《数据结构》课程实践报告

院、系	计算机学院	年级专业	21 计算机科学与技术	姓名	赵鹏	学号	2127405037
实验布置日期	2022.9.5		提交日期	2022.9.21		成绩	

课程实践实验 4 小猫钓鱼纸牌游戏

一、问题描述及要求

A 和 B 两个同学玩简单的纸牌游戏，每人手里有 n 张牌，两人轮流出牌并依次排列在桌面 上，每次出掉手里的第 1 张牌，出牌后如果发现桌面上有跟刚才打出的牌的数字相同的牌，则把从相同的那张牌开始的全部牌按次序放在自己手里的牌的末尾。当一个人手中的 牌先出完时，游戏结束，对方获胜。

如 n 为 5，A 手里的牌依次为 2 3 5 6 1，B 手里的牌依次为 1 5 4 2 9；

A 出 2；

B 出 1；

A 出 3；

B 出 5；

A 出 5，发现前面有一张 5，则把两个 5 都拿掉，这时他手里有 6 1 5 5；

桌子上的牌依次为 2 1 3；

B 出 4；

A 出 6；

B 出 2，发现前面有一张 2，则把从 2 开始的牌全部拿掉，这时他手里有 9 2 1 3 4 6 2；

桌子上没有牌了；

A 出 1；

B 出 9；

A 出 5；

B 出 2；

依次类推，直到某人先出完牌为止，则对方是胜者。

编写程序，利用栈和队列，判断谁是胜者。

二、概要设计

（1）.问题分析

这个实验实际上是一个实际的问题。本质是利用算法对游戏的整个流程进行模拟。考虑纸牌，每次打出最靠前的一张牌，把赢到的牌依次放在末尾，手中的纸牌显然满足队列 FIFO 的特性，再考虑桌上的牌，最后打出的牌堆在最上面，被最先取走，这显然

符合栈 FILO 的特性，所有很自然的想到用队列模拟手中的牌和出牌动作，用栈模拟牌放在桌上以及取牌动作。

（2）.程序结构设计思路

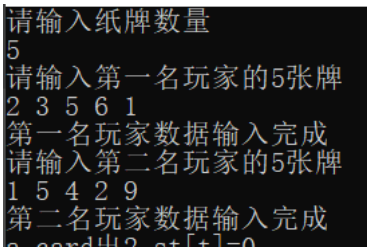
由于此游戏的流程较为简单，故结构设计思路也较简单。程序采用设计一个纸牌游戏模块来模拟一次游戏。在这一游戏模块中，设计了两个较为关键的函数，分别是 init 函数，用于读入数据，对整个游戏进行初始化和 work 函数，用于模拟整个游戏对局。后者作为接口给外部调用以进行游戏。

程序在设计过程中复用了链栈和链队列两种数据结构，避免了使用顺序队列和顺寻栈造成的空间浪费或空间不足导致的栈溢出。由于本次实验主要考察两种数据结构的应用，因此在此不再赘述两种数据结构的实现细节。这两种数据结构的具体实现可以查看附件中的 LinkStack.hpp 和 LinkQueue.hpp。

（3）.程序运行的界面设计

本实验较为简单，故程序运行中与用户的交互也相对较少。

程序首先请求用户输入每个人手中牌的数量，然后依次读入每个人的牌。随后便自动开始进行游戏。



（4）.总体设计思路

本程序在设计过程中主要采用了顺序表、链队列、链栈三种数据结构，并设计了以下类及对应的方法。

1. LinkQueue 模板类

函数名	作用
Queue	构造函数，初始化队列
~Queue	析构函数，销毁队列
front	返回队头
pop	弹出队头

push	将元素入队
empty	判断队列是否为空

2.LinkStack 模板类

函数名	作用
Queue	构造函数，初始化栈
~Queue	析构函数，销毁栈
top	返回栈顶
pop	弹出栈顶
push	将元素入栈
empty	判断栈是否为空

