《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 软件工程 | 姓名 | 梅子羽 | 学号 | 2327406107 |
| 实验布置日期 | | 2024.10.14 | | 提交  日期 | 2024.10.24 | | 成绩 |  |

课程实践实验4：小猫钓鱼游戏

## 一、问题描述及要求

[问题描述]

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。

编写程序，利用栈和队列，判断谁是胜者。

## 二、概要设计

#### （1）对实验内容的理解

本实验实现了一个基于链表的数据结构，主要包括栈（Stack）和队列（Queue），并通过简单的游戏逻辑展示了这些数据结构的实际应用。通过实现出牌规则和卡片管理，加深了对数据结构和算法的理解。

#### （2）系统功能列表

* **栈功能**
  + 入栈（push）
  + 出栈（pop）
  + 查看栈顶元素（peek）
  + 检查栈是否为空（isEmpty）
  + 打印栈元素（printOut）
* **队列功能**
  + 入队（push）
  + 出队（pop）
  + 查看队头元素（peek）
  + 检查队列是否为空（isEmpty）
  + 打印队列元素（printOut）
* **游戏逻辑**
  + 输入玩家的卡片信息
  + 处理出牌逻辑
  + 显示当前状态（玩家卡片和桌面卡片）

#### （3）程序运行的界面设计

1. **主界面**：
   * 欢迎信息和游戏规则
   * 输入玩家1和玩家2的卡片数量及具体卡片
   * 开始游戏按钮
2. **游戏进行界面**：
   * 显示玩家手中的卡片和桌面上的卡片
   * 出牌操作提示
   * 显示游戏状态（得分、胜利信息）

#### （4）总体设计思路

* **数据结构**：
  + **链表（ListNode）**：作为栈和队列的基础结构，便于动态管理元素。
  + **栈（Stack）**：实现后进先出（LIFO）行为，使用链表的头指针作为栈顶。
  + **队列（Queue）**：实现先进先出（FIFO）行为，维护前后两个指针（front和rear）。
* **类的设计**：
  + **ListNode**：
    - 属性：存储数据和指向下一个节点的指针。
  + **Stack**：
    - 属性：链表头指针。
    - 方法：
      * void push(Datatype x): 入栈。
      * Datatype pop(): 出栈。
      * Datatype peek(): 查看栈顶元素。
      * bool isEmpty(): 检查是否为空。
      * void printOut(): 打印栈内容。
  + **Queue**：
    - 属性：队头指针和队尾指针。
    - 方法：
      * void push(Datatype x): 入队。
      * Datatype pop(): 出队。
      * Datatype peek(): 查看队头元素。
      * bool isEmpty(): 检查是否为空。
      * void printOut(): 打印队列内容。
* **类之间的关系**：
  + Stack和Queue类依赖于ListNode类作为其基本构建块。

#### （5）程序结构设计

* **现有程序使用**：
  + 使用ListNode类作为基本单元，构建Stack和Queue。
* **设计程序文件**：
  + ListNode.h和ListNode.cpp：定义链表节点类。
  + Stack.h和Stack.cpp：定义栈类及其操作。
  + Queue.h和Queue.cpp：定义队列类及其操作。
  + main.cpp：实现游戏逻辑和用户交互。
* **各部分关系描述**：
  + main.cpp依赖于Stack和Queue类，通过这些数据结构实现游戏的逻辑和状态管理。Stack和Queue都使用ListNode作为元素的存储结构。

### 数据结构与算法

#### 数据结构选择

* **链表（ListNode）**：
  + **理由**：动态性强，适合频繁的插入和删除操作。
* **栈（Stack）**：
  + **理由**：实现简单，支持快速的入栈和出栈操作。
* **队列（Queue）**：
  + **理由**：通过链表实现先进先出，适合处理玩家出牌顺序。

