# Program 17

Ques 17) WAP to perform various Operation of Singly Linked List

1) Insertion

   a) At beginning

   b) At End

   c) At Location

2) Deletion

   a) At beginning

   b) At End

   c) At Location

3) Traversal?

Sol:

## **Algorithm**:

1. Start.

2. Initialize:

   - Set `head` to `NULL` (indicating an empty list).

3. Display Menu Options:

   - Show options:

     a) Insert at Beginning: Insert an element at the start of the list.
     b) Insert at End: Insert an element at the end of the list.
     c) Insert at Location: Insert an element at a specific location in the list.
     d) Delete from Beginning: Remove the first element of the list.
     e) Delete from End: Remove the last element of the list.

f) Delete from Location: Remove an element from a specific location in the list.
g) Traverse: Display all elements in the list.

- Prompt the user to enter their choice.

4. Handle User Choice:

```
1) If the user selects 1 (Insert at Beginning):
    i) Go to Step 5.
2) If the user selects 2 (Insert at End):
    i) Go to Step 6.
3) If the user selects 3 (Insert at Location):
    i) Go to Step 7.
4) If the user selects 4 (Delete from Beginning):
    i) Go to Step 8.
5) If the user selects 5 (Delete from End):
    i) Go to Step 9.
6) If the user selects 6 (Delete from Location):
    i) Go to Step 10.
7) If the user selects 7 (Traverse):
    i) Go to Step 11.
8) If the choice is invalid, display "Invalid choice" and go back to Step 3.
```

5. Insert at Beginning:

- Create a new node.
- If memory allocation fails, display "Overflow" and go back to Step 3.
- Prompt the user to enter a value for the new node.
- Set the new node's `next` pointer to `head`.
- Update `head` to the new node.
- Go back to Step 3.

6. Insert at End:

- Create a new node.
- If memory allocation fails, display "Overflow" and go back to Step 3.
- Prompt the user to enter a value for the new node.
- If the list is empty (`head` is `NULL`), set `head` to the new node.
- Otherwise, traverse to the last node in the list.
- Set the `next` pointer of the last node to the new node.
- Go back to Step 3.

7. Insert at Location:

- Prompt the user to enter the location (position) to insert the new node.
- If the location exceeds the list length + 1, display an error message and go back to Step 3.
- Create a new node and prompt the user to enter a value for it.
- Traverse to the specified location (one node before the insertion point).
- Set the new node’s `next` pointer to the `next` pointer of the current node.
- Set the current node’s `next` pointer to the new node.
- Go back to Step 3.

8. Delete from Beginning:

- If the list is empty (`head` is `NULL`), display "Underflow" and go back to Step 3.
- Otherwise, save the `head` node in a temporary variable.
- Set `head` to `head->next`.
- Display the value of the deleted node.
- Free the memory of the deleted node.
- Go back to Step 3.

9. Delete from End:

- If the list is empty (`head` is `NULL`), display "Underflow" and go back to Step 3.
- Traverse to the second-to-last node in the list.
- Set the `next` pointer of the second-to-last node to `NULL`.
- Display the value of the deleted node.
- Free the memory of the deleted node.
- Go back to Step 3.

10. Delete from Location:

- Prompt the user to enter the location (position) to delete the node.
- If the location exceeds the list length, display an error message and go back to Step 3.
- Traverse to the node one position before the location.
- Save the node at the specified location in a temporary variable.
- Set the `next` pointer of the current node to the `next` pointer of the node being deleted.
- Display the value of the deleted node.
- Free the memory of the deleted node.
- Go back to Step 3.

11. Traverse:

- If `head` is `NULL`, display "List is empty" and go back to Step 3.
- Otherwise, start at `head` and display each node’s `info` value until reaching the end of the list.
- Go back to Step 3.

12. Ask to Continue:

- Prompt the user to continue (enter 'Y' for yes, 'N' for no).
- If the user enters 'Y', go back to Step 3.
- If the user enters 'N', proceed to Step 13.

