	PUNE INSTITUTE OF COMPUTER TECHNOLOGY PUNE - 411043	
	Department of Electronics & Telecommunication	
	ASSESMENT YEAR: 2024-2025	CLASS: SE
	SUBJECT: DATA STRUCTURES	
EXPT No:	LAB Ref: SE/2024-25/	Starting date:
	Roll No:22203	Submission date:

Title:	**Operations on Linked list**
Problem Statement	Write a program in C to create a singly linked list. Perform following operations on it A. Insert (at front, at end, in the middle) B. Delete(front, end, middle) C. Display and Display reverse D. Revert the Singly linked list

Programmer Name: Arpan Agrawal
Batch: E6

```
#include <stdio.h>
#include <stdlib.h>

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

// Function prototypes
void insertFront(struct Node** head_ref, int new_data);
void insertEnd(struct Node** head_ref, int new_data);
void insertMiddle(struct Node** head_ref, int new_data, int position);
void deleteFront(struct Node** head_ref);
void deleteEnd(struct Node** head_ref);
void deleteMiddle(struct Node** head_ref, int position);
void display(struct Node* head);
void displayReverse(struct Node* head);
void revertList(struct Node** head_ref);

// Insert a node at the front
void insertFront(struct Node** head_ref, int new_data) {
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = *head_ref;
   *head_ref = new_node;
}

// Insert a node at the end
```

```
void insertEnd(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    struct Node* last = *head_ref;
    new_node->data = new_data;
    new_node->next = NULL;

    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }

    while (last->next != NULL) {
        last = last->next;
    }

    last->next = new_node;
}

// Insert a node at a specific position
void insertMiddle(struct Node** head_ref, int new_data, int position) {
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    new_node->data = new_data;

    if (position == 0) {
        insertFront(head_ref, new_data);
        return;
    }

    struct Node* temp = *head_ref;
    for (int i = 0; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Position out of bounds\n");
        free(new_node);
        return;
    }
```

```
    new_node->next = temp->next;
    temp->next = new_node;
}

// Delete a node at the front
void deleteFront(struct Node** head_ref) {
    if (*head_ref == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = *head_ref;
    *head_ref = (*head_ref)->next;
    free(temp);
}

// Delete a node at the end
void deleteEnd(struct Node** head_ref) {
    if (*head_ref == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = *head_ref;

    if (temp->next == NULL) {
        *head_ref = NULL;
        free(temp);
        return;
    }

    struct Node* prev = NULL;
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
```

```
    prev->next = NULL;
    free(temp);
}

// Delete a node at a specific position
void deleteMiddle(struct Node** head_ref, int position) {
    if (*head_ref == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = *head_ref;

    if (position == 0) {
        deleteFront(head_ref);
        return;
    }

    struct Node* prev = NULL;
    for (int i = 0; i < position && temp != NULL; i++) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Position out of bounds\n");
        return;
    }

    prev->next = temp->next;
    free(temp);
}

// Display the list
void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
```

```
    }
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

// Display the list in reverse order (recursive approach)
void displayReverse(struct Node* head) {
    if (head == NULL)
        return;

    displayReverse(head->next);
    printf("%d -> ", head->data);
}

// Revert the list (swap adjacent nodes)
void revertList(struct Node** head_ref) {
    struct Node* current = *head_ref;

    // If the list is empty or has only one element, no need to revert
    if (current == NULL || current->next == NULL) {
        return;
    }

    struct Node* prev = NULL;
    struct Node* next = NULL;

    *head_ref = current->next;  // The new head will be the second node

    while (current != NULL && current->next != NULL) {
        next = current->next;          // Move to the next pair
        current->next = next->next;    // Link current node to the next of next
        next->next = current;          // Reverse the current pair

        if (prev != NULL) {
            prev->next = next;         // Connect the previous pair with the current one
```

```
        }

        prev = current;              // Move prev to current node
        current = current->next;     // Move to the next pair
    }
}

// Main function to test the program
int main() {
    struct Node* head = NULL;
    int choice, value, position;

    // Step 1: Build the initial list
    printf("Enter values to create the initial list (Enter -1 to stop):\n");
    while (1) {
        printf("Enter value: ");
        scanf("%d", &value);
        if (value == -1) {
            break;
        }
        insertEnd(&head, value);
    }

    // Display the list after the user is done creating it
    printf("Initial List: ");
    display(head);

    // Step 2: Perform operations
    do {
        printf("\nMenu:\n");
        printf("1. Insert at front\n");
        printf("2. Insert at end\n");
        printf("3. Insert in middle\n");
        printf("4. Delete at front\n");
        printf("5. Delete at end\n");
        printf("6. Delete in middle\n");
        printf("7. Display list\n");
        printf("8. Display list in reverse\n");
```

```
printf("9. Revert the list (swap adjacent nodes)\n");
printf("10. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
   case 1:
      printf("Enter the value to insert at front: ");
      scanf("%d", &value);
      insertFront(&head, value);
      printf("List after insertion: ");
      display(head);
      break;
   case 2:
      printf("Enter the value to insert at end: ");
      scanf("%d", &value);
      insertEnd(&head, value);
      printf("List after insertion: ");
      display(head);
      break;
   case 3:
      printf("Enter the value and position to insert in middle: ");
      scanf("%d%d", &value, &position);
      insertMiddle(&head, value, position);
      printf("List after insertion: ");
      display(head);
      break;
   case 4:
      deleteFront(&head);
      printf("List after deletion: ");
      display(head);
      break;
   case 5:
      deleteEnd(&head);
      printf("List after deletion: ");
      display(head);
      break;
   case 6:
```

```
            printf("Enter the position to delete from middle: ");
            scanf("%d", &position);
            deleteMiddle(&head, position);
            printf("List after deletion: ");
            display(head);
            break;
        case 7:
            printf("List: ");
            display(head);
            break;
        case 8:
            printf("List in reverse: ");
            displayReverse(head);
            printf("NULL\n");
            break;
        case 9:
            revertList(&head);
            printf("List after reversion (adjacent nodes swapped): ");
            display(head);
            break;
        case 10:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
    }
  } while (choice != 10);

  return 0;
}
```