#### 算法选择

* **游戏逻辑**：使用哈希表管理玩家的出牌状态，快速判断卡片的使用情况，确保游戏流程的高效性。

通过以上设计，整个系统结构清晰，功能完备，便于扩展和维护。

## 三、详细设计

#### 1. 主函数设计

主函数的主要任务是：

* 初始化游戏环境
* 获取用户输入
* 调用游戏逻辑处理函数

**流程图**：

开始

|

V

显示欢迎信息和规则

|

V

获取玩家1的卡片数量和卡片信息

|

V

获取玩家2的卡片数量和卡片信息

|

V

调用游戏逻辑处理函数

|

V

游戏结束或继续

|

V

结束

#### 2. 关键算法设计

**游戏逻辑处理函数（catFishingGame）**

该函数负责核心游戏逻辑，处理两名玩家的出牌过程。其主要流程如下：

**流程图**：

开始游戏循环

|

V

检查玩家1和玩家2是否有卡片

| |

| V

| 玩家2出牌

| |

| V

| 判断卡片是否冲突

| |

| V

| 处理冲突（卡片收回）

| |

| V

| 更新游戏状态

| |

| V

| 显示状态信息

|

V

玩家1出牌

|

V

判断卡片是否冲突

|

V

处理冲突（卡片收回）

|

V

更新游戏状态

|

V

显示状态信息

|

V

检查是否有胜者

|

V

结束游戏循环

#### 3. 难点分析

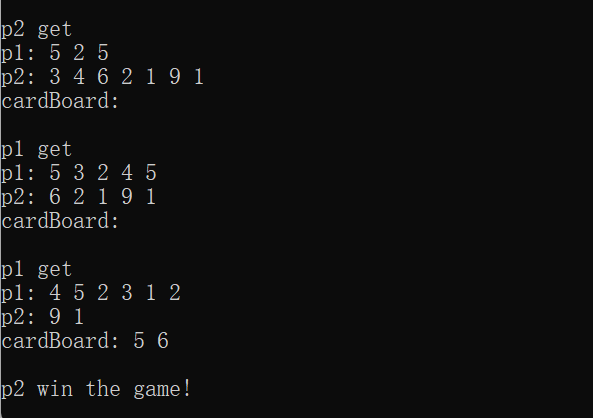
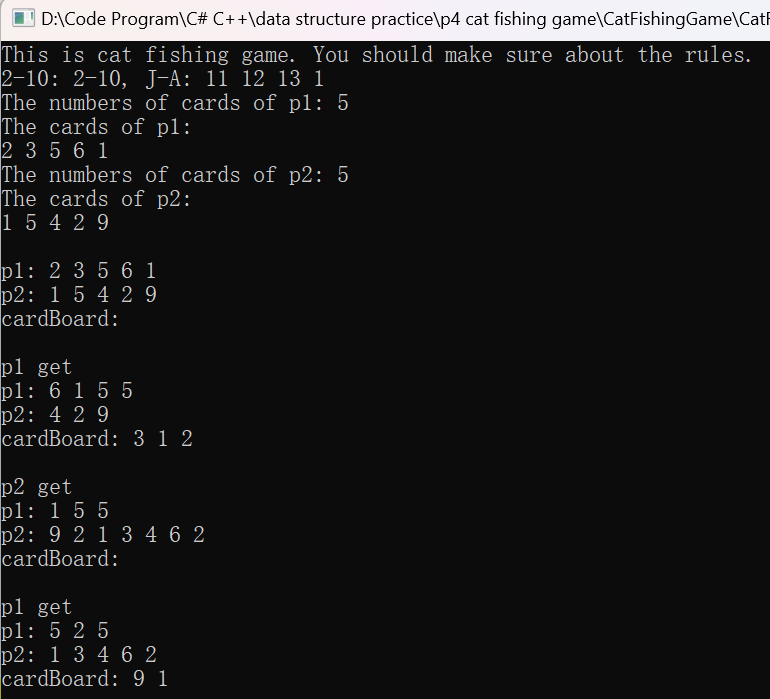
* **卡片冲突处理**：
  + 在出牌过程中，玩家可能会出与桌面上已有卡片相同的卡片，此时需要进行冲突处理。实现这一功能的关键在于使用哈希表（mymap）来快速查找冲突卡片，并通过栈（cardBasket）来暂存被冲走的卡片。
  + 当发生冲突时，算法需要找到所有在桌面上的卡片，直到找到出牌的卡片为止，然后将这些卡片收回到玩家手中。
* **游戏状态管理**：
  + 每轮出牌后，必须更新当前的游戏状态，确保显示的内容能够反映最新的卡片分布和玩家的手牌。通过函数调用（如allPrint）来输出当前状态，增强用户体验。
* **胜负判定**：
  + 需要在每轮出牌后判断是否有玩家胜利。若一名玩家手中没有卡片，则游戏结束并宣告该玩家胜利。

