

***Name:*** *Zohaib Khan*

***Section:*** *Bssem-3A*

***ROLL NO:*** *Bssem-022*

***Assignment DSA Lab***

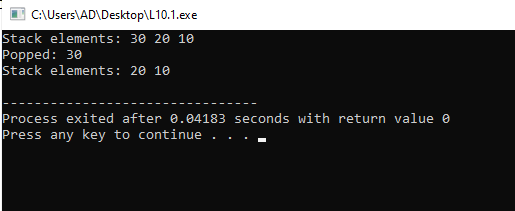
***SUBMITTED TO:***

***Sir Rasikh ALi***

***LAB TASK***

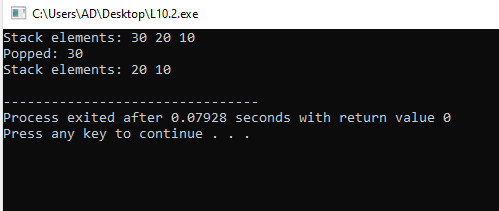
***TASK #10.1***

*This implementation defines a stack using a fixed-size array of 100 elements. The****push****method adds an element to the top of the stack, while the****pop****method removes the top element and displays it. The****display****method prints all elements in the stack from top to bottom..*

**

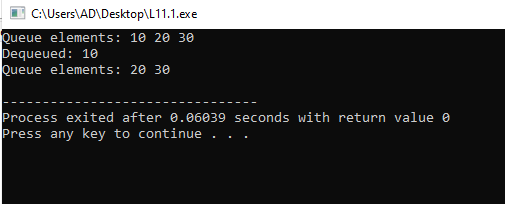
***TASK #10.2***

*This implementation defines a stack using a linked list, where each node contains a data value and a pointer to the next node. The****push****method creates a new node and adds it to the top of the stack, while the****pop****method removes the top node and displays its value. The****display****method traverses the linked list and prints all elements from top to bottom.****.***

**

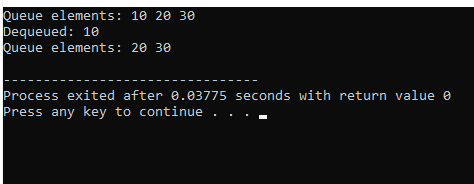
***TASK #11.1***

*This implementation defines a queue using a fixed-size array of 100 elements. The****enqueue****method adds an element to the rear of the queue, while the****dequeue****method removes the front element and displays it. The****display****method prints all elements in the queue from front to rear..*

******

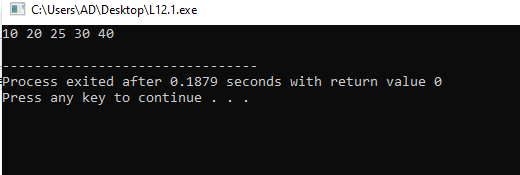
***TASK #11.2***

*This implementation defines a queue using a linked list, where each node contains a data value and a pointer to the next node. The****enqueue****method creates a new node and adds it to the rear of the queue, while the****dequeue****method removes the front node and displays its value. The****display****method traverses the linked list and prints all elements from front to rear.*

**

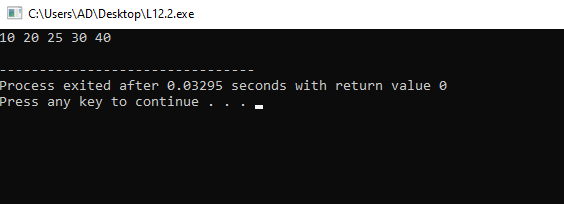
***TASK #12.1***

*The Binary Search Tree (BST) implementation allows for the insertion of integer values while maintaining the BST property, where the left child is less than the parent node and the right child is greater. The****insert****function recursively places new values in the correct position, and the****traverse****function performs an in-order traversal to print the values in sorted order.*

**

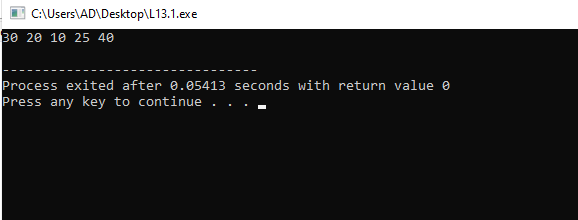
***TASK #12.2***

*The AVL Tree implementation is a self-balancing binary search tree that maintains its balance through rotations during insertion. Each node keeps track of its height, and the****insert****function ensures that the tree remains balanced by performing right and left rotations as needed. The****traverse****function performs an in-order traversal, similar to the BST, to display the values in sorted order.*

**

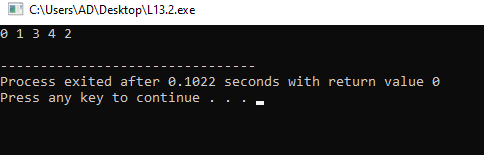
***TASK #13.1***

*This code implements a binary tree where nodes can be inserted while maintaining the binary search tree property. The****dfsTraversal****method performs a depth-first search (DFS) using recursion to visit and print each node in pre-order. The output displays the nodes in the order they are visited.*

******

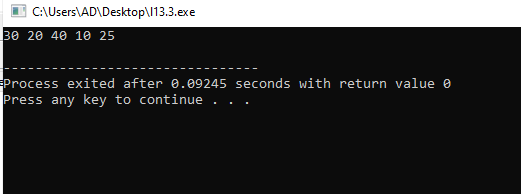
***TASK #13.2***

*This code represents a graph using an adjacency matrix, allowing edges to be added between vertices. The****dfsTraversal****method uses a stack to perform a depth-first search (DFS) starting from a specified vertex, marking nodes as visited and printing them in the order they are explored. The output shows the traversal path through the graph.*

******

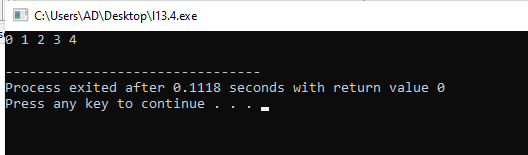
***TASK #13.3***

*This code implements a binary tree with a method for inserting nodes and a breadth-first search (BFS) traversal. The****bfsTraversal****method uses a queue to explore the tree level by level, printing each node as it is visited. The output reflects the hierarchical structure of the tree.*

**

***TASK #13.4***

*This code represents a graph using an adjacency matrix and provides functionality to add edges between vertices. The****bfsTraversal****method performs a breadth-first search (BFS) starting from a specified vertex, using a queue to manage the order of exploration and marking nodes as visited. The output displays the nodes in the order they are traversed..*

**