**《数据结构》实验报告（第三章）**

**学号：\_\_\_\_\_\_58122231\_\_\_ 姓名 ：\_\_\_\_\_陆文韬\_\_\_\_\_\_\_**

**实验题号： P167 1 实验日期：\_2023.11.21\_\_\_\_**

**实验一**

**1．问题描述：**描述实验内容和要求以及需要解决的问题。

人们花费了大量的时间玩扑克牌游戏，而赌场也因为人性的这个弱点大量获利。有一种纸牌游戏如下所述。任务是编写一个计算机程序来进行这个游戏，从而节约人们大量的时间以进行其他更有用的工作。现将 28张牌分作7堆,最左边的第1堆放一张牌,第2堆放两张牌，依此类推。最右边的第7堆放7张牌。此7堆牌称作出牌堆。余下的牌正面朝下放在另一边,称作闲置堆。出牌堆中仅有最顶上的一张牌正面朝上。根据每堆顶部正面朝上的牌的颜色(黑桃、梅花属黑色,红心、方块属红色)和点数,可将一个堆上正面朝上的牌移到另一个堆上。移动规则是:颜色不同,而被移动牌的点数少1。例如，可将方块8或红心8移至黑桃9或梅花9的上面。也就是说,如果一个堆顶部正面朝上的牌是黑桃9或梅花9,则可在上面放置另一个堆顶部正面朝上的红心8或方块8。一个堆中顶部正面朝上的牌被移走后,可将下面一张正面朝下的牌翻成正面朝上。而后再根据上述规则进行移动。当出牌堆中无牌可移动时，则将闲置堆上正面朝下的牌翻成正面朝上,如果翻开的牌与出牌堆中任一堆中顶部牌符合上述关系(即颜色不同,点数少1)即可将这张牌移至相应堆中,放置顶部。否则继续翻闲置堆中的牌。如果其中一个出牌堆完全移空了,可将其他出牌堆中正面朝上的“国王”(K)连同它上面的所有牌一起移动到该空堆处。从闲置堆中翻到的“国王”也可移至出牌堆中的空堆处。设置4个输出堆,分别对应4种牌面(黑桃、梅花、方块和红心)。每当出牌堆或闲置堆翻开的正面牌中出现A牌时,这张A牌就是输出堆中相应一组的第1张牌。以后可将点数游戏的目标是将尽可能多的牌放入输出堆中。 为2~K的牌放人输出堆中。输出规则是相同牌面按A~K的顺序放入输出堆中。在游戏过程中,如果有多于一种可能出牌方式时,就应遵循下述操作顺序:

(a)从出牌堆或闲置堆的顶部取出一张符合输出规则的牌放入相应的输出堆中。

(b)当某一出牌堆成为空堆时,若出牌堆中已翻出的牌中有“国王”时,将“国王”连同上面的牌一起移至空堆处;若无,则在闲置堆中寻找“国王”移至空堆处。

(c)出牌堆之间按规则移动。

(d) 依次进行(a)、(b)、(c),只要发生牌的移动操作,则重新从(a)开始。

(e) 如果从(a)~(d)未发生牌的移动,则从闲置堆中翻牌,按规则放到出牌堆顶部。并重新从(a)开始。

只有出牌堆或闲置堆中顶部的一张牌可以放到输出堆中。一旦被放入输出堆，就不得再取出另作它用。当所有的牌都被放入输出堆中,或上述(a)~(e)中均无法作出任何移动时,游戏结束。

当为赢钱时,玩家首先支付52美元,然后每移动一张牌到输出堆则赢得5美元。编写一个程序，使得可以多次进行上述游戏，并得到你的净收入。用随机数产生器进行洗牌。输出内容应完整并易于理解，要求记录并显示游戏过程中输出堆中牌的张数，移动次数和每次移动的得分数。同样也包括参加的局数和净收入(十或者一)。

