

**DSA-Lab Tasks**

**Task\_01**

**Explanation:**

* **n is a normal integer.**
* **myPtr store the memory address of n.**
* **using \*myPtr, we can modify n without directly accessing it.**
* **The output will show the n value and modified value also.**

**Task\_02**

**Explanation:**

* **First we create a function findMaximum.**
* **We initialize an array with 0 index.**
* **Inside the function the for loop runs to check the highest value and returns maximumValue.**
* **Inside the main function we declare the values of array and then use building function sizeof for checking the size of array and performing a function. At last we print the maximum value.**
* **The time complexity of this function is O(n). Because loop inside the function runs through n element.**

**Task\_03**

**Explanation:**

**Singly linked list is a sequence of nodes where each node contains data and pointer to the next node.**

**In this task we implement two insertion methods:**

**InsertAtStart: insert a new node at start becomes first node.**

**InsertAtLast: new node is added at last.**

* **Inserting at start, updates the new node's next pointer to point to the od head.**
* **Inserting at last traverse to the last node and update next pointer.**
* **Updated lists are displayed in display function.**

**Task\_04**

**Explanation:**

**In this task we need to insert a node at a given position in our linked list.**

* **Inserting at start at position 1 treat it as inserting at start.**
* **Traverse the position before the requirement index.**
* **Update the pointer to insert new node at the correct location.**
* **Handling case position where is out of range.**
* **And last again display the modified nodes in display function.**

**Task\_05**

**Explanation:**

* **Function we need to display the nodes.**
* **First node head of the list, last node traverse to the last, nth node traverse to the given position and last center node.in this we use slow and fast Pointers method.**
* **First node points to the head's data, last node traverse to the last and it data, nth traverse to it and it data and points to it, as so center node.**
* **We starts from head and traverse N-1 to reach required node.**
* **Using display method to Display nodes. First counting nodes, finding middle index by using count / 2 + 1, traverse the middle node and last print it.**

**Task\_06**

**Explanation:**

**In this task we want to delete specific nodes from singly linked list.**

* **Each function traverses to the list differently based on node to be deleted.**
* **In deleting first node first move head to head->next. Delete head previous.**
* **In deleting last node, we traverse the second last node (temp->next->next ==NULL). First setting (next->next ==NULL ) and then delete the last node.**
* **In the function deleting nth node first we traverse n-1 time to reach to the (n-1)th node. Then adjust prev->next = temp-> next to skip the nth. Now delete the nth node.**
* **In the function deleting center node first counting All nodes finding middle index and then delete the center node.**

**Task\_07**

**Explanation:**

**In this task we are inserting and displaying nodes in doubly linked lists. It contains two pointers next that points to the next node, and previous (prev) points to the previous node.**

**In this Doubly linked list we used these functions:**

**Insert function at start, end, at nth position, at center and at last we display it forward and reverse position.**

* **In inserting at start we set a new node newNode->next = head. Update it by head->prev = newNode, move head to new node.**
* **In function insert at end traverse to the last node (temp->next == NULL), seting temp-> = newNode and at last seting newNode->prev = temp.**
* **In function inserting at nth position traverse N-1 times to reach (N-1)th node, Insert new node and update next/prev pointer.**
* **In insert at center first count total nodes, finding middle by (count / 2 + 1), and at last call the function insert\_at\_nth(mid).**
* **In display method we start from head and moving to the next.**
* **In last reverse display first we traverse to the last node and printing using prev.**

**Task\_08**

**Explanation:**

**We apply two tasks in this lab\_8 or task\_8, first is Merging two singly linked lists, second is Merging Two Doubly lists.**

* **In Merging Two Singly Linked Lists we maintain two separate singly lists using (head1 and head2). To merge traverse to the last node of head1 and link it to the head2. At last using display function to display it.**
* **In Merging Two Doubly Linked Lists we maintain two separate doubly linked lists (head1 and head2). First to merge we traverse the last node of head1 and link it to head2. Updating head2->prev to point to the last node of head1. At last finaly display method to show the Merged linked lists.**
* **Singly linked list merge straightforward, connecting list1's last node to list2.**
* **Doubly linked list requires prev pointer for updates for proper reverse traversal.**

**Task\_09**

**Explanation:**

**A circular linked list (cList) is a variation of linked list. In which the last node's next pointer points back to the first node instead of null.**

**It can be singly only next pointer or doubly both next and prev pointers linked.**

**In circular linked list we are using functions to insert at start, insert at last, insert at nth, insert at center, and display forward and display reverse functions. The reverse display only for doubly circular linked list.**

* **In the function insert at start first creating new node. If the list is empty, make newNode->next = newNode pointing to itself. If list is non empty traverse to the last node. Update lastNode->next = newNode. Seting newNode->next = head. and at last update head = newNode.**
* **In the function insert at last creating new node first again if it is empty make newNode->next = newNode pointing to itself. If list is non empty traverse to the last node (temp->next == head). Updating temp->next == head. Seting newNode->next = head.**
* **In the function insert at nth position if position == 1, call the function inset at start(data). If position is not 1 then traverse N-1 node. Inserting node in between by updating next pointer.**
* **Inserting at center position first counting total nodes. After counting finding middle index (count / 2 + 1 ). Lastly call the function insert at nth (mid).**
* **After inserting methods, we create display method starting from head and keep printing until we reach head again.**
* **In display reverse traverse the last node and then print nodes backwards using previous (prev) pointer.**
* **In Circular linked list loops back to the head.**
* **Insert operations involves updating the last node's next pointer.**
* **Display stops when it reaches back to head.**