Data Structures and Algorithms

Assignment 01: Linked List and Stacks & Queues

Program templates for Questions 1 to 4 are available on Hackerearth. You are required to use these templates to implement your functions. For your convenience, I have attached the templates for reference and practice on other IDEs. However, you must use the templates provided on Hackerearth and submit your solutions directly through the platform. An email invitation will be sent to your school account. Deadline for program submission: **September 21st, 2025 (Sunday) 11.59 pm.**

1. Write a function `moveMinNode()` that moves the node with the minimum value in a linked list to the front of the list. If there are multiple nodes with the minimum value, all of them should be moved to the front. Note that the `removeNode()` function for a linked list is not provided in the code template: you are required to write your own within the function if necessary. The function prototype is given below:

   ```
   def moveMinNode(head):
   ```

   If the linked list is **10 -> 20 -> 30 -> 40 -> 50 -> 1**, the resulting linked list after moving the node with the minimum value to the front is **1 -> 10 -> 20 -> 30 -> 40 -> 50**.

   If the linked list is **1 -> 2 -> 3 -> 1 -> 2 -> 3 -> 1**, the resulting linked list after moving all nodes with the minimum value to the front is **1 -> 1 -> 1 -> 2 -> 3 -> 2 -> 3.**

   Some sample input and its output:

   Example 01

   ```
   Enter a list of numbers, terminated by any non-digit character:
   10 20 30 40 50 1 a
   Before: 10 -> 20 -> 30 -> 40 -> 50 -> 1 ->
   After: 1 -> 10 -> 20 -> 30 -> 40 -> 50 ->
   ```

   Example 02

   ```
   Enter a list of numbers, terminated by any non-digit character: 1
   2 3 1 2 3 1 a
   Before: 1 -> 2 -> 3 -> 1 -> 2 -> 3 -> 1 ->
   After: 1 -> 1 -> 1 -> 2 -> 3 -> 2 -> 3 ->
   ```

2. Write a function `reverseDoublyList()` that reverses the nodes in a doubly linked list by swapping the `next` and `prev` pointers of each node. After reversing, the head of the doubly linked list should be updated to point to the new first node. The reversal should be performed without using any additional memory or creating new nodes. Instead, the existing nodes in the doubly linked list should be rearranged by modifying their pointers (`next` and `prev`), changing how they link to one another. This approach ensures that the reversal is memory-efficient and utilizes only the original nodes in the list.

   The function prototype is given below:

   ```
   def reverseDoublyList(head):
   ```

   If the doubly linked list is **10 20 30 40 50**, the resulting **reversed** doubly linked list is **50 40 30 20 10**.

   Some sample input and its output:

   Example 01

   ```
   Enter a list of numbers, terminated by any non-digit character: 10 20 30 40 50 a
   Before: 10  20  30  40  50
   After: 50  40  30  20  10
   ```

   Example 2

   ```
   Enter a list of numbers, terminated by any non-digit character: 1 2 3 4 5 a
   Before: 1  2  3  4  5
   After: 5  4  3  2  1
   ```

3. Write a `deleteMiddleElement()` function that removes the middle element from a given stack using only **one temporary stack**. The `deleteMiddleElement()` function should use only `push()` and `pop()` operations to add or remove integers from the stack. If the stack contains an odd number of elements, it should delete the exact middle element. If the stack contains an even number of elements, it should remove the lower of the two middle elements—the one closer to the bottom of the stack, which was pushed earlier.

   The function prototype is given below:

   ```
   def deleteMiddleElement(s):
   ```

   If the stack has an odd number of elements, such as **1 2 3 4 5**, the resulting stack after removing the middle element is **1 2 4 5**.

   If the stack has an even number of elements, such as **1 2 3 4 5 6 7 8**, it should remove the lower

of the two middle elements—the one closer to the bottom of the stack, which was pushed earlier. The resulting stack after removing this middle element is **1 2 3 4 6 7 8**.

Some sample input and its output:

Example 01

```
Enter a list of numbers, terminated by any non-digit character: 5
4 3 2 1 a

Before: 1 2 3 4 5

After: 1 2 4 5
```

Example 02

```
Enter a list of numbers, terminated by any non-digit character: 8
7 6 5 4 3 2 1 a

After: 1  2  3  4  6  7  8
```

4. Write an `interleaveQueue()` function that rearranges the elements of a given queue of integers with an even length by interleaving the first half of the queue with the second half, using only one stack. The `interleaveQueue()` function should use only `push()` and `pop()` operations to add or remove integers from the stack, and only `enqueue()` and `dequeue()` operations to add or remove integers from the queue.

   The function prototype is given below

   ```
   def interleaveQueue(q):
   ```

   If the queue is **1 2 3 4 5 6 7 8 9 10**, the resulting queue after interleaving the first half with the second half is **1 10 2 9 3 8 4 7 5 6**.

   If the queue is **10 20 30 40 50 60**, the resulting queue after interleaving the first half with the second half is **10 60 20 50 30 40**.

   Some sample input and its output:

   Example 01

   ```
   Enter a list of numbers, terminated by any non-digit character: 1
   2 3 4 5 6 7 8 9 10

   Before: 1  2  3  4  5  6  7  8  9  10

   After: 1  10  2  9  3  8  4  7  5  6
   ```

   Example 02

   ```
   Enter a list of numbers, terminated by any non-digit character:
   ```

```
10 20 30 40 50 60 a
Before: 10  20  30  40  50  60
After: 10  60  20  50  30  40
```