**★★☆☆☆**

**題組：基礎48題**

**題號：Q127: "Accordian" Patience**

**整理者：呂易澄**

**學號：ACS105111**

**使用語言:C++**

**解題日期：2018年7月22日**



**題目:**

You are to simulate the playing of games of “Accordian” patience, the rules for which are as follows:

Deal cards one by one in a row from left to right, not overlapping. Whenever the card matches its immediate neighbour on the left, or matches the third card to the left, it may be moved onto that card. Cards match if they are of the same suit or same rank. After making a move, look to see if it has made additional moves possible. Only the top card of each pile may be moved at any given time. Gaps between piles should be closed up as soon as they appear by moving all piles on the right of the gap one position to the left. Deal out the whole pack, combining cards towards the left whenever possible. The game is won if the pack is reduced to a single pile.

Situations can arise where more than one play is possible. Where two cards may be moved, you should adopt the strategy of always moving the leftmost card possible. Where a card may be moved either one position to the left or three positions to the left, move it three positions.

**Input**

Input data to the program specifies the order in which cards are dealt from the pack. The input contains pairs of lines, each line containing 26 cards separated by single space characters. The final line of the input file contains a ‘#’ as its first character. Cards are represented as a two character code. The first character is the face-value (A=Ace, 2–9, T=10, J=Jack, Q=Queen, K=King) and the second character is the suit (C=Clubs, D=Diamonds, H=Hearts, S=Spades).

**Output**

One line of output must be produced for each pair of lines (that between them describe a pack of 52 cards) in the input. Each line of output shows the number of cards in each of the piles remaining after playing “Accordian patience” with the pack of cards as described by the corresponding pairs of input lines.

**Sample Input**

QD AD 8H 5S 3H 5H TC 4D JH KS 6H 8S JS AC AS 8D 2H QS TS 3S AH 4H TH TD 3C 6S  
8C 7D 4C 4S 7S 9H 7C 5D 2S KD 2D QH JD 6D 9D JC 2C KH 3D QC 6C 9S KC 7H 9C 5C  
AC 2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC AD 2D 3D 4D 5D 6D 7D 8D TD 9D JD QD KD  
AH 2H 3H 4H 5H 6H 7H 8H 9H KH 6S QH TH AS 2S 3S 4S 5S JH 7S 8S 9S TS JS QS KS  
#

**Sample Output**

6 piles remaining: 40 8 1 1 1 1   
1 pile remaining: 52

**問題描述：**你的任務是模擬一種叫「Accordian」的紙牌遊戲，他的遊戲規則如下：

一副撲克牌有52張牌，首先把紙牌一張一張由左到右排好（不能有重疊，所以共有52堆牌，每堆一張），當某一張牌與他左邊那張牌或者左邊的第三張牌有「Match」的時候，就把這張牌移到那張牌上面去。在這裡兩張牌「Match」指的是這兩張牌的花色（suit）或者點數（rank）一樣。當你做了一個移動之後，要察看是否還可以做其他的移動。在任何時間，只有最上面那張牌可以被移動。如果因為移動一張牌使得產生一個空格（也就是被移動的那堆牌只有一張牌），你必須把右邊所有的牌堆往左移一格。如此不斷的尋找可移動的牌，直到沒有一張牌可以移動遊戲就結束了。

在選擇可以移動的牌的時候可能有些狀況會發生。如果有兩張牌都可以移動，你應該要移動最左邊的那張牌。當一張牌可以被移動到左邊一格，或左邊三格的時候，你必須移動到左邊三格。

**Input**

輸入包含多組測試資料。每組測試資料兩列，每列有26張牌的資料。每張牌以2個字元代表。第一個字元代表牌的點數（A=Ace, 2~9, T=10, J=Jack, Q=Queen, K=King），第二個字元代表牌的花色（C=Clubs, D=Diamonds, H=Hearts, S=Spades）

