山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000130198 | 姓名： 隋春雨 | | 班级： 20.4 |
| 实验题目：队列 | | | |
| 实验学时：2 | | 实验日期： 2021-11-11 | |
| 实验目的：  1、掌握队列结构的定义与实现；  2、掌握队列结构的使用。 | | | |
| 软件开发环境：  CLION2020 | | | |
| 1. 实验内容   题目描述：  首先创建队列类，采用数组描述；实现卡片游戏，假设桌上有一叠扑克牌，依次编号为1-n（从最上面开始）。当至少还有两张的时候，可以进行操作：把第一张牌扔掉，然后把新的第一张放到整叠牌的最后。输入n，输出最后剩下的牌。  输入输出格式：  输入：  一个整数n，代表一开始卡片的总数。  输出：  最后一张卡片的值。   1. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法） 2. 数据结构：可以看出来这道题的本质就是循环双端队列，队列，一种特殊的线性表特点：只允许在一端输入，在另一端输出。输入端称为队尾，输出端称为队头。因此，队列，又称为先进先出表（FIFO），类似于生活中的排队，先来的排在前头，后来的排在后头，一个一个办理业务。同时由于如果不使用循环队列的话，一个一个删除之后的队列维护比较麻烦，空间利用率低，因此我们使用循环队列。   循环数据     1. 算法：     从题目要求中我们可以看到，当它的size>=2的时候，就pop队列头，再将队列头的值Push到队列尾，最后再pop，最后输出即可   1. 测试结果（测试输入，测试输出）   输入：    输出：    提交OJ最后的结果：     1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径） 2. 对于情况的考虑我们应该全面，比如说这个，循环队列很有可能theBack在队头，front在队尾，那么我们加上一个arrayLength再取模就行了，在结合点和线的映射关系，就可以计算出来      1. 我们在使用函数的时候，应该充分的考虑异常情况，比如说调用top函数的时候或者Pop的时候，都应该考虑一下是否为空，如果是空，那么就抛出异常      1. 要注意私有成员的更新，public函数知道自己所处的状态都是靠着私有成员才知道的，如果没有及时更新，那数据就成了垃圾数据，没有任何意义。我在写实验的时候，也经历过没更新导致的Bug，最终debug查出来，就是下面这个：      1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释） 2. #include<iostream> 3. #include <sstream> 4. **using** **namespace** std; 6. **template**<**class** T> 7. **class** arrayQueue 8. { 9. **public**: 10. arrayQueue(**int** initialCapacity = 10);//初始化 11. ~arrayQueue() { **delete**[] queue; }//析构 12. **bool** empty() **const** { **return** theFront == theBack; } 13. **int** size() **const**//计算size 14. { 15. **return** (theBack - theFront + arrayLength) % arrayLength;//加上arrayLength是为了 16. // 防止theBack在头，theFront在队列尾的情况 17. } 18. T& front() 19. {// return front element 20. **if** (theFront == theBack)//空 21. **throw** "queueEmpty()"; 22. **return** queue[(theFront + 1) % arrayLength]; 23. } 24. T& back() 25. {// return theBack element 26. **if** (theFront == theBack)//空 27. **throw** "queueEmpty()"; 28. **return** queue[theBack]; 29. } 30. **void** pop() 31. {// remove theFront element 32. **if** (theFront == theBack) 33. **throw** "queueEmpty()"; 34. theFront = (theFront + 1) % arrayLength; 35. } 36. **void** push(**const** T& theElement); 37. **private**: 38. **int** theFront;       // 1 counterclockwise from theFront element 39. **int** theBack;        // position of theBack element 40. **int** arrayLength;    // queue capacity 41. T\* queue;           // element array 42. }; 44. **template**<**class** T> 45. arrayQueue<T>::arrayQueue(**int** initialCapacity) 46. {// Constructor. 48. arrayLength = initialCapacity; 49. queue = **new** T[arrayLength]; 50. theFront = 0; 51. theBack = 0; 52. } 54. **template**<**class** T> 55. **void** arrayQueue<T>::push(**const** T& theElement) 56. {// Add theElement to queue. 58. // increase array length if necessary 59. **if** ((theBack + 1) % arrayLength == theFront)//如果已经满了 60. {// double array length 61. // allocate a new array 62. T\* newQueue = **new** T[2 \* arrayLength]; 64. **for** (**int** i = 0, cnt = 0, pos = (theFront + 1) % arrayLength; cnt < arrayLength; i++) 65. { 66. newQueue[i] = queue[pos]; 67. pos = (pos + 1) % arrayLength; 68. cnt++; 69. } 71. // switch to newQueue and set theFront and theBack 72. theFront = 2 \* arrayLength - 1;//更新私有成员 73. theBack = arrayLength - 2;   // queue size arrayLength - 1 74. arrayLength \*= 2; 75. queue = newQueue;//指针赋值 76. } 78. // put theElement at the theBack of the queue 79. theBack = (theBack + 1) % arrayLength; 80. queue[theBack] = theElement; 81. } 83. **int** main() 84. { 85. **int** n; 86. cin >> n; 87. arrayQueue<**int**>q; 88. **for** (**int** i = 0; i < n; i++) 89. { 90. q.push\_back(i + 1);//一个一个压入 91. } 92. **int** cnt = n; 93. **while** (cnt >= 2) 94. { 95. q.pop\_front(); 96. **int** val = q.front(); 97. q.pop\_front(); 98. q.push\_back(val); 99. cnt--;//更新变量 100. } 101. cout << q.front();//输出  104. } | | | |