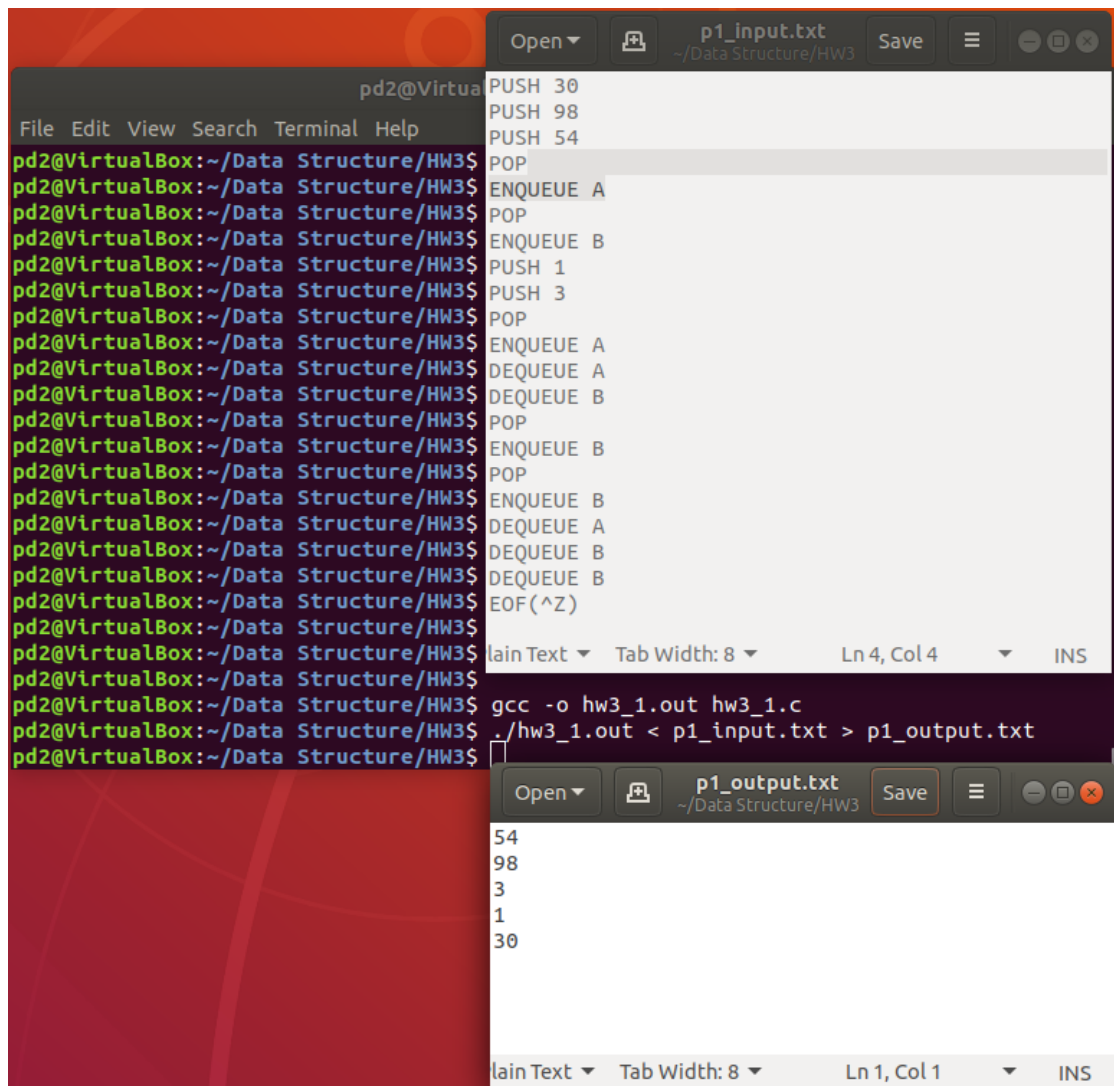


HW3_1

1. Result screenshot



The screenshot shows a terminal window with a menu bar (File, Edit, View, Search, Terminal, Help) and a title bar (pd2@VirtualBox). The terminal displays a series of commands and their outputs for a queue implementation. The commands are: 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, EOF(^Z). The output shows the state of the queue after each operation. The terminal also shows the execution of a C program (hw3_1.c) using gcc, which reads from p1_input.txt and writes to p1_output.txt. The output file p1_output.txt contains the numbers 54, 98, 3, 1, and 30.

```
pd2@VirtualBox:~/Data Structure/HW3$ PUSH 30
pd2@VirtualBox:~/Data Structure/HW3$ PUSH 98
pd2@VirtualBox:~/Data Structure/HW3$ PUSH 54
pd2@VirtualBox:~/Data Structure/HW3$ POP
pd2@VirtualBox:~/Data Structure/HW3$ ENQUEUE A