若遇到僅含#的一列代表輸入結束。請參考Sample Input。

**Output**

對每組測試資料輸出遊戲結束時剩下幾堆牌，以及每堆牌有多少張牌。請注意如果只有1堆牌，pile後沒有加s，請參考Sample Output。

**Sample Input**

QD AD 8H 5S 3H 5H TC 4D JH KS 6H 8S JS AC AS 8D 2H QS TS 3S AH 4H TH TD 3C 6S  
8C 7D 4C 4S 7S 9H 7C 5D 2S KD 2D QH JD 6D 9D JC 2C KH 3D QC 6C 9S KC 7H 9C 5C  
AC 2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC AD 2D 3D 4D 5D 6D 7D 8D TD 9D JD QD KD  
AH 2H 3H 4H 5H 6H 7H 8H 9H KH 6S QH TH AS 2S 3S 4S 5S JH 7S 8S 9S TS JS QS KS  
#

**Sample Output**

6 piles remaining: 40 8 1 1 1 1   
1 pile remaining: 52

**解法：**

使用Vector處理牌堆直到無法再進行移動為止

**解法範例：**

1. 用struct定義卡牌(可跳過)
2. 建立二維vector儲存所有卡牌，代表一次遊戲中的數堆牌堆以及一張牌堆中的數張牌
3. 以迴圈將輸入的26張牌push\_back於26堆牌堆之中
4. 如果輸入為#則程式結束
5. 從第二個牌堆(=第二張牌)開始處理，先後檢驗是否可向左三移動或左一移動，皆否則判斷下一堆牌堆
6. 若可移動時則於移動處push\_back欲移動的牌，並將原處的牌堆pop\_back
7. 若原牌堆pop\_back後為empty則erase該牌堆
8. 移動後重新回到第二個牌堆進行檢驗
9. 若持續到最後一個牌堆皆無法進行移動則跳出迴圈
10. 利用size()找出總共的牌堆數以及各牌堆的牌數

**討論：**

* Vector的push\_back(),pop\_back(),erase()等函數應用
* 先從左邊的牌進行檢驗，並先檢查移動左三再檢查移動左一
* 可以改使用stack但不建議使用array
* 輸出結果需檢查單複數

**程式：**

#include<iostream>

#include<string>

#include<vector>

#include<cstdlib>

using namespace std;

//建立struct

struct Card{

char suit;

int value;

};

int main(){

int i;

string input;

Card newCard;

//建立二維Vector

vector<vector<Card> > cards;

for(;;){

//初始化

cards.resize(52);

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

cards[i].clear();

}

//處理輸入

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

cin>>input;

if(input=="#")break;

newCard.suit=input[1];

if(input[0]=='A')newCard.value=1;

else if(input[0]=='T')newCard.value=10;

else if(input[0]=='J')newCard.value=11;

else if(input[0]=='Q')newCard.value=12;

else if(input[0]=='K')newCard.value=13;

else newCard.value=input[0]-'0';

cards[i].push\_back(newCard);

}

if(input=="#")break;

//從第二個牌堆進行處理，至最後一個牌堆時仍無法移動則跳出

for(i=1;i<cards.size();){

//檢查左邊第三張牌

if(i>=3 && ((cards[i].back().value == cards[i-3].back().value)||(cards[i].back().suit == cards[i-3].back().suit))){

//將牌放至牌堆最後一張

cards[i-3].push\_back(cards[i].back());

//移除原牌堆最後一張

cards[i].pop\_back();

//若變成空牌堆則消除

if(cards[i].empty())cards.erase(cards.begin()+i);

//回到第二個牌堆

i=1;

}

//檢查左邊第一張牌

else if(i>=1 && ((cards[i].back().value == cards[i-1].back().value)||(cards[i].back().suit == cards[i-1].back().suit))){

cards[i-1].push\_back(cards[i].back());

cards[i].pop\_back();

if(cards[i].empty())cards.erase(cards.begin()+i);

i=1;

}

//無法移動則檢查下一個牌堆

else i++;

}

//總牌堆數

cout<<cards.size();

//處理單複數

if(cards.size()==1)cout<<" pile remaining: ";

else cout<<" piles remaining: ";

//每牌堆張數

for(i=0;i<cards.size();i++){

cout<<cards[i].size();

if(i==cards.size()-1)cout<<endl;

else cout<<" ";

}

}

}