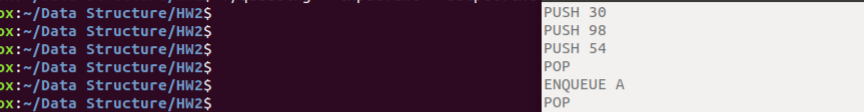
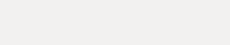


1. Result screenshot

```
pd2@VirtualBox:~/Data Structure/HW2$ gcc -o queueing queueing.c
pd2@VirtualBox:~/Data Structure/HW2$ ./queueing < input.txt > output.txt
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
pd2@VirtualBox:~/Data Structure/HW2$
```

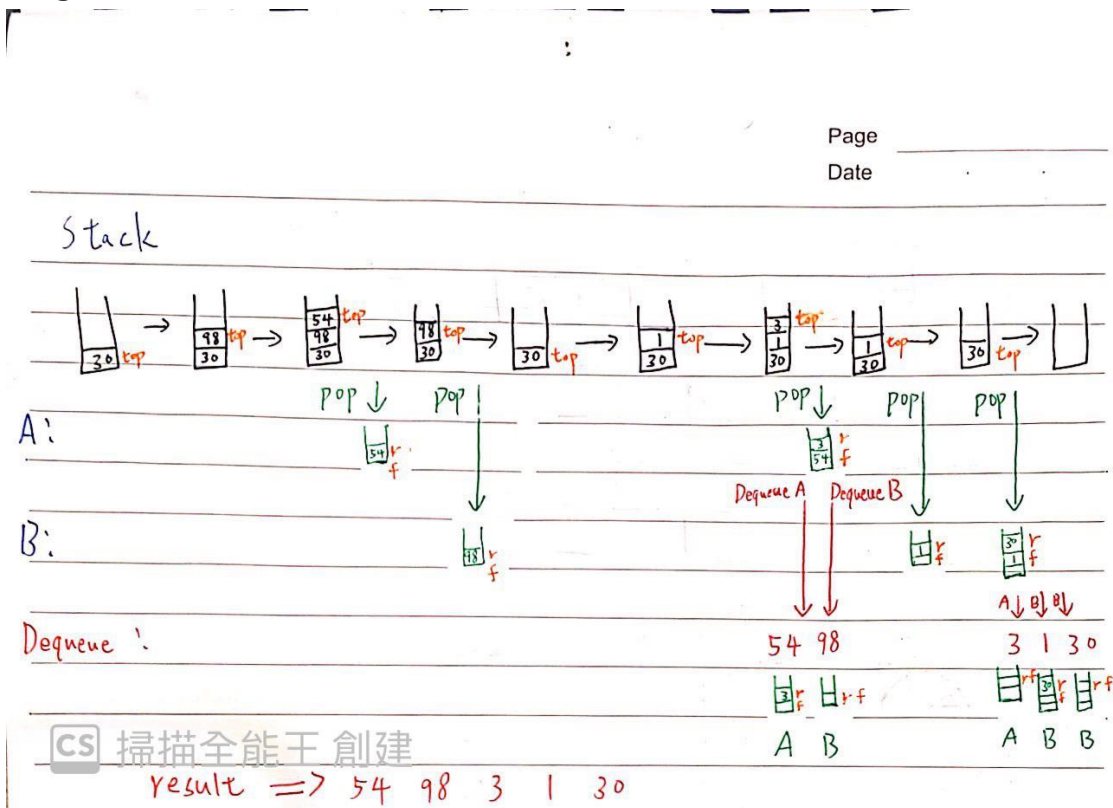


```
Open ▾ ⓘ
input.txt
~/Data Structure/HW2
PUSH 30
PUSH 98
PUSH 54
POP
ENQUEUE A
POP
ENQUEUE B
PUSH 1
PUSH 3
POP
ENQUEUE A
DEQUEUE A
DEQUEUE B
POP
ENQUEUE B
POP
ENQUEUE B
DEQUEUE A
DEQUEUE B
DEQUEUE B
DEQUEUE B
^Z (EOF)
```



```
Open ▾ ⓘ
output.txt
~/Data Structure/HW2
54
98
3
1
30
```

2. Program architecture



3. Program functions

(1)

strcmp():用來比對字串

聲明

以下是聲明的strcmp() 函數。

```
int strcmp(const char *str1, const char *str2)
```

參數

- str1 -- 這是第一個要比較的字符串。
- str2 -- 這是第二個的字符串進行比較。

返回值

這個函數的返回值如下：

- 如果返回值<0，則表明str1小於str2
- 如果返回值，如果> 0，則表明str2 小於 str1
- 如果返回值= 0，則表明str1 等於str2

(2)

void add_plate(int top, int plate):用來把 plate 放入 stack 中

top – stack 的頂端

plate – 要放入的 plate 的號碼

(3)

int delete_plate(int top):用來取出 stack 中的 plate

top – stack 的頂端

return – 取出的 plate 的號碼

(4)

void enqueue(int queue[], int rear, int plate):把客人列入隊伍

queue[] – 要放入的隊伍(queue_A 或 queue_B)

rear – queue 的頂端

plate – 客人手上拿的 plate 的號碼

(5)

int dequeue(int queue[], int front):用來取得離開隊伍的客人的 plate 的號碼

queue[] – 要離開的隊伍(queue_A 或 queue_B)

front – queue 的底端

return – 客人放回的 plate 的號碼

4. How I design my program

設陣列 `plate_stack` 來創立一個 `stack`，並設立一個 `top_plate` 來標示 `plate_stack` 的頂端；設陣列 `queue_A`、`queue_B` 來建立兩個 `queue`，並設立 `rear_A`、`rear_B` 來標示這兩個 `queue` 的頂端，`front_A`、`front_B` 則標示這兩個 `queue` 的底端。

設一個陣列 `operation` 來接從 `input.txt` 接收到的字串，用 `strcmp` 加上 `if`、`else if`，用類似 `switch case` 的方式來判斷現在是要 `PUSH`、`POP`、`ENQUEUE`、或是 `DEQUEUE`。

若接收到 `PUSH`，則設一個 `int` 變數 `N` 來接收後面的號碼，也就是 `plate` 的號碼，並且呼叫 `add_plate()` 這個函式，將 `N` 放入 `plate_stack` 中，並將 `top_plate+1`。

若接收到 `POP` 則直接呼叫 `continue`，因為 `POP` 後面是接 `ENQUEUE A` 或 `ENQUEUE B`，也就是將 `plate_stack` 中的 `N` `pop` 進隊伍中，而這一步我直接放到 `ENQUEUE` 的地方做，因此 `POP` 這邊我直接 `continue`。

若接收到 `ENQUEUE`，先設立一個陣列 `AorB`，並把 `ENQUEUE` 後面的字串放入其中，也就是 `A` 或是 `B`，並一樣透過 `strcmp()` 來判斷是 `A` 或 `B`，接著 `assign rear_A` 或 `rear_B` 為其與 `MAX_SIZE` 的餘數+1，這邊的用意是為了節省 `queue` 的空間(講義上圓圈的作法)，然後呼叫 `enqueue()` 來將 `plate_stack` 中位於 `top_plate` 的 `N` 放入 `queue_A` 或 `queue_B` 中，並且將 `top_plate-1`。

若接收到 `DEQUEUE`，做法和 `ENQUEUE` 一樣，先判斷 `A` 或 `B`，然後將 `assign front_A` 或 `front_B` 為其與 `MAX_SIZE` 的餘數+1，然後呼叫 `dequeue()`，將返回的結果直接輸出在 `output.txt` 中，並且將 `top_plate-1`。