## 四、实验结果

测试样例一：person1:2 3 5 6 1

person2:1 5 4 2 9

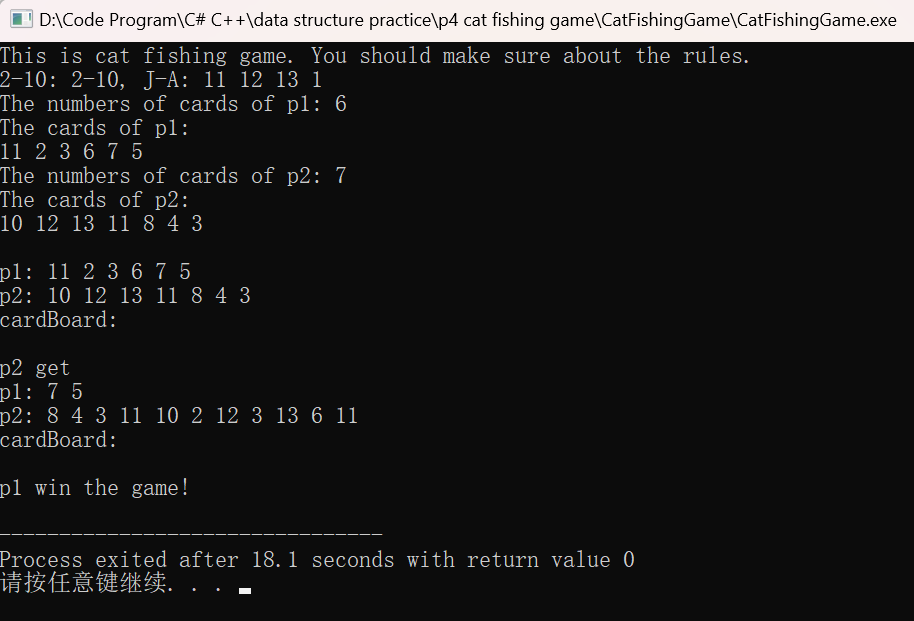
该样例为题目所给样例，目的是为了测试代码方法的可行性，根据手动模拟得出，确实是person2胜出，故输出正确



测试样例二：person1:11 2 3 6 7 5

person2:10 12 13 11 8 4 3

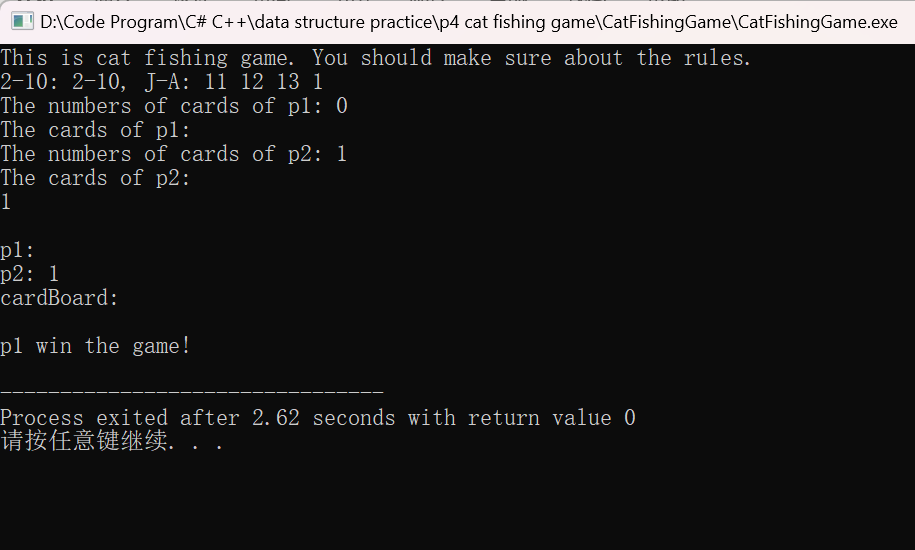
该样例为普通样例，目的是测试代码方法的概括性，根据手动模拟得出，确实是person1胜出，故输出正确



