Algorithm explanation

- Allocate two pieces of space foe the deque, and set two pointer `LP` & `RP` in the middle.
  - At first, `LP = RP`, representing the deque is empty.
- And the pointer acts like this:
  - `RP` points to the next location of the elements need to be push in.
  - `LP` points to the location of the elements that is next to the left end of the queue.
  - That is, `LP` points to a real elements in the deque (if it isn't empty), while `RP` points to the space where the next elements pushed from right will be put.
- Then, when I get a character from the console, I will prove it is an option rather than an operand. Then I will check which end of the queue will be operated.
- After that, I will check whether it is a `push` option or a `pop` option. And achieve it separately.
  - If it's a `pop` option, I will store the result into a special memory space to store the output message, which I called "output cache" in my code.
- In my cods, process about differentiating operations above is described as "routing".
- In the end, I will simply output all the message in the "output cache".

Essential parts of your code with sufficient comments

```
            .ORIG   x3000
            LEA     R4, DSTACK_R
            LEA     R5, DSTACK_R
            LD      R6, STACK_ADD

            JSR     incSP           ; Keep Safe.
RD_OPT      GETC                    ; Read in
            OUT                     ; Show on Screen
            ADD     R1, R0, x-A    ; if '\n'
            BRz     SHOW_RES        ; Meet '\n'
            STR     R0, R6, #-1     ; Keep Safe.
            BRnzp   ROUTING_OPT     ; Routing the options.

SHOW_RES    LD      R1, OUTPUT_ADD  ; Find the output string's location.
PRINT       ADD     R1, R1, #1      ; Move
            LDR     R0, R1, #0      ; Load ASCII
            BRz     EXIT
            OUT
            BRnzp   PRINT

EXIT        HALT
; And something else.
```

Questions TA asked you and your answer in Check

- Q: How to achieve this using list?

- A: A node should have a data memory and a pre-pointer and a next-pointer, the two pointers should point to the previous node and the next node. And the stack pointer movement should be defined like point to the memory pointed by the next-pointer or the pre-pointer.