3.Node.h 模板类

成员名	作用
data	存储数据
next	指向下一个对象

3.Game 类

函数名	作用
Game	初始化每个人当前手中牌数
init	读入每个人的牌
Work	模拟游戏流程，输出结果

在设计的这些类中，Game 类用于表示一场游戏，Node 用于实现链式结构 LinkQueue 和 LinkStack 为 Game 中的 work 函数模拟游戏流程提供帮助。

三、详细设计

程序的主函数设计通过一定程度的封装达到了较为简洁的效果,主要是由读入数据和进行游戏这两个部分组成。

本次实验的关键点和重难点便是模拟整个游戏的流程。

在 work 函数实现模拟游戏流程中，首先将两个玩家（下面简称为 A.B）牌放入两人对应的队列中，并定义一个名为 Table 的栈用于存放已经打出的牌，具体理由在问题分析时已经解释，在此不再重述。由于两人依次出牌，很自然的想到可以记录下出牌次数，偶数次由 A 出牌，奇数次由 B 出牌，每次出牌后将次数+1 即可。

随后便是出牌和赢牌两个流程的设计。

首先从当前一轮的出牌人的队列中取出队头 card 并将 card 入栈。对于判断栈里是否已经出现过这张牌，可以采用一个 state 数组进行标记，如 state[1]=1 表示 1 这张牌已经出现过，state[2]=0 表示 2 这张牌还未出现过。

若 state[card]=0 则将 state[card]修改为 1 即可。若 state[card]==1 则说明当前需要赢牌。由于赢的牌需要顺序接到队列末尾，我们可以定义一个临时栈 Temp 对赢得的牌进行翻转。首先弹出刚入栈的 card 并 push 到 Temp 栈中，再依次弹出栈 Table 的队头元素并将其存储到临时栈 Temp 中直至栈顶等于 card，再将栈顶弹出同样存储到 Temp 中即可。至此已将所有赢的牌存储到 Temp 中，随后将 Temp 中的所有元素依次弹出并 push 到此轮玩家的队列中。

不断重复这个流程，直至 A 或 B 存在队列为空即可。

结束游戏后进行判断，队列不为空的一方即为赢家。

四、实验结果

本实验在题目中已给出了一个样例，我们用其进行测试

测试输入：

5

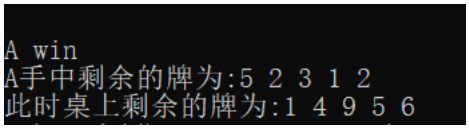
2 3 5 6 1

1 5 4 2 9

测试目的：游戏的正常流程

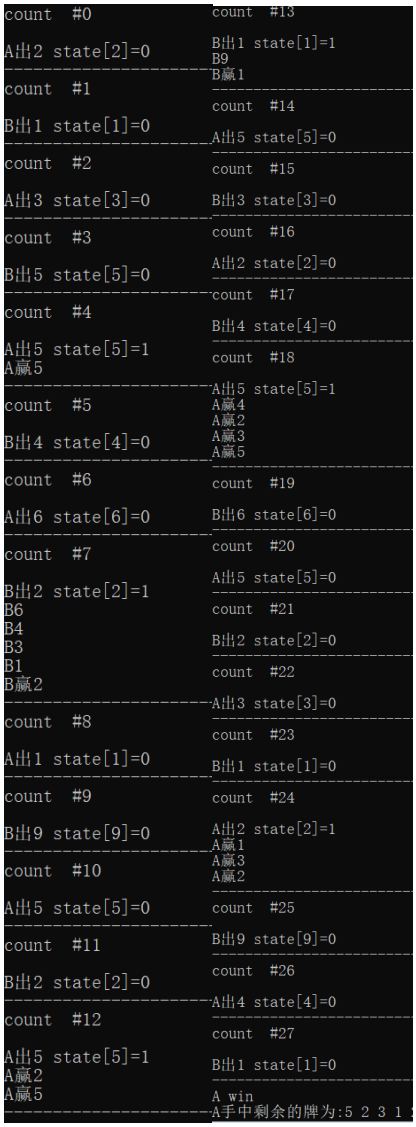
正确输出： A win!