Output :-

```
Enter values to create the initial list (Enter -1 to stop):
Enter value: 562
Enter value: 3
Enter value: 14
Enter value: 87
```

Enter value: 954
Enter value: 10
Enter value: -1
Initial List: 56 -> 3 -> 14 -> 87 -> 95 -> 10 -> NULL

Menu:
1. Insert at front
2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end
6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit
Enter your choice: 1
Enter the value to insert at front: 22
List after insertion: 22 -> 56 -> 3 -> 14 -> 87 -> 95 -> 10 -> NULL

Menu:
1. Insert at front
2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end
6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit
Enter your choice:
2
Enter the value to insert at end: 5
List after insertion: 22 -> 56 -> 3 -> 14 -> 87 -> 95 -> 10 -> 5 -> NULL

Menu:
1. Insert at front

2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end
6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit

Enter your choice: 3
Enter the value and position to insert in middle: 54 3
List after insertion: 22 -> 56 -> 3 -> 54 -> 14 -> 87 -> 95 -> 10 -> 5 -> NULL

Menu:
1. Insert at front
2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end
6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit

Enter your choice: 4
List after deletion: 56 -> 3 -> 54 -> 14 -> 87 -> 95 -> 10 -> 5 -> NULL

Menu:
1. Insert at front
2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end
6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit

Enter your choice: 5
List after deletion: 56 -> 3 -> 54 -> 14 -> 87 -> 95 -> 10 -> NULL

Menu:
1. Insert at front
2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end
6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit
Enter your choice: 6
Enter the position to delete from middle: 5
List after deletion: 56 -> 3 -> 54 -> 14 -> 87 -> 10 -> NULL

Menu:
1. Insert at front
2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end
6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit
Enter your choice: 7
List: 56 -> 3 -> 54 -> 14 -> 87 -> 10 -> NULL

Menu:
1. Insert at front
2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end

6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit
Enter your choice: 8
List in reverse: 10 -> 87 -> 14 -> 54 -> 3 -> 56 -> NULL

Menu:
1. Insert at front
2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end
6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit
Enter your choice: 9
List after reversion (adjacent nodes swapped): 3 -> 56 -> 14 -> 54 -> 10 -> 87 -> NULL

Menu:
1. Insert at front
2. Insert at end
3. Insert in middle
4. Delete at front
5. Delete at end
6. Delete in middle
7. Display list
8. Display list in reverse
9. Revert the list (swap adjacent nodes)
10. Exit
Enter your choice: 10
Exiting...