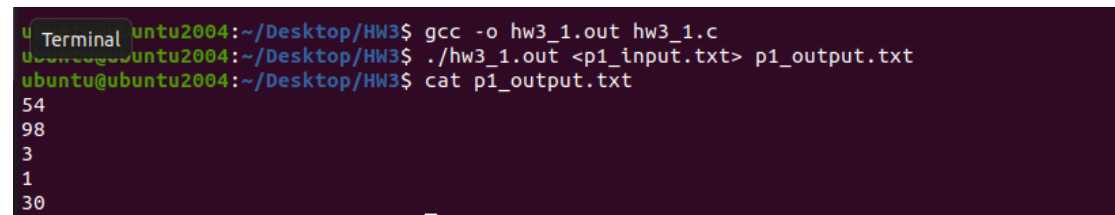


HW3_1

Result screenshot



```
Terminal ubuntu2004:~/Desktop/HW3$ gcc -o hw3_1.out hw3_1.c
ubuntu2004:~/Desktop/HW3$ ./hw3_1.out <p1_input.txt> p1_output.txt
ubuntu2004:~/Desktop/HW3$ cat p1_output.txt
54
98
3
1
30
```

Program Architecture

先用函式讀入資料，將盤子建立成堆疊，並在 **enqueue** 的時候將 **POP** 的盤子的索引值分別建立出 **A B** 兩個佇列，並按照題目規則，根據四個不同的指令去執行。詳細的方式會在下面的部份說明

Program function

1. 自定義函式: `int power(int a,int b)`

這個自定義函式用來處理 **PUSH** 指令後面的 **random index** 的，將字元轉換成 **int** 型態去儲存的時候，由於不同位數，相當於該位數的數字乘上 **10** 的不同次方，為了縮短程式碼並簡化，使其能寫在同一個 **for** 回圈內就解決，所以自定義了這個 **function**。**a** 代表底數，**b** 代表指數，以下是這個副程式的內容：

```
int power(int a,int b)
{
    int ans=1;
    if(b<1)
    {
        return ans;
    }else
    {
        for(int i=1;i<=b;i++)
        {
            ans*=a;
        }
        return ans;
    }
}
```

2. `Platpush(int index)`

這個函式是在 PUSH 指令的時候，用 linked list 實作 stack，將盤子索引值建立出一個堆疊

3. Platepop()

這是在 POP 指令的時候，用串列走訪的方式，取出最上方的盤子索引值的函式

4.Aenqueue(int num) Benqueue(int num)

這是把剛剛的 Platepop()取出的盤子值以鏈結串列實作 queue 的方式，丟進 A B 裡面並建立 queue 的函式

5.Adequeue() Bdequeue()

這是將 A B 裡面的開頭盤子取出並印出的函式

2.系統函式

```
1. char *strchr(const char *str, int c)
```

這個函式我為了處理 Unix 與 windows 與 MacOS 三種不同的 OS 底下，換行符號的不同而使用的函式，先傳入某個字元陣列的開頭位址，然後再寫入我需要搜尋的目標字元，這個函式就可以幫我找到字串中的目標字元的位址，並回傳那個位址給我，我就可以對目標字元進行修改或處理。

how you design your program

這次作業與上次不同的是，這次要使用 linked list 去模擬 stack 和 queue，我根據我自己買的一本屬圖解資料結構(使用 C 語言)裡面教的，將許多功能，比如 POP PUSH DEQUEUE ENQUEUE 都寫成副程式，這樣到時候 main 函式裡面就只需要直接呼叫，非常方便。這次使用到了串列走訪，front 與 rear 指標等。因為堆疊是先進後出，所以只要管開頭那個節點的問題就好，而佇列是先進先出，所以必須設兩個指標去做頭尾的管理。