实际输出：



测试结论：通过

具体输出为



五、实验分析与探讨

通过样例的具体输出，可以得知本程序很好的完成了整个游戏流程的模拟。

由于本次实验较为简单，在实验过程中遇到的问题也相对较少。

问题 1.存储在 Table 栈中的元素需要顺序接到队列末尾，但弹出顺序为逆序。
解决办法：将元素存储在一个临时栈中存储，在依次弹出即可完成翻转。

问题 2.在链栈结构中判断一个元素是否出现过需要遍历整个链
解决办法：额外定义一个数组，用 0/1 表示这个元素是否出现过。在入栈出栈时更新数组信息即可。

六、小结

通过本次实验，我加强了对栈和队列两种数据结构的应用。在这次的程序中，我实现了最后的结果判断以及每个人每次出牌赢牌过程的输出。但尚未实现输出每次结束后每个人手上和桌面上的牌情况。

附录：源代码

1、实验环境 Visual Studio 2022…
2、

（1）Game.h

```
#pragma once
class Game
{	private:
	static const int MAX_SIZE = 1000 + 10;
	int player_a[MAX_SIZE], player_b[MAX_SIZE];
	int state[MAX_SIZE];
	int card_num;
public:
	Game(int num);
	void init();
	void work();
};
```

(2)
Node.h

```
#pragma once
template<typename data_type>
struct Node
{
	data_type data;
	Node<data_type>* next;
};
```

(3)utility.h

```
#pragma once
#include<iostream>
#include<cstring>
using namespace std;
```

(4)LinkQueue.hpp

```
#include "Node.h"

template<typename data_type>
class Queue
{
	public:
	Queue();
	~Queue();
	bool empty();
	data_type front();
	void push(data_type item);
	void pop();

	private:
	Node<data_type> *Front,*Rear;

};
template<typename data_type>
Queue<data_type>::Queue()
{
	Node<data_type>* s=NULL;
	s=new Node<data_type>;
	s->next=NULL;
	Front=Rear=s;
}
template<typename data_type>
Queue<data_type>::~Queue()
{
	while(!empty())
		pop();
}
template<typename data_type>
bool Queue<data_type>::empty()
{
	return Front==Rear;
}
template<typename data_type>
void Queue<data_type>::push(data_type    item)
{
	Node<data_type> * s=NULL;
	s=new Node<data_type>;
	s->data=item;
	s->next=NULL;
```

```
        Rear->next=s;
        Rear=s;
}
template<typename data_type>
void Queue<data_type>::pop()
{
Node<data_type> *p = NULL;
p = Front->next;
Front->next = p->next;
if (p->next == NULL) Rear = Front;
delete p;
}
template<typename data_type>
data_type Queue<data_type>::front()
{
        data_type x=Front->next->data;

        return x;
}
```

(5)LinkStack.hpp

```
#include "Node.h"
template<typename data_type>
class Stack
{
        public:
        Stack();
        ~Stack();
        bool empty()const;
        void push(const data_type &item);
        void pop();
        data_type top();

        private:
        Node<data_type> * top_node;
};
template<typename data_type>
Stack<data_type>::Stack()
{
        top_node=NULL;
}
template<typename data_type>
void Stack<data_type>:: push(const data_type &item)
```

```
{
	Node<data_type> *s=NULL;
	s=new Node<data_type>;
	s->data=item;
	s->next=top_node;
	top_node=s;

}
template<typename data_type>
bool Stack<data_type>:: empty()const
{
	return top_node==NULL;
}
template<typename data_type>
Stack<data_type>::~Stack()
{
	while(!empty())
		pop();
}
template<typename data_type>
void Stack<data_type>::pop()
{
   Node<data_type>*p=top_node;
   top_node=top_node->next;
   delete p;
}
template<typename data_type>
data_type Stack<data_type>::top()
{
	data_type x=top_node->data;
	return x;
}
```