pd2@VirtualBox:~/Data Structure/HW3$ POP
pd2@VirtualBox:~/Data Structure/HW3$ ENQUEUE B
pd2@VirtualBox:~/Data Structure/HW3$ PUSH 1
pd2@VirtualBox:~/Data Structure/HW3$ PUSH 3
pd2@VirtualBox:~/Data Structure/HW3$ POP
pd2@VirtualBox:~/Data Structure/HW3$ ENQUEUE A
pd2@VirtualBox:~/Data Structure/HW3$ DEQUEUE A
pd2@VirtualBox:~/Data Structure/HW3$ DEQUEUE B
pd2@VirtualBox:~/Data Structure/HW3$ POP
pd2@VirtualBox:~/Data Structure/HW3$ ENQUEUE B
pd2@VirtualBox:~/Data Structure/HW3$ POP
pd2@VirtualBox:~/Data Structure/HW3$ ENQUEUE B
pd2@VirtualBox:~/Data Structure/HW3$ DEQUEUE A
pd2@VirtualBox:~/Data Structure/HW3$ DEQUEUE B
pd2@VirtualBox:~/Data Structure/HW3$ DEQUEUE B
pd2@VirtualBox:~/Data Structure/HW3$ EOF(^Z)
pd2@VirtualBox:~/Data Structure/HW3$ gcc -o hw3_1.out hw3_1.c
pd2@VirtualBox:~/Data Structure/HW3$ ./hw3_1.out < p1_input.txt > p1_output.txt
pd2@VirtualBox:~/Data Structure/HW3$
```

