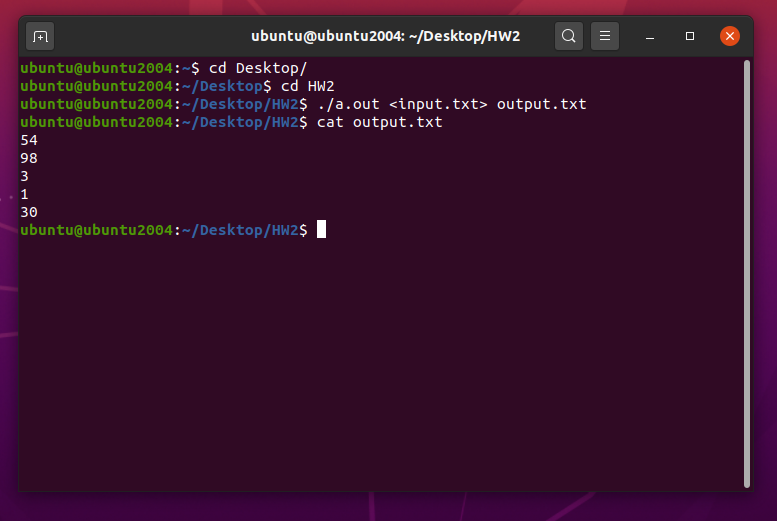
Result screenshot:



Program architecture:

先將每行指令逐一讀入，再根據不同的指令，使用if else判斷去個別處理，最終將答案寫入output.txt內，就完成了(詳細說明在how you design your program與functions)

我是用switch與if else搭配使用來做事的。這次的開檔不同於以往，這次是用bash redirection的方式做的。

Program functions:

1. 自定義函式:int power(int a,int b)

這個自定義函式式用來處理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. int sprintf(char \*str, const char \*format, ...)

這個函式是我在將int型態的數字轉為string的時候用的，因為必須先將數字轉為char或是char array，才能用fputs或是fputc寫入output.txt裡面，我原本打算自己寫一個副程式來處理int轉string，可是上網查詢後發現有這個好用的函式可以直接幫我處理。我可以先設一個char型態的陣列c，然後這樣做:

sprintf(c,”%d”,需要被我轉換成字串的某個int型態的整數)，就可以直接將該整數直接轉成字串型態並存入字元陣列c裡面。

3. char \*strchr(const char \*str, int c)

這個函式式我為了處理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。

4.how you design your program

這次的作業是要我們模擬stack與queue，觀察作業說明可以發現，那疊盤子填充的順序與客人拿取的順序是相反的，比如盤子填充的順序是30 98 54，但客人拿取盤子的順序是54 98 30，這就是堆疊的概念，先進後出，後進先出，所以我在這個部分，我使用一個plate陣列去存放盤子的編號，但是我採用的方式不是按照盤子進來的先後順序，從陣列第0格、第一格這樣排，我是將每次PUSH的盤子編號都往陣列的最後方丟，然後每次POP再從陣列最前方開始取，這樣就可以模擬stack的感覺。

我一開始初始化的時候就將每格plate陣列的值設為0，有新的盤子進入該格，再更改該格的值，這邊有一個點就是，若盤子被POP走了，必須將該格重新設為0。我還有設一個plate1變數去存目前plate陣列中盤子堆疊的高度，原始值為10000，每PUSH一個盤子就-1，比如目前PUSH了三個盤子進來，則plate1值為9997，代表目前自陣列第9997格開始有盤子，若盤子被POP走了，則該格歸0並將plate1值+1，比如剛剛的情況，若POP一個盤子，則plate1值為9998，代表陣列自第9998格開始才有盤子。

接下來我發現一件事，每次是A還是B去POP，是由下一行，哪條線ENQUEUE去決定的，所以我在while裡面，每次讀到那行，只要是POP，就continue，直到遇到ENQUEUE再去處理實際上是POP的動作。