**2．算法思想：**详细描述解决相应问题所需要的算法设计思想。

这个程序实际上是一个空当接龙的游戏，在这个程序中，将牌简化为整型数据，通过stack来表示每个牌堆，再用vector数组将多个出牌堆和输出堆串联起来，

再通过实现一系列功能函数完成这个游戏。

**3．功能函数：**描述所设计的功能函数。如果有多个函数，需要描述它们之间的关系。

outputCard：这个函数用于选取符合要求的牌从出牌堆和闲置堆的顶部移动到输出堆的顶部

isEmptyHeap: 这个函数用于判断出牌堆有没有空堆

haveKing：判断这个牌堆已经翻开的牌中有没有国王牌

moveKing：将有国王的堆移动到空堆上

judgeMove：判断两个堆之间是否符合移动规则

moveCard：在出牌堆之间按规则移动

moveFreeHeap：闲置堆翻牌并放置到出牌堆

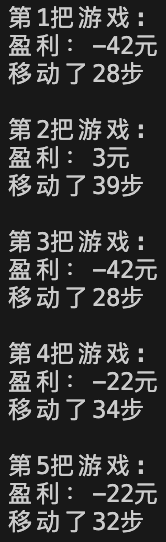
**4．测试数据：**设计测试数据，或具体给出测试数据。

提示：要求测试数据能全面地测试所设计程序的功能。

输入要游玩游戏的次数

****

**5．测试情况：**给出程序的测试情况，分析运行结果，显示实验结果截图。

****

**6．实验总结：**写出实验过程中遇到的问题，以及问题的解决过程。总结实验心得体会。

在程序编写的过程中，因为涉及到很多关于栈的操作，在操作的时候很容易因为粗心导致数组越界的问题，还多次出现了未检测栈是否为空栈就调用top函数导致程序崩溃的问题，问题解决的方式就是采用输入输出检测和断点监测变量情况的方法，在分析问题的时候要考虑周到，充分考虑可能出现的各种意外情况。同时还要读懂题意，理解对题目的意思。

**7. 源代码：**给出项目所有源程序清单。

建议：源程序中应有充分的注释，例如注释每个函数参数的含义、函数返回值的含义、函数的功能、主要语句段的功能，等等。

实验源程序清单

game\_func.h

#pragma once

using namespace std;

// 判断在出牌堆和闲置堆中的顶部是否有符合输出规则的牌，如果有，则放入对应的输出堆中

// suit代表花色，方块为0，红心为1，黑桃为2，梅花为3

bool outputCard(stack<int> &currentHeap, stack<int> &outputHeap, int suit)

{

// 判断是否符合输出规则

bool operation = 0;

if (currentHeap.empty())

return operation;

if (currentHeap.top() == suit \* 13 + outputHeap.size())

{

outputHeap.push(currentHeap.top());

currentHeap.pop();

operation = 1;

}

return operation;

}

// 判断出牌堆有没有空堆

int isEmptyHeap(stack<int> &currentHeap, int heapNum)

{

if (currentHeap.empty())

return heapNum;

else

return -1;

}

// 判断这个牌堆有没有国王牌

bool haveKing(stack<int> &currentHeap,int flag[])

{

stack<int> temp = currentHeap;

bool flag1 = false;

while (!temp.empty()&&flag[temp.top()]==1)

{

if ((temp.top() + 1) % 13 == 0)

{

flag1 = true;

break;

}

temp.pop();

}

if (flag1)

{

if (!temp.empty())

return 1;

}

return 0;

}

// 将有国王的堆移到空堆上

bool moveKing(stack<int> &kingStack, stack<int> &emptyStack)

{

stack<int> temp;

while ((kingStack.top() + 1) % 13 != 0)

{

temp.push(kingStack.top());

kingStack.pop();

}

temp.push(kingStack.top());

kingStack.pop();

while (!temp.empty())

{

emptyStack.push(temp.top());

temp.pop();

}

return 1;

}

// 判断两个堆之间是否符合移动规则

int judgeMove(stack<int> &a, stack<int> &b,int flag[])

{

stack<int> temp = a;

int count = 1;

int k = 0;

while(!(temp.empty()||b.empty()))

{

//碰到了未翻开的牌

if(flag[temp.top()]==0)

{

return k;

}

if (temp.top() > 25 && b.top() < 26)

{

if (temp.top() % 13 + 1 == b.top() % 13)

k = count;

}

if (temp.top() < 26 && b.top() > 25)

{

if (temp.top() % 13 + 1 == b.top() % 13)

k = count;

}

temp.pop();

count++;

}

return k;

}

// 出牌堆之间按规则移动

bool moveCard(vector<stack<int> > &arr,int flag[])

{

int temp[7];

for (int i = 0; i < 7; i++)

{

temp[i] = arr[i].size();

}

bool operation = 0;

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < 7; j++)

{

if (i == j)

continue;

if (judgeMove(arr[i], arr[j],flag))

{

int count = judgeMove(arr[i], arr[j], flag);

stack<int> temp;

for(int m=0; m < count; m++)

{

temp.push(arr[i].top());

arr[i].pop();

}

for(int m = 0; m < count; m++)

{

arr[j].push(temp.top());

temp.pop();

}

}

}

}

for (int i = 0; i < 7; i++)

{

if (temp[i] != arr[i].size())

{

operation = 1;

break;

}

}

return operation;

}

// 闲置堆翻牌并放置到出牌堆

bool moveFreeHeap(stack<int> &preHeap, stack<int> &postHeap, vector<stack<int> > &arr,int flag[])

{

// 翻牌

if (preHeap.empty())

{

return 0;

}

else

{

postHeap.push(preHeap.top());

preHeap.pop();

}

// 移动牌到出牌堆

int count;

do

{

count = 0;

for (int i = 0; i < 7; i++)

{

if (judgeMove(postHeap, arr[i],flag))

{

arr[i].push(postHeap.top());

postHeap.pop();

count++;

}

}

} while (count);

return 1;

}

main.cpp

#include <iostream>

#include <stack>

#include <vector>

#include "game\_func.h"