测试样例三：person1:

person2:1

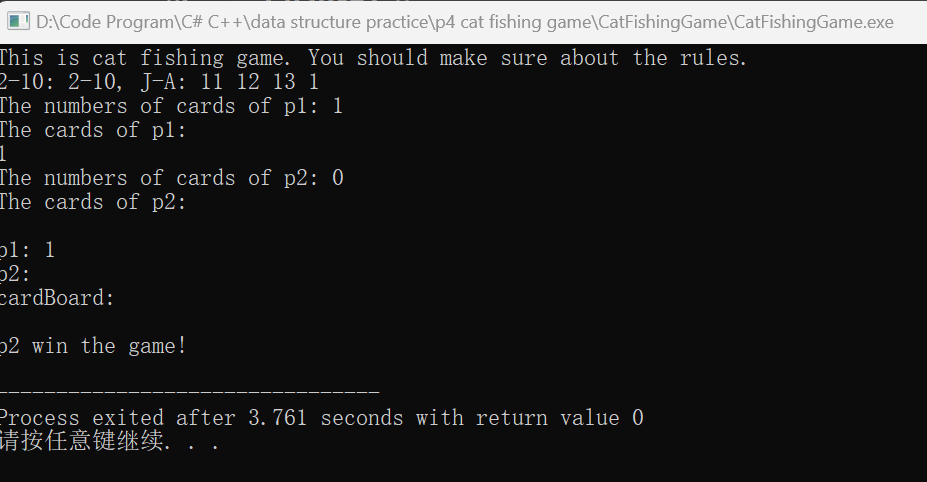
该样例为极端样例，目的是测试代码对于特殊情况的适应能力，即开局person1就没有牌，故person1直接胜出，输出正确



测试样例四：person1:1

person2:

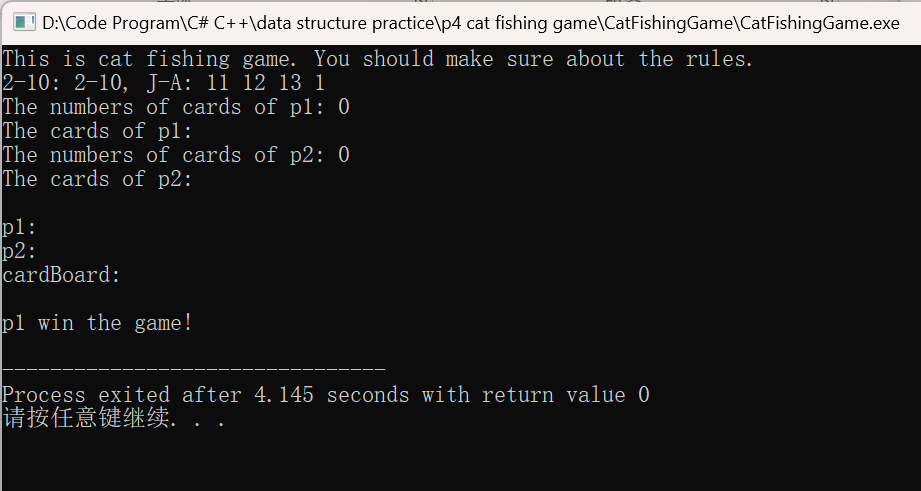
该样例为极端样例，目的同测试样例三，person2胜出，输出正确



测试样例五：person1：

person2：

该样例为极端样例，目的同三，四。由于是person1先出牌，故是person1胜出，输出正确



此次实验实际上是一个模拟题，即利用栈和队列的性质来模拟一个生活中的小游戏，涉及到的算法不多，一是利用哈希表mymap来进行牌是否在牌桌上的判断，这里的时间复杂度是由c++内置的查找函数决定的，一般是O(logn)；二是利用两个栈来进行牌从牌桌上抽离再依次放入玩家手牌，设person1的卡牌数量为m，person2的卡牌数量为n，则应为O(m\*n)。代码整体的时间复杂度即为O(n^2)，极大可能是不能继续优化了，因为这就是模拟实际情况最直接的处理，笔者也没有再想出其他的方法。