HW3_2

Result screenshot

```

ubuntu@ubuntu2004:~/Desktop/HW3$ gcc -o hw3_2.out hw3_2.c
ubuntu@ubuntu2004:~/Desktop/HW3$ ./hw3_2.out <p2_input.txt> p2_output.txt
ubuntu@ubuntu2004:~/Desktop/HW3$ cat p2_output.txt
K 3 5 9 A 10 2 8 4 Q 6 7 J
3 5 9 A 10 2 8 4 Q 6 7 J
5 9 A 10 2 8 4 Q 6 7 J 3
9 A 10 2 8 4 Q 6 7 J 3 5
A 10 2 8 4 Q 6 7 J 3 5 9
10 2 8 4 Q 6 7 J 3 5 9 A
2 8 4 Q 6 7 J 3 5 9 A 10
8 4 Q 6 7 J 3 5 9 A 10 2
4 Q 6 7 J 3 5 9 A 10 2 8
Q 6 7 J 3 5 9 A 10 2 8 4
6 7 J 3 5 9 A 10 2 8 4
7 J 3 5 9 A 10 2 8 4 6
J 3 5 9 A 10 2 8 4 6 7
3 5 9 A 10 2 8 4 6 7
5 9 A 10 2 8 4 6 7 3
9 A 10 2 8 4 6 7 3 5
A 10 2 8 4 6 7 3 5 9
10 2 8 4 6 7 3 5 9 A
2 8 4 6 7 3 5 9 A
8 4 6 7 3 5 9 A 2
4 6 7 3 5 9 A 2 8
6 7 3 5 9 A 2 8 4
7 3 5 9 A 2 8 4 6
3 5 9 A 2 8 4 6 7
5 9 A 2 8 4 6 7 3
9 A 2 8 4 6 7 3 5
A 2 8 4 6 7 3 5
2 8 4 6 7 3 5 A
8 4 6 7 3 5 A 2
4 6 7 3 5 A 2
6 7 3 5 A 2 4
7 3 5 A 2 4 6
3 5 A 2 4 6

```

```

1 3 5 A 2 4 6
3 5 A 2 4 6
5 A 2 4 6 3
A 2 4 6 3 5
2 4 6 3 5 A
4 6 3 5 A 2
6 3 5 A 2 4
3 5 A 2 4
5 A 2 4 3
A 2 4 3
2 4 3 A
4 3 A 2
3 A 2
A 2
2 A
A

```

Program architecture

這次是有 13 張牌，分別含有 A~K 13 種大小，我先將所有牌都讀入，建立成一個 queue，再根據題目規則去判斷 front 指向的第一個數字是否為我要取的目標數字，如果是就將其取出，不是的話就將其丟到 queue 尾端。

Program function

自定義函式:

Enqueue(int num)

這是用來一開始建立撲克牌 queue，與將不為 draw out 目標的牌丟到 queue 尾端的函式。

Dequeue()

這是用來將如果第一張牌就是目標的牌，則將其 draw out 出來的函式

Int show()

這是利用串列走訪，在每次檢查時均將整個 **queue** 印出來的函式

Int check(int num,int mover)

這是用來比對 **queue** 第一張牌是否為當前的目標的函式，若是的話就回傳

1 不是的話就回傳 0

2.系統函式

```
1. char *strchr(const char *str, int c)
```

這個函式我為了處理 **Unix** 與 **windows** 與 **MacOs** 三種不同的 **OS** 底下，換行符號的不同而使用的函式，先傳入某個字元陣列的開頭位址，然後再寫入我需要搜尋的目標字元，這個函式就可以幫我找到字串中的目標字元的位址，並回傳那個位址給我，我就可以對目標字元進行修改或處理。

How you design your program

第一次看到題目的時候，由於有看到第一張牌丟到最尾端的這個說明，我一開始以為是環狀串列的實作，但是後來發現，我可以利用第一題的 **enqueue** **dequeue** 來完成，並且只需要單向串列就足以實作。我只要利用 **check** 函式先去判斷是否第一張牌就是我的目標，如果是的話，直接呼叫 **dequeue** 函式，把這張牌從 **queue** 中去除就好，如果第一張牌不是我要的目標，則先將這張牌利用 **enqueue** 丟入 **queue** 末尾(此時 **queue** 的頭跟尾都是同一張牌)，接著使用 **dequeue()** 將 **front** 指標指向的那張牌丟棄，這樣就完成了將牌丟到末尾去重新排隊的動作。這次作業也是利用 **linked list** 來實作 **queue**，並且在 **show()** 裡面也用到了串列走訪的動作。