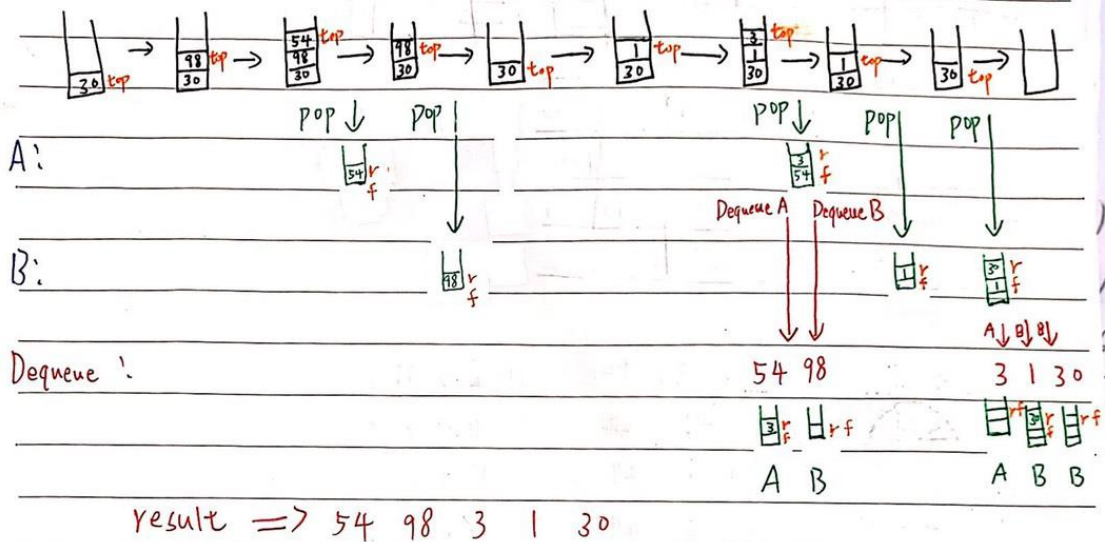
54
98
3
1
30

2. Program architecture

Page _____

Date _____

Stack



用 linked list 实现:

stack.

ex: push 30

push 98

push 54

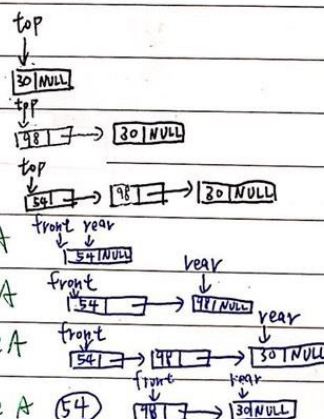
queue

ex: enqueue A

enqueue A

enqueue A

dequeue A



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(stack_pointer *top, int plate):把 plate 放入 stack 中

top – 指向 stack 中最上面那一筆資料的 pointer

plate – 要放入的 plate 的號碼

(3)

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

top – 指向 stack 中最上面那一筆資料的 pointer

return – 取出的 plate 的號碼

(4)

void enqueue(queue_pointer *front, queue_pointer *rear, int data):

把客人列入隊伍

*front – 指向該隊伍(A 或 B)第一個進來(隊伍最前面)那一筆資料的 pointer

*rear – 指向該隊伍(A 或 B)最後一個進來(隊伍最前面)那一筆資料的 pointer

data – 客人手上拿的 plate 的號碼

(5)

int dequeue(queue_pointer *front):用來取得離開隊伍的客人的 plate 的號碼

*front – 指向該隊伍(A 或 B)第一個進來(隊伍最前面)那一筆資料的 pointer

return – 客人放回的 plate 的號碼

4. How I design my program

利用 `typedef` 和 `struct` 來建立 `stack_pointe`、`queue_pointer`，來建立能夠存放資料和指向下一筆資料的 `pointer`。

設立 `stack_pointer` 陣列 `top_plate` 來建立一個 `stack`，設立 `queue_pointer` 陣列 `front_A`、`front_B`、`rear_A`、`rear_B`，

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，宣告一個新節點 `temp`，把 `N` 輸入到 `temp` 的 `data` 中(`temp->data = plate`)，之後再透過 `add_plate` 中的操作，把 `top` 指向 `temp`。