## 五、小结

在本次设计与实现过程中，成功构建了一个基于链表实现的栈和队列结构，设计了一个简单的“猫钓鱼”游戏。游戏的核心逻辑包括玩家出牌、处理冲突、以及胜负判定等功能。整个过程让我深入理解了数据结构的应用和面向对象设计的思路，尤其是在游戏逻辑中如何有效地使用数据结构来管理状态和操作。

### 完成的内容

1. **数据结构实现**：
   * 成功实现了基于链表的栈（Stack）和队列（Queue）类，并进行了基本操作的封装。
2. **游戏逻辑**：
   * 实现了“猫钓鱼”游戏的基本流程，包括卡牌输入、出牌、冲突处理及胜负判定。
3. **用户界面**：
   * 设计了简单的命令行界面，用于获取用户输入并显示游戏状态。

### 未完成的内容

1. **无限循环处理**：
   * 当前实现未加入处理无限循环的机制，可能导致游戏在特定情况下无法结束。
2. **更丰富的用户界面**：
   * 可以增加图形用户界面（GUI）或更友好的命令行交互，提升用户体验。
3. **更完善的游戏规则**：
   * 进一步扩展游戏规则，比如引入更多种类的卡牌和复杂的游戏策略。

### 程序的局限性

1. **输入限制**：
   * 当前程序对输入格式没有严格的校验，用户输入错误可能导致程序崩溃或运行不稳定。
2. **数据结构的扩展性**：
   * 当前实现主要针对整数卡牌，若未来想支持其他数据类型，需要进行修改。
3. **性能问题**：
   * 在卡牌数量较多时，处理冲突的复杂度可能导致游戏性能下降。

### 展望

未来可以在以下几个方面进行学习与完善：

1. **深入学习数据结构与算法**：
   * 继续探索更高效的算法，以提升游戏性能，特别是在处理冲突和状态管理方面。
2. **面向对象设计**：
   * 加强对面向对象设计原则的理解，如单一职责原则、开闭原则等，以提高代码的可维护性和扩展性。
3. **图形化界面设计**：
   * 学习如何使用图形化库（如Qt或SFML）为游戏构建图形用户界面，使游戏更具吸引力。

## 附录：源代码

**1、实验环境：Dev-C++ 5.11 C++11标准**

2、

（1）//ListNode.h

#ifndef LISTNODE\_H

#define LISTNODE\_H

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

template <typename Datatype>

struct ListNode

{

Datatype val;

ListNode\* next;

ListNode() : val(), next(nullptr) {}

ListNode(Datatype x) : val(x), next(nullptr) {}

};

#endif

（2）//Queue.h

#ifndef QUEUE\_H

#define QUEUE\_H

#include "ListNode.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

template <typename Datatype>

class Queue

{

private:

ListNode<Datatype>\* front;

ListNode<Datatype>\* rear;

public:

Queue();

void push(Datatype);

Datatype pop();

Datatype peek() const;

bool isEmpty() const;

void printOut() const;

};

#endif

（3）//Queue.cpp

#include "Queue.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

template <typename Datatype>

Queue<Datatype>::Queue() : front(nullptr), rear(nullptr) {}

template <typename Datatype>

void Queue<Datatype>::push(Datatype x)

{

ListNode<Datatype>\* temp = new ListNode<Datatype>(x);

if (rear)

{

rear->next = temp;

}

else

{

front = temp;

}

rear = temp;

}

template <typename Datatype>

Datatype Queue<Datatype>::pop()

{

if (front == nullptr)

{

cout << "Queue is empty!" << endl;

return Datatype();

}

else

{

Datatype num = front->val;

ListNode<Datatype>\* temp = front;

front = front->next;

if (front == nullptr)

{

rear = nullptr;

}

delete temp;

return num;

}

}

template <typename Datatype>

Datatype Queue<Datatype>::peek() const

{

if (front == nullptr)

{

cout << "Queue is empty!" << endl;

return Datatype();

}

else

{

return front->val;

}

}

template <typename Datatype>

bool Queue<Datatype>::isEmpty() const

{

return (front == nullptr);

}

template <typename Datatype>

void Queue<Datatype>::printOut() const

{

ListNode<Datatype>\* pc = front;

while (pc != nullptr)

{

cout << pc->val << " ";

pc = pc->next;

}

cout << endl;

}

template class Queue<int>;

template class Queue<double>;

template class Queue<string>;

