Aim: Implement Singly Linked List ADT.

Theory:

A linked list is a linear data structure used to store elements of the same data type but not in contiguous memory locations. It is a collection of nodes where each node contains a data field and a next pointer indicating the address of the next node. So only the current node in the list knows where the next element of the list is stored. In this article, we will learn how to implement a singly linked list in C.

Procedure

1. Node Structure:

    - Define a node with:

    - data: Value stored in the node.

    - next: Pointer to the next node.

2. Initialize the List:

    - Set the head pointer to NULL.

3. Insertion Operations:

    - At the Beginning:

    - Create a new node, set its next to the current head, and update the head.

    - At the End:

    - Create a new node with next as NULL. If the list is empty, update head. Otherwise, traverse to the last node and link it to the new node.

    - At a Specific Position:

    - Traverse to the node before the desired position, create a new node, and adjust pointers accordingly.

4. Deletion Operations:

- From the Beginning:
- If the list is not empty, move head to head->next and free the old head.

- From the End:
- Traverse to the second-last node, set its next to NULL, and free the last node.

-From a Specific Position:

- Traverse to the node before the position, adjust its next to skip the node to be deleted, and free that node.

5. Traversal:

- Start at head and process each node's data until reaching NULL.

Code:

```
// C Program for Implementation of Singly Linked List
#include <stdio.h>
#include <stdlib.h>

// Define the Node structure struct
Node {
   int data;
   struct Node* next;
};

// Function to create a new node struct
Node* createNode(int data) {     struct
```

```
Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
newNode->data = data;     newNode-
>next = NULL;
   return newNode;
}

// Function to insert a new element at the beginning of the singly linked list
void insertAtFirst(struct Node** head, int data) {     struct Node*
newNode = createNode(data);     newNode->next = *head;
   *head = newNode;
}

// Function to insert a new element at the end of the singly linked list
void insertAtEnd(struct Node** head, int data) {     struct Node*
newNode = createNode(data);     if (*head == NULL) {         *head
= newNode;
       return;
   }
   struct Node* temp = *head;
while (temp->next != NULL) {
temp = temp->next;
   }
   temp->next = newNode;
}

// Function to insert a new element at a specific position in the singly linked list
void insertAtPosition(struct Node** head, int data, int position) {
```

