**Algorithm explanation:**

We are supposed to design a flexible data structure to load and store the data in both left and right, therefore we can use two point to locate the bottom and head of the structure. To avoid repeat store, when we add to the left, we store data first and then move the point, on the other hand we move the point first and then store the data. For the right side, the opposite is true. After every input, we store the corresponding output to the output stack, in the end, we output it as a whole, then we stop the program.

But there also exists some bugs in this program, when we tick the wrong key and we use the backspace to delete it, but it will confuse the LC-3. Although I polish the program to deal with the bug, it only can deal with a single backspace, because of the characteristic of the certain algorithm, this cannot be avoided.

**Essential parts of your code with sufficient comments:**

......

test trap x20 ;display the input of operate

trap x21

ld r2,enter ;load the minis hex number of enter

add r2,r0,r2 ;if input equals to enter, jump to output mode

brz output

ld r2,minis

add r2,r0,r2

brz opemi ;if input equals to minis, jump to minis mode

ld r2,plus

add r2,r0,r2

brz opepl ;if input equals to plus, jump to plus mode

ld r2,left

add r2,r0,r2

brz opele ;if input equals to left brackets, branch to left brackets mode

ld r2,right

add r2,r0,r2

brz operi ;if input equals to right brackets, branch to right brackets mode

brnzp test ;other input, no processing, continue the loop

.......

opemi and r4,r4, #0 ;judge the location fo r5 and r6, if r5 equals to r6, the structure is empty

not r4,r5

add r4,r4, #1

add r4,r6,r4

brz addmr ;if the structure is empty, branch to the add-to-output mode

brnzp admi ;if the structure is not empty, branch to the other output mode

opepl trap x20 ;display the character be inputed

trap x21

add r2,r0, #0 ;load r0 to r2 to add to the output stack

brnzp adpl

......

output ld r2,mark ;at the end of output stack, add x0000 to stop the trap x22

str r2,r7, #0

ld r0,ouput ;output the output stack

trap x22

trap x25 ;halt

addmr ld r2,unline ;output the '\_'

addmr2 str r2,r7, #0 ;add it to output stack, move r7 to next location

add r7,r7, #1

brnzp test ;continue the loop

admi add r5,r5, #1 ;if we pop in the left, we first move the point, then load the data

ldr r2,r5, #0

brnzp addmr2 ;add the data into the output stack

adpl str r2,r5, #0 ;if we store in the left, we first store the data, then move the point

add r5,r5, #-1

brnzp test ;continue the loop

......

stack .fill x4000 ;store the start location of input stack

ouput .fill x5000 ;store the start location of output stack

enter .fill xfff6 ;store the minis of enter

minis .fill xffd3 ;store the minis of minis

plus .fill xffd5 ;store the minis of plus

left .fill xffa5 ;store the minis of left brackets

right .fill xffa3 ;store the minis of right brackets

unline .fill x005f ;store the minis of underline

......

**Questions TA asked you and your answer in Check:**

Question 1:

Please describe your algorithm.

Answer 1:

Similar to algorithm explanation mode, and add some necessary comments of the machine code. See above for details.

Question 2:

What is your opinion of the key --- backspace ?

Answer 2:

Because of the certain algorithm in my program, I deal with the output during the input, so I cannot perfect deal with the backspace, and I only can discard one wrong input each time. But if change the algorithm, input all operations and then deal it, we can avoid the harm of the backspace, but it requires larger space to store the input operation and it need more instruction to realize it.