. //   cout << "[error] breakpoint in class:QUEUE function: operator[] message:error index" << endl;
73. STACK::error\_flag = **true**;
74. **return** 0;
75. }
76. **return** STACK::operator [](x);
77. }
79. QUEUE & QUEUE::operator=(**const** QUEUE & s)
80. {
81. **this**->s2 = s.s2;
82. STACK::operator=(s);
83. **return** \***this**;
84. }
86. **void** QUEUE::print() **const**
87. {
88. //cout << "[info] breakpoint in class:QUEUE function: print message:开始打印队列元素" << endl;
89. **for** (**int** i = (**int**)(**this**->s2) - 1; i >= 0; --i) {
90. **this**->file\_writer->writer((**this**->s2)[i]);
92. }
93. **for** (**int** i = 0; i < STACK::operator **int**(); ++i) {
94. **this**->file\_writer->writer(STACK::operator[](i));
95. }
96. }
98. QUEUE::~QUEUE()
99. {
100. **this**->file\_writer->~FILE\_WRITER();
101. **this**->STACK::~STACK();
102. }
104. **void** run\_queue\_ex\_6(**int** argc, **char** \* argv[])
105. {
106. **char** file\_name[] = "U201614515\_4.txt";
107. **if**(argc >1){
109. COMMAND\_LINE\_PARSER \* command\_line\_parser = **new** COMMAND\_LINE\_PARSER(argc, argv);
110. cmd\_list \* head = command\_line\_parser->get\_list()->next;
111. QUEUE \* queue = **new** QUEUE(10);
112. **while** (head != NULL) {
113. **switch** (head->cmd) {
114. **case** 'S':
115. queue = **new** QUEUE(head->elem\_list[0]);
116. queue->file\_writer = **new** FILE\_WRITER(file\_name);
117. queue->file\_writer->writer(head->cmd);
118. printf("%c  ", head->cmd);
119. queue->file\_writer->writer(head->elem\_list[0]);
120. printf("%d  ", head->elem\_list[0]);
121. **break**;
122. **case** 'I':
123. **for** (**int** i = 0; i < head->size\_elem\_list; i++) {
124. (\*queue) << head->elem\_list[i];
125. }
126. queue->file\_writer->writer(head->cmd);
127. printf("%c  ", head->cmd);
128. **if** (queue->error\_flag == **true**) {
129. queue->file\_writer->writer('E');
130. printf("%c  ", 'E');
131. **return**;
132. }
133. queue->print();
134. **break**;
135. **case** 'O':
136. **int** e;
137. printf("option O point hit\n");
138. queue->file\_writer->writer(head->cmd);
139. printf("%c  ", head->cmd);
140. printf("option O point hit par: size = %d\n", head->size\_elem\_list);
141. **if** (head->size\_elem\_list == 0 || head->elem\_list[0] > **int**(\*queue)) {
142. queue->error\_flag = **true**;
143. }
144. **else** {
146. **for** (**int** i = 0; i < head->elem\_list[0]; i++) {
147. (\*queue) >> e;
148. }
149. }
150. **if** (queue->error\_flag == **true**) {
151. queue->file\_writer->writer('E');
152. printf("%c ", 'E');
153. **return**;
154. }
156. queue->print();
157. **break**;
158. **case** 'C':
159. queue->file\_writer->writer(head->cmd);
160. printf("%c  ", head->cmd);
161. queue->print();
162. **break**;
163. **case** 'A':
164. queue->file\_writer->writer(head->cmd);
165. printf("%c  ", head->cmd);
166. queue->print();
167. **break**;
168. **case** 'N':
169. queue->file\_writer->writer(head->cmd);
170. printf("%c  ", head->cmd);
171. queue->file\_writer->writer(**int**(\*queue));
172. //\_num(howMany(&stack\_main));
173. **break**;
174. **case** 'G':
175. queue->file\_writer->writer(head->cmd);
176. printf("%c  ", head->cmd);
177. **if** (head->elem\_list[0] >= **int**(\*queue)) {
178. queue->file\_writer->writer('E');
179. printf("%c  ", 'E');
180. **return**;
181. }
182. **else** {
183. queue->file\_writer->writer((\*queue)[head->elem\_list[0]]);
184. }
185. **break**;
186. **default**:
187. **break**;
188. }
189. head = head->next;
190. }
191. }**else**{
192. **while** (**true**)
193. {
194. cout << "input the number of man and women:" << endl;
195. **int** M, F;
196. cin >> M >> F;
197. **if** (M == F)
198. {
199. cout << "input error! can not input equal number!" << endl;
200. **continue**;
201. }
202. **if** (!isPrime(M) || !isPrime(F))
203. {
204. cout << "input error! must input prime num!" << endl;
205. **continue**;
206. }
207. cout << "input location of the man and the woman:" << endl;
208. **int** manLocation, womanLocation;
209. cin >> manLocation >> womanLocation;
210. QUEUE man(M), woman(F);//男士队伍，女士队伍
211. man.file\_writer = **new** FILE\_WRITER(file\_name);
212. woman.file\_writer =**new** FILE\_WRITER(file\_name);
213. **int** i;
214. /\*初始化队伍\*/
215. **for** (i = 1; i <= M; i++)
216. man << i;
217. **for** (**int** i = 1; i <= F; i++)
218. woman << i;
219. **int** res = 1;
220. /\*模拟跳舞\*/
221. **while** (1)
222. {
223. **int** m, w;
224. man >> m; woman >> w;
225. **if** (m == manLocation && w == womanLocation) **break**;
226. man << m; woman << w;
227. res++;
228. //if (res > 10000) { cout << "no answer" << endl; break; }
229. }
230. cout << "在第" << res << "支舞曲该男士可和该女士一起跳舞" << endl;
231. system("pause");
232. }
234. }
235. system("pause");
236. }

### c.U201614515\_4.h

1. #define \_CRT\_SECURE\_NO\_WARNINGS
2. #pragma once
3. //#define RUN\_EX\_1
4. //#define RUN\_EX\_2
5. //#define RUN\_EX\_3
6. #define RUN\_EX\_6
8. #ifndef STACK\_EX\_1\_H
9. #ifdef RUN\_EX\_1
10. #include"stack\_ex\_1.h"
11. #endif // RUN\_EX\_1
12. #endif // !STACK\_EX\_1\_H

15. #ifndef STACK\_OPERATOR\_OVERLOADING\_H
16. #ifdef RUN\_EX\_2
17. #include"stack\_operator\_overloading.h"
18. #endif // RUN\_EX\_2
19. #endif // !STACK\_OPERATOR\_OVERLOADING\_H

22. #ifndef QUEUE\_EX\_3\_H
23. #ifdef RUN\_EX\_3
24. #include"queue\_ex\_3.h"
25. #endif // RUN\_EX\_3
26. #endif // !QUEUE\_EX\_3\_H
28. #ifndef QUEUE\_EX\_6
29. #ifdef RUN\_EX\_6
30. #include"queue\_ex\_6.h"
31. #endif // RUN\_EX\_6
32. #endif // !QUEUE\_EX\_6
34. **using** **namespace** std;

### d.U201614515\_4.cpp

1. #include"U201614515\_4.h"
2. **int** main(**int** argc, **char**\* argv[]) {
3. //运行栈检测
4. //run\_stack(argc, argv);
5. //run\_stack\_2(argc, argv);
6. //run\_queue\_ex\_3(argc, argv);
7. run\_queue\_ex\_6(argc, argv);
8. //system("pause");
9. **return** 0;
10. }