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。

2. 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）

(1) 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。

```
class arrayQueue
{
public:
    arrayQueue(int initialCapacity = 10);
    ~arrayQueue() { delete[] queue; }
    bool empty() const { return theFront == theBack; }
    int size() const
    {
        return (theBack - theFront + arrayLength) % arrayLength;
    }
    T& front()
    {// return front element
        if (theFront == theBack)
            throw "queueEmpty()";
        return queue[(theFront + 1) % arrayLength];
    }
    T& back()
    {// return theBack element
        if (theFront == theBack)
            throw "queueEmpty()";
        return queue[theBack];
    }
    void pop()
    {// remove theFront element
```

(2) 算法：

假设桌上有一叠扑克牌，依次编号为1-n（从上至下）。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（原先扔掉的牌下方的那张牌,即第二张牌）放到整叠牌的最后。输入n，输出最后剩下的牌。

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

3. 测试结果（测试输入，测试输出）

输入：

输入

```
100
```

输出：

```
C:\Users\4399\untitled99\cmake-build-debug\untitled99.exe
100
72
进程已结束，退出代码为 0
```

提交 OJ 最后的结果：

✓Accepted

#	Result	Score	Time	Memory
1	✓Accepted	10	1 ms	3748 KiB
2	✓Accepted	10	1 ms	4144 KiB
3	✓Accepted	10	2 ms	4428 KiB
4	✓Accepted	10	2 ms	4656 KiB
5	✓Accepted	10	3 ms	5472 KiB
6	✓Accepted	10	3 ms	5684 KiB
7	✓Accepted	10	4 ms	6092 KiB
8	✓Accepted	10	4 ms	6556 KiB
9	✓Accepted	10	2 ms	6788 KiB
10	✓Accepted	10	5 ms	7328 KiB

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

(1) 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

```
int size() const//计算size
{
    return (theBack - theFront + arrayLength) % arrayLength;//加上arrayLength是为了
    // 防止theBack在头，theFront在队列尾的情况
}
```

(2) 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

```
T& back()
{// return theBack element
    if (theFront == theBack)//空
        throw "queueEmpty()";
    return queue[theBack];
}
void pop()
{// remove theFront element
    if (theFront == theBack)//空
        throw "queueEmpty()";
    theFront = (theFront + 1) % arrayLength;
}
```

(3) 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：

```
    // switch to newQueue and set theFront and theBack
    theFront = 2 * arrayLength - 1;//更新私有成员
    theBack = arrayLength - 2;   // queue size arrayLength - 1
    arrayLength *= 2;
    queue = newQueue;//指针赋值
 }

 // put theElement at the theBack of the queue
 theBack = (theBack + 1) % arrayLength;
 queue[theBack] = theElement;
```

5. 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）

```
1.  #include<iostream>
2.  #include <sstream>
3.  using namespace std;
4.
5.  template<class T>
```

```
6.  class arrayQueue
7.  {
8.  public:
9.      arrayQueue(int initialCapacity = 10);//初始化
10.     ~arrayQueue() { delete[] queue; }//析构
11.     bool empty() const { return theFront == theBack; }
12.     int size() const//计算 size
13.     {
14.         return (theBack - theFront + arrayLength) % arrayLength;//加上 arrayLength 是为了
15.         // 防止 theBack 在头，theFront 在队列尾的情况
16.     }
17.     T& front()
18.     {// return front element
19.         if (theFront == theBack)//空
20.             throw "queueEmpty()";
21.         return queue[(theFront + 1) % arrayLength];
22.     }
23.     T& back()
24.     {// return theBack element
25.         if (theFront == theBack)//空
26.             throw "queueEmpty()";
27.         return queue[theBack];
28.     }
29.     void pop()
30.     {// remove theFront element
31.         if (theFront == theBack)
32.             throw "queueEmpty()";
33.         theFront = (theFront + 1) % arrayLength;
34.     }
35.     void push(const T& theElement);
36. private:
37.     int theFront;       // 1 counterclockwise from theFront element
38.     int theBack;        // position of theBack element
39.     int arrayLength;    // queue capacity
40.     T* queue;           // element array
41. };
42.
43. template<class T>
44. arrayQueue<T>::arrayQueue(int initialCapacity)
45. {// Constructor.
46.
47.     arrayLength = initialCapacity;
48.     queue = new T[arrayLength];
49.     theFront = 0;
50.     theBack = 0;
51. }
```

```
52.
53. template<class T>
54. void arrayQueue<T>::push(const T& theElement)
55. {// Add theElement to queue.
56.
57.    // increase array length if necessary
58.    if ((theBack + 1) % arrayLength == theFront)//如果已经满了
59.    {// double array length
60.       // allocate a new array
61.       T* newQueue = new T[2 * arrayLength];
62.
63.       for (int i = 0, cnt = 0, pos = (theFront + 1) % arrayLength; cnt < arrayLength; i++)
64.       {
65.          newQueue[i] = queue[pos];
66.          pos = (pos + 1) % arrayLength;
67.          cnt++;
68.       }
69.
70.       // switch to newQueue and set theFront and theBack
71.       theFront = 2 * arrayLength - 1;//更新私有成员
72.       theBack = arrayLength - 2;   // queue size arrayLength - 1
73.       arrayLength *= 2;
74.       queue = newQueue;//指针赋值
75.    }
76.
77.    // put theElement at the theBack of the queue
78.    theBack = (theBack + 1) % arrayLength;
79.    queue[theBack] = theElement;
80. }
81.
82. int main()
83. {
84.    int n;
85.    cin >> n;
86.    arrayQueue<int>q;
87.    for (int i = 0; i < n; i++)
88.    {
89.       q.push_back(i + 1);//一个一个压入
90.    }
91.    int cnt = n;
92.    while (cnt >= 2)
93.    {
94.       q.pop_front();
95.       int val = q.front();
96.       q.pop_front();
97.       q.push_back(val);
```

```
98.             cnt--;//更新变量
99.         }
100.        cout << q.front();//输出
101.
102.
103. }
```