（4）//Stack.h

#ifndef STACK\_H

#define STACK\_H

#include "ListNode.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

template <typename Datatype>

class Stack

{

private:

ListNode<Datatype>\* p;

public:

Stack();

void push(Datatype)const;

Datatype pop();

Datatype peek()const;

bool isEmpty()const;

void printOut()const;

};

#endif

（5）//Stack.cpp

#include "Stack.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

template <typename Datatype>

Stack<Datatype>::Stack()

{

p = new ListNode<Datatype>();

}

template <typename Datatype>

void Stack<Datatype>::push(Datatype x)const

{

ListNode<Datatype>\* temp = new ListNode<Datatype>(x);

temp->next = p->next;

p->next = temp;

}

template <typename Datatype>

Datatype Stack<Datatype>::pop()

{

if(p->next == nullptr)

{

cout << "Stack is empty!" << endl;

return 0;

}

else

{

Datatype num = p->next->val;

ListNode<Datatype>\* temp = p->next;

p->next = p->next->next;

delete temp;

return num;

}

}

template <typename Datatype>

Datatype Stack<Datatype>::peek()const

{

if(p->next == nullptr)

{

cout << "Stack is empty!" << endl;

return 0;

}

else

{

return p->next->val;

}

}

template <typename Datatype>

bool Stack<Datatype>::isEmpty()const

{

return (p->next == nullptr);

}

template <typename Datatype>

void Stack<Datatype>::printOut()const

{

ListNode<Datatype>\* pc = p->next;

while(pc != nullptr)

{

cout << pc->val << " ";

pc = pc->next;

}

cout << endl;

}

template class Stack<int>;

template class Stack<double>;

template class Stack<string>;

（6）//main.cpp

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

#include "Stack.h"

#include "Queue.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

void allPrint(Queue<int> &p1, Queue<int> &p2, Stack<int> &cardBoard)

{

cout << "p1: ";

p1.printOut();

cout << "p2: ";

p2.printOut();

cout << "cardBoard: ";

cardBoard.printOut();

cout << endl;

}

void catFishingGame()

{

map<int, int> mymap;

Queue<int> p1, p2;

Stack<int> cardBoard, cardBasket;

int n1, n2, card1, card2, card;

cout << "This is cat fishing game. You should make sure about the rules.\n";

cout << "2-10: 2-10, J-A: 11 12 13 1\n";

cout << "The numbers of cards of p1: ";

cin >> n1;

cout << "The cards of p1:\n";

for(int i = 0; i < n1; i ++)

{

cin >> card1;

p1.push(card1);

}

cout << "The numbers of cards of p2: ";

cin >> n2;

cout << "The cards of p2:\n";

for(int i = 0; i < n2; i ++)

{

cin >> card2;

p2.push(card2);

}

cout << endl;

allPrint(p1, p2, cardBoard);

if(p1.isEmpty())

cout << "p1 win the game!\n";

else if(p2.isEmpty())

cout << "p2 win the game!\n";

while(!p1.isEmpty() && !p2.isEmpty())

{

card1 = p1.pop();

if(mymap.count(card1))

{

while(cardBoard.peek() != card1)

cardBasket.push(cardBoard.pop());

cardBasket.push(cardBoard.pop());

while(!cardBasket.isEmpty())

{

card = cardBasket.pop();

p1.push(card);

mymap.erase(card);

}

p1.push(card1);

cout << "p1 get\n";

allPrint(p1, p2, cardBoard);

}

else

{

if(p1.isEmpty())

{

cout << "p1 win the game!\n";

break;

}

mymap[card1] = 1;

cardBoard.push(card1);

}

card2 = p2.pop();

if(mymap.count(card2))

{

while(cardBoard.peek() != card2)

cardBasket.push(cardBoard.pop());

cardBasket.push(cardBoard.pop());

while(!cardBasket.isEmpty())

{

card = cardBasket.pop();

p2.push(card);

mymap.erase(card);

}

p2.push(card2);

cout << "p2 get\n";

allPrint(p1, p2, cardBoard);

}

else

{

if(p2.isEmpty())

{

cout << "p2 win the game!\n";

break;

}

mymap[card2] = 1;

cardBoard.push(card2);

}

}

}

int main(int argc, char\*\* argv)

{

catFishingGame();

return 0;

}