Question 1:
Given two stacks S1 and S2, implement a queue that supports the following operations:

- enqueue(x): Add an element x to the rear of the queue
- dequeue(): Remove an element from the front of the queue

You are allowed to perform only stack operations push() and pop(). Array or linked list implementations can be used.

Question 2:
Given an infix expression, convert it into prefix notation. You can use standard stack operations for implementation. Testcases for evaluation are given below:

Testcase 1: infix expression: a + b * (c ^ d – e) ^ (f + g * h) - i

Prefix: +a -* b ^ ^ c d – e + f * g h i

Testcase 2: infix expression: (a / b + (c * d) + e)

Prefix: + + / a b * c d e

Question 3:

Implement a split function ***SLL_Split (head, key)*** that splits a linked list L (pointed to by ***head***) into two lists Ll and L2 such that,

- elements in Ll are less than a specified ***key***
- elements in L2 are greater than or equal to the specified ***key***

For example if L = {−5, 4, 3, 2, 9, 1, −47, 19} and key = 4, your aim is to create two lists as follows:
Ll = {−5, 3, 2, 1, −47}
L2 = {4, 9, 19}

Question 4:

Input two linked lists Ll and L2 pointed to by ***head1*** and ***head2*** respectively from the user. Merge these lists into a third linked list such that in the merged list, all even numbers occur first followed by odd numbers. A sample set of input and output is given below:
Testcase 1:
input list 1: 12 47 878 2 0 3
input list 2: 5 45 20 81 100 1008 87 25
The merged list is
12 878 2 0 20 100 1008 47 3 5 45 81 87 25

Question 5:

Create a data structure to represent a memory block, which should include:

- Size of the block
- Status (allocated or free)
- Pointer to the next block

Implement functions to simulate memory allocation using best-fit strategy on the following process requests:
Testcase 1: Memory blocks: 200 KB, 400 KB, 600 KB, 500 KB, 300 KB, 250 KB
Process requests: 357 KB, 210 KB, 468 KB, 491 KB
After performing allocation, print the memory blocks along with their allocation status.

Question 6:
A string of parentheses is said to be balanced if each opening parenthesis has a corresponding closing parenthesis and the pairs of parentheses are properly nested. (()()()()) and ((()())) are

examples of balanced strings of parentheses whereas ())) and (()()(() are not balanced. Implement a parenthesis checker using stack to check whether a given input string of parentheses is balanced or not.

Question 7:

Create a stack of integers. Input a value called ***threshold*** and remove all stack items that are greater than ***threshold***. The rest of the elements should be in the stack. You are permitted to perform only stack operations. You may use a second stack if required. Two sample sets of input and output are given below:
Testcase 1: 34 -1 9 5 6 -567 55 2 94
threshold 10
Output: stack after deleting elements -1 9 5 6 -567 2
Testcase 2: 44 3 -7 1 5 27
threshold 50
Output: Sorry, no elements greater than threshold! stack after deleting elements 44 3 -7 1 5 27

Question 8:

Implement a circular queue using linked list. The queue should support the following operations:

- enqueue(x): Add an element x to the rear of the queue
- dequeue(): Remove an element from the front of the queue
- display(): print the queue

Question 9:

Given two singly linked lists L1 and L2 pointed to by ***head1*** and ***head2*** respectively, implement a merge function ***SLL_Merge (head1, head2)*** that appends the smaller list (in terms of number of nodes) to the tail of larger list. If the lists are of same size, then append L2 to the tail of L1.

Question 10:

Implement a reversal function ***DLL_Reverse (head)*** that reverses a doubly linked list L (pointed to by ***head***). Print the reversed list.