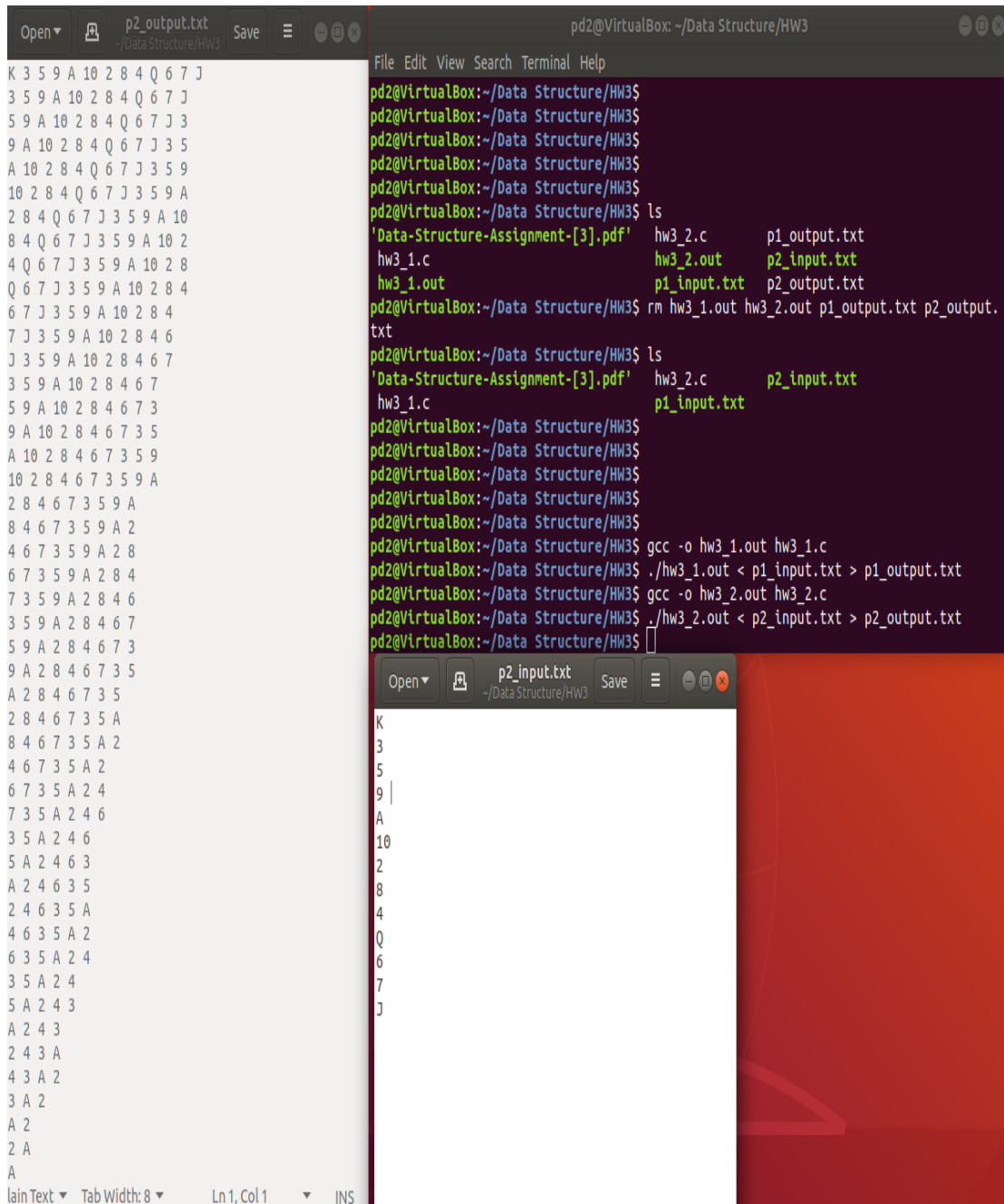
若接收到 `POP` 則直接呼叫 `continue`，因為 `POP` 後面是接 `ENQUEUE A` 或 `ENQUEUE B`，也就是將 `stack` 中的 `N` `pop` 進隊伍中，而這一步我直接放到 `ENQUEUE` 的地方做，因此 `POP` 這邊我直接 `continue`。

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`，接著呼叫 `enqueue()` 來將 `stack` 中位於 `top` 所指向的 `node` 中的 `data` (也就是 `N`) 用來建立一個新的節點，然後把 `top` 指向 `top->next`，並把原本 `top` 的地方 `free` 掉，以免造成 `memory leak`，然後透過 `enqueue()` 中的操作，新的節點接在 `rear` 所指向的地方，並把 `front` 跟 `rear` 這兩個 `pointer` 指向該指的地方(`front` 指在第一個進來的資料、`rear` 指在最後一個進來的資料)。

若接收到 `DEQUEUE`，做法和 `ENQUEUE` 一樣，先判斷 `A` 或 `B`，然後呼叫 `dequeue()`，將返回的結果直接輸出在 `output.txt` 中，並且將透過 `dequeue()` 中的操作，使 `front` 指向 `front->next`，並將原本 `front` 指向的那個節點 `free` 掉，以免造成 `memory leak`。

