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 ： 202200400053	姓名： 王宇涵	班级： 22 级 2 班
实验题目：队列		
实验学时：2	实验日期： 2023-10-25	

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：

Vscode

1. 实验内容

题目描述： 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，编号依次为 1～n（从最上面开始）。当至少还有两张牌时，可以进行如下操作：把第一张牌扔掉，然后把新的第一张牌放到整叠牌的最后。对于输入的 n，输出最后剩下的牌的编号。 输入输出格式： 输入：一个整数 n，代表开始时扑克牌的总数。 输出：最后剩下的牌的编号。

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）

ArrayQueue<int> cards(n+1);：创建一个队列用于表示纸牌。

循环从 1 到 n，表示将 1 到 n 编号的纸牌依次放入队列

进入循环 while(cards.size()>=2)，直到队列中剩下一张纸牌为止

循环由以下四部分构成

1. cards.pop();：弹出队列中的第一张纸牌

2.int t = cards.front();：获取队列中的当前第一张纸牌

3.cards.pop();：再次弹出队列中的第一张纸牌

4.cards.push(t);：将第一张纸牌下面的纸牌移到了队列的末尾。

最后，输出队列中剩下的唯一一张纸牌的编号，即 cards.front()显示结果。

3. 测试结果（测试输入，测试输出）

输入 100

输出 72

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

问题:

一开始没有注意到 size()>=2 的时候,队列出去两张牌,以为只出去一张牌,进行修改后通过了 OJ

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<iostream>
#ifndef myExceptions_
#define myExceptions_
#include <string>
#include<iostream>

using namespace std;

// illegal parameter value
class illegalParameterValue
{
    public:
        illegalParameterValue(string theMessage = "Illegal parameter value")
                {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
};

// illegal input data
class illegalInputData
{
    public:
```

```
        illegalInputData(string theMessage = "Illegal data input")
                {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
};

// illegal index
class illegalIndex
{
    public:
        illegalIndex(string theMessage = "Illegal index")
                {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
};

// matrix index out of bounds
class matrixIndexOutOfBounds
{
    public:
        matrixIndexOutOfBounds
                (string theMessage = "Matrix index out of bounds")
                {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
};

// matrix size mismatch
class matrixSizeMismatch
```

```
{
    public:
        matrixSizeMismatch(string theMessage =
                           "The size of the two matrics doesn't match")
                {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
};

// stack is empty
class stackEmpty
{
    public:
        stackEmpty(string theMessage =
                           "Invalid operation on empty stack")
                {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
};

// queue is empty
class queueEmpty
{
    public:
        queueEmpty(string theMessage =
                           "Invalid operation on empty queue")
                {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
```

```
};


// hash table is full
class hashTableFull
{
    public:
        hashTableFull(string theMessage =
                              "The hash table is full")
                    {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
};


// edge weight undefined
class undefinedEdgeWeight
{
    public:
        undefinedEdgeWeight(string theMessage =
                              "No edge weights defined")
                    {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
};


// method undefined
class undefinedMethod
{
    public:
        undefinedMethod(string theMessage =
                              "This method is undefined")
```

```
                {message = theMessage;}
        void outputMessage() {cout << message << endl;}
    private:
        string message;
};
#endif

using namespace std;

template<class T>
class Queue
{
public:
    virtual ~Queue(){}
    virtual bool empty()const =0;
    virtual int size()const =0;
    virtual T front()=0;
    virtual T back()=0;
    virtual void pop()=0;
    virtual void push(const T& theElement)=0;
};

template<class T>
class ArrayQueue :public Queue<T>
{
public:
    ArrayQueue(int theCapacity=10);
    ~ArrayQueue(){delete[] queue;}
    bool empty() const{return queueBack==queueFront;}
    bool full() const {return (queueBack+1)%arrayLength==queueFront;}
    int size()const {return (queueBack-queueFront+arrayLength)%arrayLength;}
    T front()
```

```
        {
            if(empty())
                throw queueEmpty();
            return queue[(queueFront+1)%arrayLength];
        }
        T back()
        {
            if(empty())
                throw queueEmpty();
            return queue[queueBack];
        }
        void pop()
        {
            if(empty())
                throw queueEmpty();
            queueFront=(queueFront+1)%arrayLength;
            queue[queueFront].~T();
        }
        void push(const T& theElement);
        void changeQueueLength();
        void inputQueue();

private:
        int queueFront;
        int queueBack;
        int arrayLength;
        T *queue;
};

template <class T>
inline ArrayQueue<T>::ArrayQueue(int theCapacity)
{
```

```
    if(theCapacity<1)
        throw illegalParameterValue("Capacity must >0");
    arrayLength=theCapacity;
    queue=new T[theCapacity];
    queueFront=queueBack=0;
}

template <class T>
inline void ArrayQueue<T>::push(const T &theElement)
{
    //扩容
    if(size()==arrayLength-1)
    {
        changeQueueLength();
    }
    queueBack=(queueBack+1)%arrayLength;
    queue[queueBack]=theElement;
}

template <class T>
inline void ArrayQueue<T>::changeQueueLength()
{
    T* newQueue=new T[2*arrayLength];

    int start=(queueFront+1)%arrayLength;
    //未形成环形
    if(start<2)
        copy(queue+start,queue+start+arrayLength-1,newQueue);
    //形成环形
    else
    {
        copy(queue+start,queue+arrayLength,newQueue);
```

```
            copy(queue,queue+queueBack+1,newQueue+arrayLength-start);
        }


        queueFront=2*arrayLength-1;
        queueBack=arrayLength-2;
        arrayLength=arrayLength*2;
        delete[]queue;
        queue=newQueue;
}


int n;
int main()
{
        cin>>n;
        ArrayQueue<int>cards(n+1);
        for(int i=1;i<=n;i++)
        {
                cards.push(i);
        }
        while(cards.size()>=2)
        {
                cards.pop();
                int t=cards.front();
                cards.pop();
                cards.push(t);
        }
        cout<<cards.front()<<endl;
        return 0;
  }
```