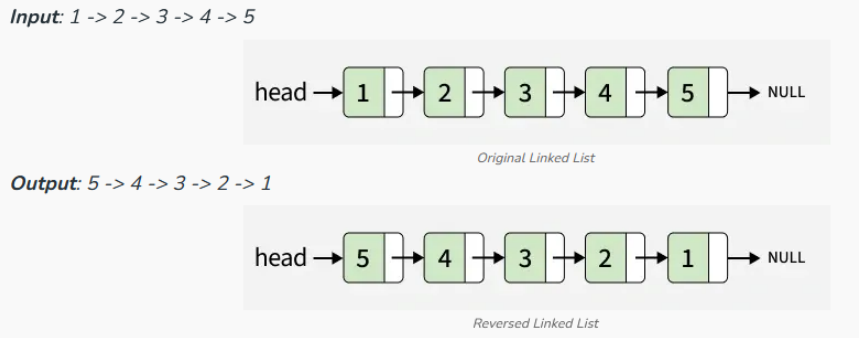
Computing Lab Assignment - In class task

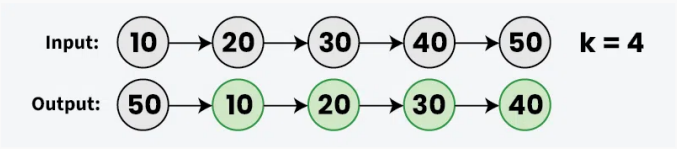
August 22, 2025

1. Given a singly linked list, find the middle node of the linked list.
2. Given a Linked List of M nodes and a number N, find the value at the N^{th} node from the end of the Linked List. If there is no N^{th} node from the end, print -1.
3. Given a singly linked list, reverse the linked list by changing the links between nodes.



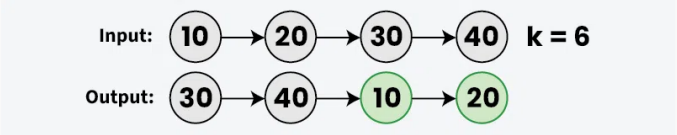
4. Given a singly linked list $L_0 \to L_1 \to \ldots L_{n-1} \to L_n$, rearrange the nodes in the list so that the newly formed list is: $L_0 \to L_n \to L_1 \to L_{n-1} \to L_2 \to L_{n-2} \to \ldots$. You are required to do this in place without altering the node values.
5. Given a singly linked list and an integer k, rotate the linked list to the left by k places.

Input: *linked list = 10 -> 20 -> 30 -> 40 -> 50, k = 4*
Output: *50 -> 10 -> 20 -> 30 -> 40*



Explanation: *After rotating the linked list to the left by 4 places, the **5th** node, i.e node **50** becomes the head of the linked list and next pointer of node **50** points to node **10**.*

Input: *linked list = 10 -> 20 -> 30 -> 40, k = 6*
Output: *30 -> 40 -> 10 -> 20*

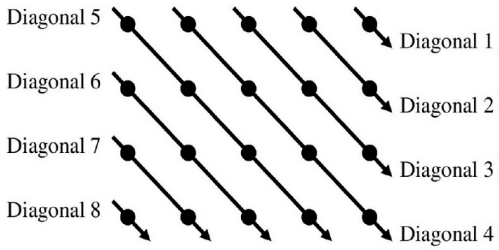


Explanation: *After rotating the linked list to the left by 6 places (same as rotating by 2 places as (k % len) = (6 % 4 = 2)) , the **3rd** node, i.e node **30** becomes the head of the linked list and next pointer of node **40** points to node **10**.*

6. Given a singly linked list, check if the given linked list is palindromic.

7. Given a linked list and a value x, partition it such that all nodes less than x come first, then all nodes with a value equal to x, and finally nodes with a value greater than x. The original relative order of the nodes in each of the three partitions should be preserved. [Related: Given a linked list of 0s, 1s and 2s, The task is to sort the list in non-decreasing order.]

8. Define a two-dimensional array A in the main() function of a C program. In the main() function, the user supplies the number r of rows and the number c of columns in A. The user then calls an initialize function to initialize the array with user inputs. Each row is allocated memory to store exactly c int variables.



Write a function to print the diagonals of A. A diagonal of A starts from the left or the top boundary, proceeds toward south-east, and ends at the right or the bottom boundary. Let us number the diagonals in the south-west direction starting from top right. As an example, the adjacent figure shows a 4×5 matrix. The elements are represented by black dots. Note that not all the diagonals contain the same number of elements.

9. HopSearch is an algorithm for searching in sorted arrays. Although slower than binary search, HopSearch is much faster than linear search. HopSearch uses a doubly nested loop, and employs a Hop strategy at exponentially increasing indices. Like binary search, Hop search maintains a search interval I = [L, R]. To start with, the interval I encompasses the entire array. Each iteration of the outer loop reduces the interval I to a strictly smaller sub-interval I' = [L', R'] determined as follows. It keeps on looking at the indices 1, 2, 4, 8, 16, … relative to L until an index is found, at which the array element is greater than or equal to the key, or the array size is exceeded. These comparisons help to choose the sub-interval I' to the correct one from [L, L], [L+1, L+1], [L+2, L+3], [L+4, L+7], [L+8, L+15], . . . , [L+2^t, R]. In the next iterations, the same search mechanism is continued, on gradually shrinking intervals. The loop stops when the interval length reduces to one. A final comparison of the key is made with the element at the only index of the interval, and depending on the outcome of this comparison, L (key found) or –1 (key not found) is returned. Code up HopSearch() as a single function. Write an application to call HopSearch with an array and get it sorted. Put the search routine and application in separate files. Assume that a sorted array is passed as input along with an element to search from the command line (you do not know the size of the array in advance).

10. Consider an array of n positive integers A. We plan to sort A in place, using an algorithm called *magicSort*. This algorithm repeatedly sorts the array A, based on a digit position starting from the least and going to the most significant ends of the numbers. As an example, take A = {415, 73, 516, 923, 890, 318}. In the first iteration, the array gets sorted based on the rightmost / least-significant digits, and changes to {89$\underline{0}$, 7$\underline{3}$, 92$\underline{3}$, 41$\underline{5}$, 51$\underline{6}$, 31$\underline{8}$}. The second iteration sorts A using the second / middle digits, changing A to {41$\underline{5}$, 51$\underline{6}$, 31$\underline{8}$, 92$\underline{3}$, 7$\underline{3}$, 89$\underline{0}$}. The third / last iteration looks at the the leftmost / most-significant digits, and changes A to {07$\underline{3}$, 31$\underline{8}$, 41$\underline{5}$, 51$\underline{6}$, 89$\underline{0}$, 92$\underline{3}$}. Not all elements of A are required to have the same number of digits (see the element 73 in the above example).