HW3_2

1. Result screenshot



```
pd2@VirtualBox: ~/Data Structure/HW3
File Edit View Search Terminal Help

pd2@VirtualBox:~/Data Structure/HW3$ ls
'Data-Structure-Assignment-[3].pdf' hw3_2.c p1_output.txt
hw3_1.c hw3_2.out p2_input.txt
hw3_1.out p1_input.txt p2_output.txt
pd2@VirtualBox:~/Data Structure/HW3$ rm hw3_1.out hw3_2.out p1_output.txt p2_output.txt
pd2@VirtualBox:~/Data Structure/HW3$ ls
'Data-Structure-Assignment-[3].pdf' hw3_2.c p2_input.txt
hw3_1.c p1_input.txt
pd2@VirtualBox:~/Data Structure/HW3$
pd2@VirtualBox:~/Data Structure/HW3$
pd2@VirtualBox:~/Data Structure/HW3$
pd2@VirtualBox:~/Data Structure/HW3$
pd2@VirtualBox:~/Data Structure/HW3$ gcc -o hw3_1.out hw3_1.c
pd2@VirtualBox:~/Data Structure/HW3$ ./hw3_1.out < p1_input.txt > p1_output.txt
pd2@VirtualBox:~/Data Structure/HW3$ gcc -o hw3_2.out hw3_2.c
pd2@VirtualBox:~/Data Structure/HW3$ ./hw3_2.out < p2_input.txt > p2_output.txt
pd2@VirtualBox:~/Data Structure/HW3$
```

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
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
```

p2_input.txt

```
K
3
5
9 |
A
10
2
8
4
Q
6
7
J
```

2. Program architecture

int output_num = 13

Page

Date

node:

card	number
------	--------

 → ex:

K	13	NULL
---	----	------

利用 node → card 來判斷, 並串成一個 linked list.

head

K	13
---	----

 →

3	13
---	----

 → ... →

J	11	NULL
---	----	------

when (head != NULL) ⇒ 還有沒有抽出的牌.

⇒ 先印出全部

⇒ if (head → number == output_num)

抽出此牌且 output_num - 1

並且把 head 指向 head → next, 把原本 head 的空間 free 掉

⇒ else

建立新 node: current, 把 head 的資料都傳給 current

把 current 接在 linked list 的最後面

並且把 head 指向 head → next, 把原本 head 的空間 free 掉

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) strcpy():用來複製字串

聲明

以下是聲明的strcpy() 函數。

```
char *strcpy(char *dest, const char *src)
```

參數

- dest -- 這就是指針的內容將被複製到目標數組。
- src -- 這是要複製的字符串。

返回值

這返回一個指向目標字符串dest

4. How I design my program

首先利用 typedef 和 struct 建立節點，一個節點包含一組字串(card) 一個數字(number)、和指向下一節點的 pointer(next)。

再來先宣告一個 integer output_num=13，用來對之後判斷是否要抽出此牌做準備，並宣告兩個 solitaire_pointer，head 和 current，head 永遠指向第一個節點，而 current 則是現在要用來判斷讀進來的是哪一張卡片。

利用 strcmp()將傳進來的字母(K、Q、J、A)或是數字(2~10)建立成一個一個的節點，利用 strcpy()將其字母傳入 node->card，而代表的數字則直接傳入 node->number(例如 K，則 strcpy(current->card, "K")、current->number = 13)，而要注意的是，當 head 還是空的時候，則

head=current，其他則透過 while 迴圈將 current 放入 linked list 的最後面。

再來用一個 do while 迴圈，先把所有節點中的字串 card 印出來，然後開始判斷，利用前面的 output_num，如果 head 指向的資料的數字 number == output_num，則把 head 指向 head->next，並且把原本 head 的位置 free 掉，以免造成 memory leak，如果 number != output_num，則建立一個新節點 current，把 head 中的 card、number 都傳給他，然後接在 linked list 的最後面，之後把 head 指向 head->next，再把 head 原本的位置 free 掉，以免造成 memory leak。