(6)Game.cpp

```
#include "utility.h"
#include "Game.h"
#include "LinkQueue.hpp"
#include "LinkStack.hpp"
Game::Game(int num)
{
	card_num = num;
	memset(state, 0, sizeof state);
}
void Game::init()
```

```
{
	std::cout << "请输入第一名玩家的" << card_num << "张牌" << endl;
	for (int i = 1; i <= card_num; i++)
		cin >> player_a[i];
	std::cout << "第一名玩家数据输入完成" << endl;
	std::cout << endl;
	std::cout << "请输入第二名玩家的" << card_num << "张牌" << endl;
	for (int i = 1; i <= card_num; i++)
		cin >> player_b[i];
	std::cout << "第二名玩家数据输入完成" << endl;
}
void Game::work()
{
	Queue<int>a_card, b_card;
	Stack<int>Table;
	int count = 0;

	for (int i = 1; i <= card_num; i++)
		a_card.push(player_a[i]);
	for (int i = 1; i <= card_num; i++)
		b_card.push(player_b[i]);

	while (!a_card.empty() && !b_card.empty())
	{
		std::cout << "count    #"<< count << endl<<endl;
		if (count % 2 == 0)//A 出牌
		{
			count++;
			auto t = a_card.front(); a_card.pop();
			std::cout << "A 出" << t << " state[" <<t<<"]="<< state[t] << endl;
			Table.push(t);
			if (state[t] == 1)
			{
				Stack<int>temp;
				Table.pop();
				temp.push(t);
				while (Table.top() != t)
				{
					auto Top = Table.top(); Table.pop();
					state[Top] = 0;
					temp.push(Top);
					std::cout << "A 赢" << Top << endl;

				}
```

```
            auto Top = Table.top(); Table.pop();
            state[Top] = 0;
            temp.push(Top);
            std::cout << "A 赢" << Top << endl;
            while (!temp.empty())
            {
                auto now = temp.top(); temp.pop();
                a_card.push(now);
            }
        }
        else
        {

            state[t] = 1;
        }
    }
    else
    {
        count++;
        auto t = b_card.front(); b_card.pop();
        std::cout << "B 出" << t << " state[" << t << "]=" << state[t] << endl;
        Table.push(t);
        if (state[t] == 1)
        {
            Stack<int>temp;
            Table.pop();
            temp.push(t);
            while (Table.top() != t)
            {
                auto Top = Table.top(); Table.pop();
                state[Top] = 0;
                temp.push(Top);
                std::cout << "B" << Top << endl;

            }

            auto Top = Table.top(); Table.pop();
            state[Top] = 0;
            temp.push(Top);
            std::cout << "B 赢" << Top << endl;

            while (!temp.empty())
            {
                auto now = temp.top(); temp.pop();
```

```
                    b_card.push(now);
                }
            }
            else
            {

                state[t] = 1;
            }
        }
        std::cout                                                                                    <<
"--------------------------------------------------------------------------------"
<< endl;
    }

    if (a_card.empty())
    {
        std::cout << "B win" << endl;
        std::cout << "B 手中剩余的牌为:";
        while (!b_card.empty())
        {
            std::cout << b_card.front() << " ";
            b_card.pop();
        }

    }
    else
    {
        std::cout << "A win" << endl;
        std::cout << "A 手中剩余的牌为:";
        while (!a_card.empty())
        {
            std::cout << a_card.front() << " ";
            a_card.pop();
        }

    }
    std::cout << endl;
    std::cout << "此时桌上剩余的牌为:";
    while (!Table.empty())
    {
        std::cout << Table.top() << " ";
        Table.pop();
    }
```

```
}
```

(7)main.cpp

```
#include "Game.h"
#include "utility.h"
int main()
{
	//5
	//2 3 5 6 1
	//1 5 4 2 9
	cout << "请输入纸牌数量" << endl;
	int num;
	cin >> num;
	Game A(num);
	A.init();
	A.work();
	return 0;
}
```