苏 州 大 学 实 验 报 告

院、系	计算机学院	年级专业	19 计算机类	姓名	张昊	学号	1927405160
课程名称	数据结构课程实践					成绩	
指导教师	孔芳	同组实验者	无		实验日期	2020 年 10 月 15 日	

实 验 名 称 小猫钓鱼纸牌游戏

一、实验目的

通过本次实验要达到如下目的：

1. 熟悉栈“后进先出”的操作特性；
2. 掌握栈基本操作的实现；
3. 熟悉队列“先进先出”的操作特性；
4. 掌握队列基本操作的实现；
5. 掌握应用栈和队列解决实际的方法。

二、问题描述及要求

小猫钓鱼纸牌游戏

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。

三、概要设计

1. 问题简析

本次实验内容是一个实际问题。问题中的关键是纸牌的保存问题，以及整个游戏流程的控制。对于纸牌，可以看到出牌位第一张牌，赢排放在末尾，手中的牌满足队列“先进先出”的特性；桌面上的牌，最后打出的总会最先被取走，满足栈“后进先出”的操作特性。

2. 程序结构设计思路

程序采用设计一个可以模拟这一纸牌游戏的模块的方案，在主函数中给出测试程序

（代码中给出一个例子）来测试程序的输出。在这一游戏的模块中，设计两个关键函数，分别是判断是否赢排和游戏主流程的函数，并将后者提供给使用方作为接口调用。此外，为了表示获胜情况，程序设计了一个枚举类 state 来枚举胜利者的情况。以上代码放置于 solution.hpp 文件中的 fishing_game 命名空间中。

程序复用了链队列和顺序栈这两个现有数据结构。考虑到本次实验是基于这两种数据结构的应用，故其实现细节在报告中不再赘述。有关两个数据结构的实现，可见源代码文件夹 src 中 util/container 文件夹下的代码。

四、详细设计

1 判断是否赢排算法的设计

对于这个纸牌游戏，其操作对象有两类：手中的牌和桌面上的牌；对于手中的牌来说，其操作也有两类：出牌和赢牌。由题意，每次出掉的时手中的第一张牌，赢排时牌依次放在手牌的末尾，显然手中的牌满足“先进先出”的特性，可以抽象为队列。而对于桌面上的牌，最后打出的总会被最先取走，满足“后进先出”的操作特性，故可以用栈这一数据结构来模拟。这样，出牌可以用手牌队列出队、桌面压入栈来模拟；赢排可以用桌面弹出栈、手牌队列入队来模拟。

另外设计一个记牌器记录桌面上已有的各个数字的数量，用数组来实现。当出牌后某一张牌的数量达到 2，就说明有相同的牌，则赢牌：将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。由于需要按次序放置，而栈不断出栈得到的扑克牌序列是原入栈次序的倒序，需要利用一个临时的栈来将整个序列进行一遍入栈和出栈，使其顺序倒转，同时按正确的顺序放入手牌队列。

以上流程则为每位同学分别进行出牌并判断是否赢牌的 `playCard` 算法，可由如下C++代码描述：

```
1.  void playCard(Queue<int> &student, Stack<int> &desk, int counter[]) {
2.      int now = student.front();  // 出牌
3.      desk.push(student.pop());
4.      counter[now]++;
5.      if (counter[now] == 2) {  // 赢排
6.          Stack<int> temp;
7.          do {
8.              counter[desk.top()]--;
9.              temp.push(desk.pop());
10.         } while (desk.top() != now);
11.         counter[desk.top()]--;
12.         temp.push(desk.pop());
13.         while (!temp.empty()) {  // 保证顺序
14.             student.push(temp.pop());
15.         }
16.     }
17. }
```

2 游戏流程设计

再来考虑游戏的流程，即获胜的情况。显然，当一位玩家的手牌队列为空时，则其对手获胜。这样，整个游戏的流程就变为了两名玩家依次出牌，同时判断是否赢牌（playCard），每位玩家结束出牌阶段后，判断其手牌队列是否为空，若为空则输（对手赢），若都不为空则循环这一过程。以上流程可由如下 C++代码描述：

```
template<int cardsNumberUpperBound>
state solution(Queue<int> &stu1, Queue<int> &stu2, Stack<int> &desk) {
    int counter[cardsNumberUpperBound]{};   // 计牌器
    int count = 0;
    std::cout << "Start\n";
    showState(stu1, stu2, desk);  // 初始状态
    for (;;) {
        std::cout << "Round #" << ++count << std::endl;
        playCard(stu1, desk, counter);
        if (stu1.empty()) {
            showState(stu1, stu2, desk);    // 结束状态输出
            return state::student2_win;
        }
        playCard(stu2, desk, counter);
        if (stu2.empty()) {
            showState(stu1, stu2, desk);  // 结束状态输出
            return state::student1_win;
        }
        showState(stu1, stu2, desk);  // 中间状态输出
    }
}
```