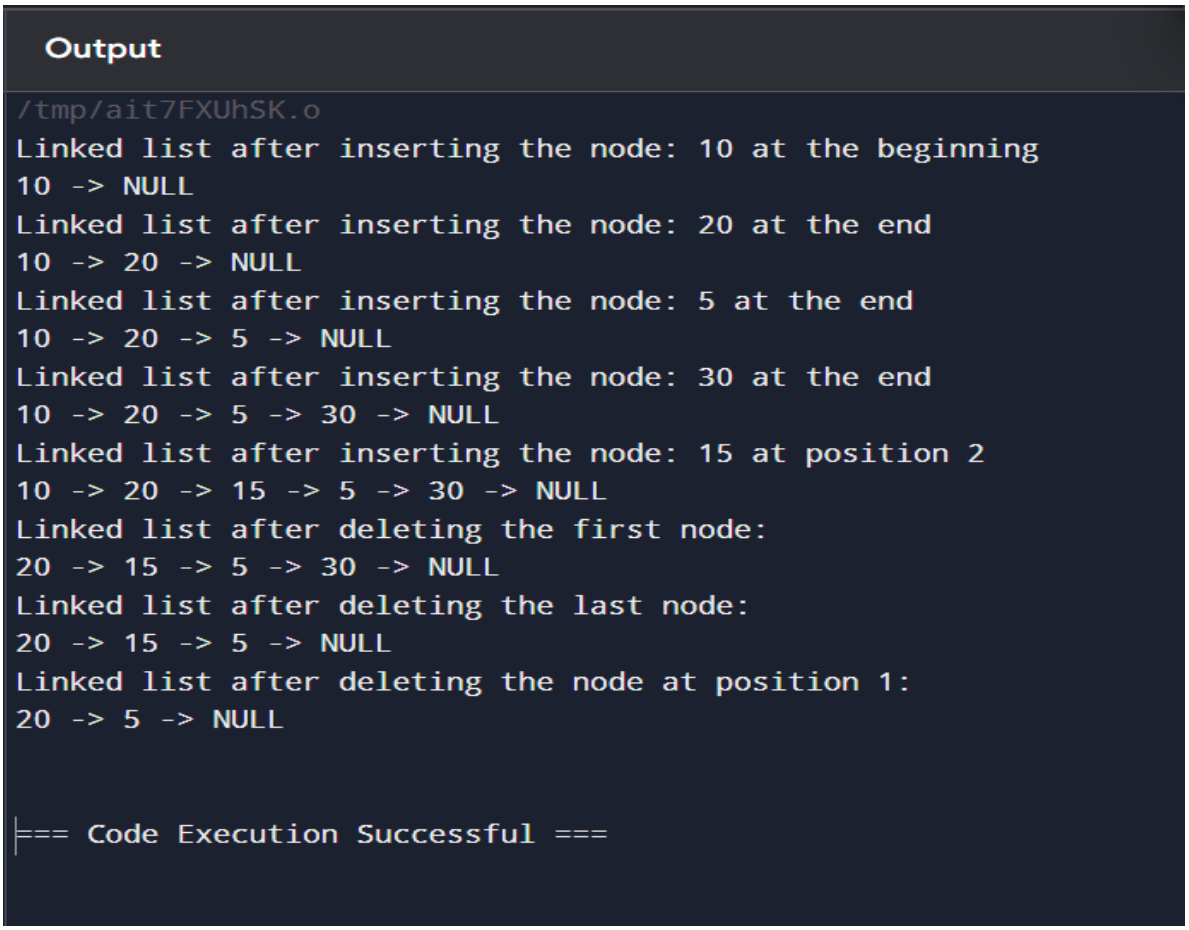
```
    struct Node* newNode = createNode(data);
    if (position == 0) {
insertAtFirst(head, data);
        return;
    }
    struct Node* temp = *head;     for (int i = 0; temp !=
NULL && i < position - 1; i++) {         temp = temp-
>next;
    }
    if (temp == NULL) {
        printf("Position out of range\n");
free(newNode);
        return;
    }
    newNode->next = temp->next;     temp-
>next = newNode;
}

// Function to delete the first node of the singly linked list
void deleteFromFirst(struct Node** head) {     if (*head
== NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;
*head = temp->next;     free(temp);
}

// Function to delete the last node of the singly linked list
```

```
void deleteFromEnd(struct Node** head) {
if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;     if (temp->next ==
NULL) { // Only one node in the list
        free(temp);
        *head = NULL;
        return;
    }
    while (temp->next->next != NULL) {
temp = temp->next;
    }
    free(temp->next);     temp-
>next = NULL;
}

// Function to delete a node at a specific position in the singly linked list
void deleteAtPosition(struct Node** head, int position) {     if (*head
== NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;
    if (position == 0) {
deleteFromFirst(head);
        return;
    }
```

```
    for (int i = 0; temp != NULL && i < position - 1; i++) {
temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("Position out of range\n");
        return;
    }
    struct Node* next = temp->next->next;
free(temp->next);     temp->next = next;
}

// Function to print the LinkedList
void print(struct Node* head) {
struct Node* temp = head;     while
(temp != NULL) {
        printf("%d -> ", temp->data);
temp = temp->next;
    }
    printf("NULL\n");
}

// Driver Code int main() {
struct Node* head = NULL;

    insertAtFirst(&head, 10);     printf("Linked list after inserting the
node: 10 at the beginning \n");     print(head);
```

```
    printf("Linked list after inserting the node: 20 at the end \n");
insertAtEnd(&head, 20);
    print(head);

    printf("Linked list after inserting the node: 5 at the end \n");
insertAtEnd(&head, 5);
    print(head);

    printf("Linked list after inserting the node: 30 at the end \n");
insertAtEnd(&head, 30);
    print(head);

    printf("Linked list after inserting the node: 15 at position 2 \n");
insertAtPosition(&head, 15, 2);     print(head);

    printf("Linked list after deleting the first node: \n");
deleteFromFirst(&head);
    print(head);

    printf("Linked list after deleting the last node: \n");
    deleteFromEnd(&head);
    print(head);

    printf("Linked list after deleting the node at position 1: \n");
deleteAtPosition(&head, 1);     print(head);


    return 0;
}
```

Output:



```
Output
/tmp/ait7FXUhSK.o
Linked list after inserting the node: 10 at the beginning
10 -> NULL
Linked list after inserting the node: 20 at the end
10 -> 20 -> NULL
Linked list after inserting the node: 5 at the end
10 -> 20 -> 5 -> NULL
Linked list after inserting the node: 30 at the end
10 -> 20 -> 5 -> 30 -> NULL
Linked list after inserting the node: 15 at position 2
10 -> 20 -> 15 -> 5 -> 30 -> NULL
Linked list after deleting the first node:
20 -> 15 -> 5 -> 30 -> NULL
Linked list after deleting the last node:
20 -> 15 -> 5 -> NULL
Linked list after deleting the node at position 1:
20 -> 5 -> NULL


=== Code Execution Successful ===
```

Conclusion: This experiment focuses on implementing a Singly Linked List ADT. It explores the structure of linked lists, including nodes, pointers, and basic operations like insertion, deletion, and traversal. It highlights the flexibility of linked lists compared to arrays.