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：202000130143	姓名： 郑凯饶	班级： 计科 20.1
实验题目：队列		
实验学时：2	实验日期： 1110	

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：

Windows 10 家庭中文版 64 位（10.0，版本 18363）

Dev-C++ IDE

1. 实验内容

创建队列类，采用数组描述，完成卡片游戏。

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）

```
template<class T>
class arrayQueue {
    public:
        arrayQueue(int iniCap = 10);
        ~arrayQueue() { delete [] queue; }
        bool empty() const { return Front == Back; }
        int size() const { return (Back - Front + Length) % Length; }
        T& front() {
            if (Front == Back) { cout << "The queue is empty.\n"; return; }
            return queue[(Front + 1) % Length];
        }
        T& back() {
            if (Front == Back) { cout << "The queue is empty.\n"; return; }
            return queue[Back];
        }
        void pop() {
            Front = (Front + 1) % Length;
            queue[Front].~T();
        }
        void push(const T& Ele);
    private:
        int Front, Back, Length;
        T* queue;
```

Front:列首元素的前一位

Back:列尾元素

通过 Front == Back 判断队列是否为空，队列的容量为 Length – 1

卡牌游戏直接根据描述模拟即可。

3. 测试结果（测试输入，测试输出）

在 oj 平台上成功提交。

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

这次实验比较简单，但是我还是犯了一些严重的错误。在题目没有明确数据规模的情况下，我自己很自信地把空间扩展的部分代码省略了，结果测试点甚至没有样例，数据规模达到了 1e6 – 1e7，尝试了几次 wrong answer 之后才通过。

因此，在写程序的时候还是要注重空间的拓展性。

题目本质是一个约瑟夫环问题，之前我们可能是通过循环链表模拟实现，后来发现其中有直接的数学的递推规律。现在我们又通过队列进行模拟。

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）

```
#include<bits/stdc++.h>
using namespace std;

template<class T>
class arrayQueue {
 public:
  arrayQueue(int iniCap = 10);
  ~arrayQueue() { delete [] queue; }
  bool empty() const { return Front == Back; }
  int size() const { return (Back - Front + Length) % Length; }
  T& front() {
   if (Front == Back) { cout << "The queue is empty.\n"; return; }
   return queue[(Front + 1) % Length];
  }
  T& back() {
   if (Front == Back) { cout << "The queue is empty.\n"; return; }
   return queue[Back];
  }
  void pop() {
   Front = (Front + 1) % Length;
   queue[Front].~T();
  }
  void push(const T& Ele);
 private:
  int Front, Back, Length;
  T* queue;
};

template<class T>
arrayQueue<T>::arrayQueue(int iniCap) {
 Length = iniCap;
 queue = new T[Length];
```

```
33. Front = 0;
34. Back = 0;
35.}
36.
37.template<class T>
38.void arrayQueue<T>::push(const T& Ele) {
39. // CL
40. if ((Back + 1) % Length == Front) {
41.  T* newQ = new T[2 * Length];
42.  int st = (Front + 1) % Length;
43.  if (st < 2) { // _ 1 2 3 4 or 1 2 3 4 _
44.   copy(queue + st, queue + st + Length - 1, newQ);
45.  }
46.  else { // 3 4 5 _ 1 2 -> 1 2 3 4 5 _ _ _ _ ...
47.   copy(queue + st, queue + Length, newQ);
48.   copy(queue, queue + Back + 1, queue + Length - st);
49.  }
50.
51.  Front = 2 * Length - 1;
52.  Back = Length - 2;
53.  Length *= 2;
54.  queue = newQ;
55. }
56. Back = (Back + 1) % Length;
57. queue[Back] = Ele;
58.}
59.
60.arrayQueue<int> Q;
61.
62.// 1 2 3 4 5 6 7 8 9 10
63.// 3 4 5 6 7 8 9 10 2
64.// 5 6 7 8 9 10 2 4
65.// 2 4 6 8 10
66.// 6 8 10 4
67.// 10 4 8
68.// 8 4
69.// 4
70.
71.int main(){
72. int n; cin >> n;
73. for (int i = 1; i <= n; i++) {
74.  Q.push(i);
75. }
76. while (Q.size() >= 2) {
77.  Q.pop();
78.  auto f = Q.front();
```

```
  Q.pop();
  Q.push(f);
 }
 int Ans = Q.front(); Q.pop();
 cout << Ans << '\n';

 return 0;
}
```