其中 showState 函数输出各玩家的手牌以及桌面上的卡牌情况。函数使用了非类型模板参数 cardsNumberUpperBound，为卡牌编号的上界，作为计牌数组的大小，由函数使用者给出。

五、实验结果测试与分析

main 函数给出了一个样例：

```
int main() {
    // a sample example
    int student1Cards[5] = {2, 3, 5, 6, 1};
    int student2Cards[5] = {1, 5, 4, 2, 9};
    Queue<int> student1, student2;
    Stack<int> desk;
    initializeQueue<int>(student1, student1Cards, 5);
    initializeQueue<int>(student2, student2Cards, 5);
    switch (fishing_game::solution<10>(student1, student2, desk)) {
```

```
        case fishing_game::state::student1_win:
            cout << "Student1 win the game.\n";
            break;
        case fishing_game::state::student2_win:
            cout << "Student2 win the game.\n";
            break;
    }
    return 0;
}
```

其运行结果为：第一位同学（A）获胜。（具体流程如下）

```
Student1: A and Student2: B are playing the card game called Cat Fishing.
The cards on the desk:
The cards in student1's hand: 2 3 5 6 1
The cards in student2's hand: 1 5 4 2 9
Round #1
The cards on the desk: 2 1
The cards in student1's hand: 3 5 6 1
The cards in student2's hand: 5 4 2 9
Round #2
The cards on the desk: 2 1 3 5
The cards in student1's hand: 5 6 1
The cards in student2's hand: 4 2 9
Round #3
The cards on the desk: 2 1 3 4
The cards in student1's hand: 6 1 5 5
The cards in student2's hand: 2 9
Round #4
The cards on the desk:
The cards in student1's hand: 1 5 5
The cards in student2's hand: 9 2 1 3 4 6 2
Round #5
The cards on the desk: 1 9
The cards in student1's hand: 5 5
The cards in student2's hand: 2 1 3 4 6 2
Round #6
The cards on the desk: 1 9 5 2
The cards in student1's hand: 5
The cards in student2's hand: 1 3 4 6 2
Round #7
The cards on the desk:
The cards in student1's hand: 5 2 5
The cards in student2's hand: 3 4 6 2 1 9 1
Round #8
The cards on the desk: 5 3
The cards in student1's hand: 2 5
The cards in student2's hand: 4 6 2 1 9 1
Round #9
```

```
The cards on the desk: 5 3 2 4
The cards in student1's hand: 5
The cards in student2's hand: 6 2 1 9 1
Round #10
The cards on the desk: 6
The cards in student1's hand: 5 3 2 4 5
The cards in student2's hand: 2 1 9 1
Round #11
The cards on the desk: 6 5 2
The cards in student1's hand: 3 2 4 5
The cards in student2's hand: 1 9 1
Round #12
The cards on the desk: 6 5 2 3 1
The cards in student1's hand: 2 4 5
The cards in student2's hand: 9 1
Round #13
The cards on the desk: 6 5 9
The cards in student1's hand: 4 5 2 3 1 2
The cards in student2's hand: 1
Round #14
The cards on the desk: 6 5 9 4 1
The cards in student1's hand: 5 2 3 1 2
The cards in student2's hand:
Student1 win the game.
```

六、附录

1. 源代码路径：C++源代码位于附件中 src 目录下。

2. 文件编码：UTF-8；行分隔符：LF（\n）。

3. 实验环境：Linux 操作系统，g++编译器，基于 cmake 构建；在 CLion 集成开发环境中调试运行通过。手动编译运行方法为（在 Linux/Unix shell 中）：

```
$ mkdir build  # pwd: src/
$ cd build
$ cmake ..
$ make
$ ./CatFishing
```