#include <algorithm>

#include <random>

#include <stdio.h>

using namespace std;

// 一共52张牌，按方块（0～12）、红心（13～25）、黑桃（26～38）、梅花（39～51）的顺序用整型表示

int main()

{

// 记录每次盈利

vector<int> money;

vector<int> movements;

int times = 0;

int num;

cout<<"请输入要游玩的次数："<<endl;

cin>>num;

while (times<num)

{

// 创建牌堆

vector<stack<int> > currentHeaps(7);

vector<stack<int> > outputHeaps(4);

stack<int> preFreeHeap;

stack<int> postFreeHeap;

int flag[52]={0};

// 发牌

int cardArray[52];

for (int i = 0; i < 52; i++)

cardArray[i] = i;

shuffle(begin(cardArray), end(cardArray), default\_random\_engine(random\_device()()));

int count = 0;

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < i + 1; j++)

{

currentHeaps[i].push(cardArray[count]);

count++;

}

}

while (count < 52)

{

preFreeHeap.push(cardArray[count]);

count++;

}

//记录已经翻开的牌

for(int i = 0; i < 7; i++)

{

if (!currentHeaps[i].empty())

flag[currentHeaps[i].top()] = 1;

}

if (!postFreeHeap.empty())

flag[postFreeHeap.top()] = 1;

int operationCount = 0;

int sum\_movement = 0;

// 单次游戏循环

while (true)

{

int sum\_output = 0;

// （a）从出牌堆或闲置堆的顶部取出一张符合输出规则的牌放入相应的输出堆中

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < 4; j++)

{

operationCount += outputCard(currentHeaps[i], outputHeaps[j], j);

operationCount += outputCard(postFreeHeap, outputHeaps[j], j);

// 记录已经翻开的牌

for (int i = 0; i < 7; i++)

{

if (!currentHeaps[i].empty())

flag[currentHeaps[i].top()] = 1;

}

if (!postFreeHeap.empty())

flag[postFreeHeap.top()] = 1;

}

}

// （b）当某一出牌堆成为空堆时，若出牌堆已翻出的牌中有“国王”时

// 将“国王”连同上面的牌一起移至空堆处

// 若无，则在闲置堆中寻找“国王”移至空堆处

int emptyHeapNum;

for (int i = 0; i < 7; i++)

{

emptyHeapNum = isEmptyHeap(currentHeaps[i], i);

}

if (emptyHeapNum != -1)

{

for (int i = 0; i < 7; i++)

{

if (haveKing(currentHeaps[i],flag))

{

operationCount += moveKing(currentHeaps[i], currentHeaps[emptyHeapNum]);

// 记录已经翻开的牌

for (int i = 0; i < 7; i++)

{

if (!currentHeaps[i].empty())

flag[currentHeaps[i].top()] = 1;

}

if (!postFreeHeap.empty())

flag[postFreeHeap.top()] = 1;

}

}

if(haveKing(postFreeHeap,flag))

{

operationCount += moveKing(postFreeHeap,currentHeaps[emptyHeapNum]);

// 记录已经翻开的牌

for (int i = 0; i < 7; i++)

{

if (!currentHeaps[i].empty())

flag[currentHeaps[i].top()] = 1;

}

if (!postFreeHeap.empty())

flag[postFreeHeap.top()] = 1;

}

}

// （c）出牌堆之间按规则移动

operationCount += moveCard(currentHeaps,flag);

// 记录已经翻开的牌

for (int i = 0; i < 7; i++)

{

if (!currentHeaps[i].empty())

flag[currentHeaps[i].top()] = 1;

}

if (!postFreeHeap.empty())

flag[postFreeHeap.top()] = 1;

// （d）依次进行（a）（b）（c），只要发生牌的移动操作，则重新从（a）开始

bool gameOver;

sum\_movement+=operationCount;

if (operationCount)

{

operationCount = 0;

continue;

}

// （e）如果从（a）～（d）未发生牌的移动，则从闲置堆中翻牌

// 按规则放到出牌堆顶部。并重新从（a）开始

else

{

gameOver = moveFreeHeap(preFreeHeap, postFreeHeap, currentHeaps,flag);

sum\_movement++;

// 记录已经翻开的牌

for (int i = 0; i < 7; i++)

{

if (!currentHeaps[i].empty())

flag[currentHeaps[i].top()] = 1;

}

if (!postFreeHeap.empty())

flag[postFreeHeap.top()] = 1;

}

// 判断游戏结束

if (!gameOver)

{

for (int i = 0; i < 4; i++)

{

sum\_output += outputHeaps[i].size();

}

money.push\_back(5\*sum\_output-52);

movements.push\_back(sum\_movement);

times++;

break;

}

}

}

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

{

cout<<"第"<<i+1<<"把游戏:"<<endl;

cout<<"盈利："<<money[i]<<"元"<<endl;

cout<<"移动了"<<movements[i]<<"步"<<endl<<endl;

}

}