13. End.

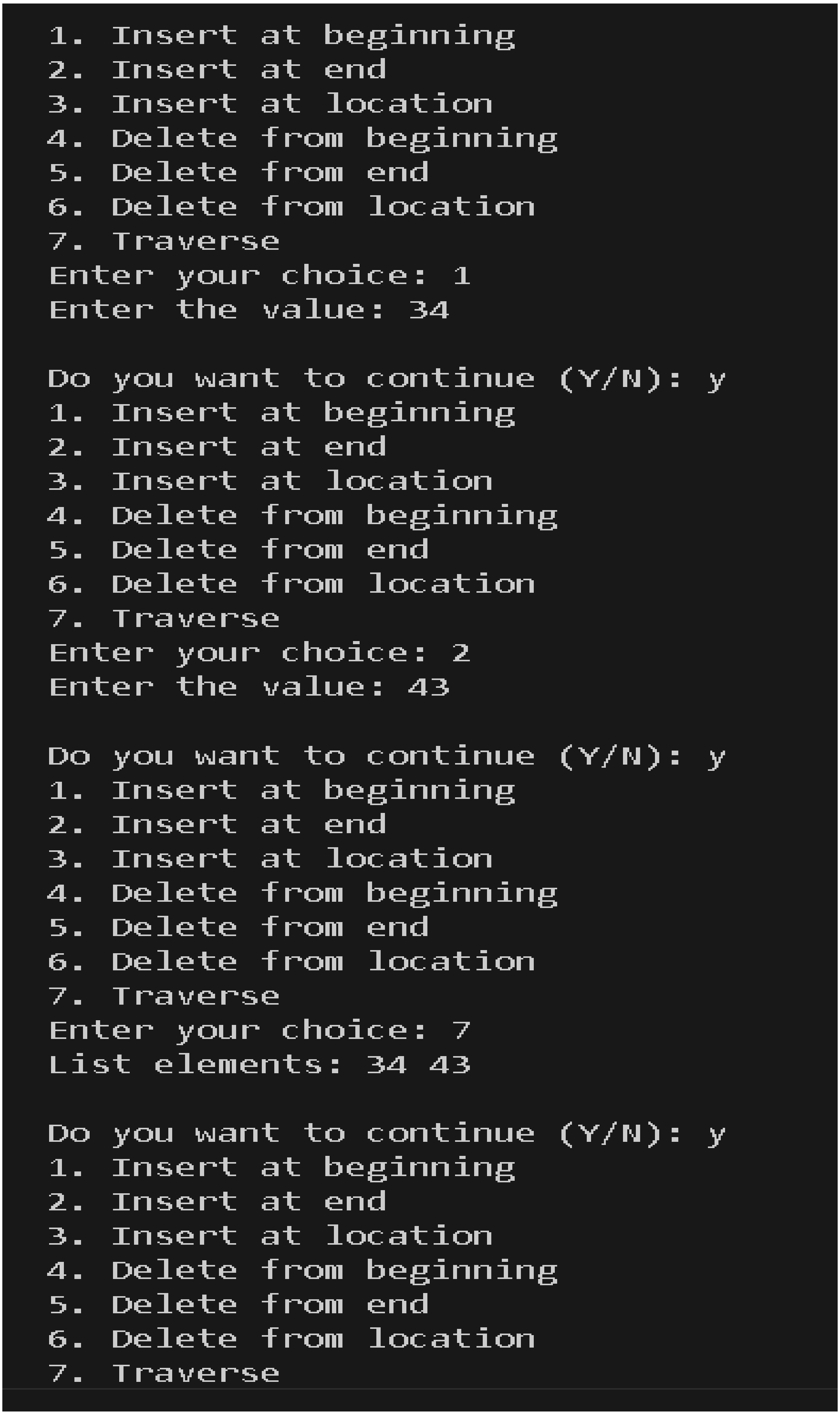
## **Code:**

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



// Define the structure for a node
struct node {
    int info;
    struct node *next;
};



// Global variable for the head of the linked list
struct node *head = NULL;



// Function to insert an element at the beginning of the list
void insertAtBeginning() {
    struct node *ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL) {
        printf("Overflow \n");
        return;
    }
    printf("Enter the value: ");
    scanf("%d", &ptr->info);  // Get the value to insert
    ptr->next = head;  // New node points to the current head
    head = ptr;  // Update head to point to the new node
}


// Function to insert an element at the end of the list
void insertAtEnd() {
    struct node *ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL) {
        printf("Overflow \n");
        return;
    }
    printf("Enter the value: ");
    scanf("%d", &ptr->info);  // Get the value to insert
    ptr->next = NULL;  // New node will point to NULL


    if (head == NULL) {
        // If the list is empty, make the new node the head
        head = ptr;
    } else {
        // Traverse to the end of the list and link the new node
        struct node *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = ptr;  // Link the new node
    }
}


// Function to insert an element at a specific location
void insertAtLocation() {
    int loc, i;
```

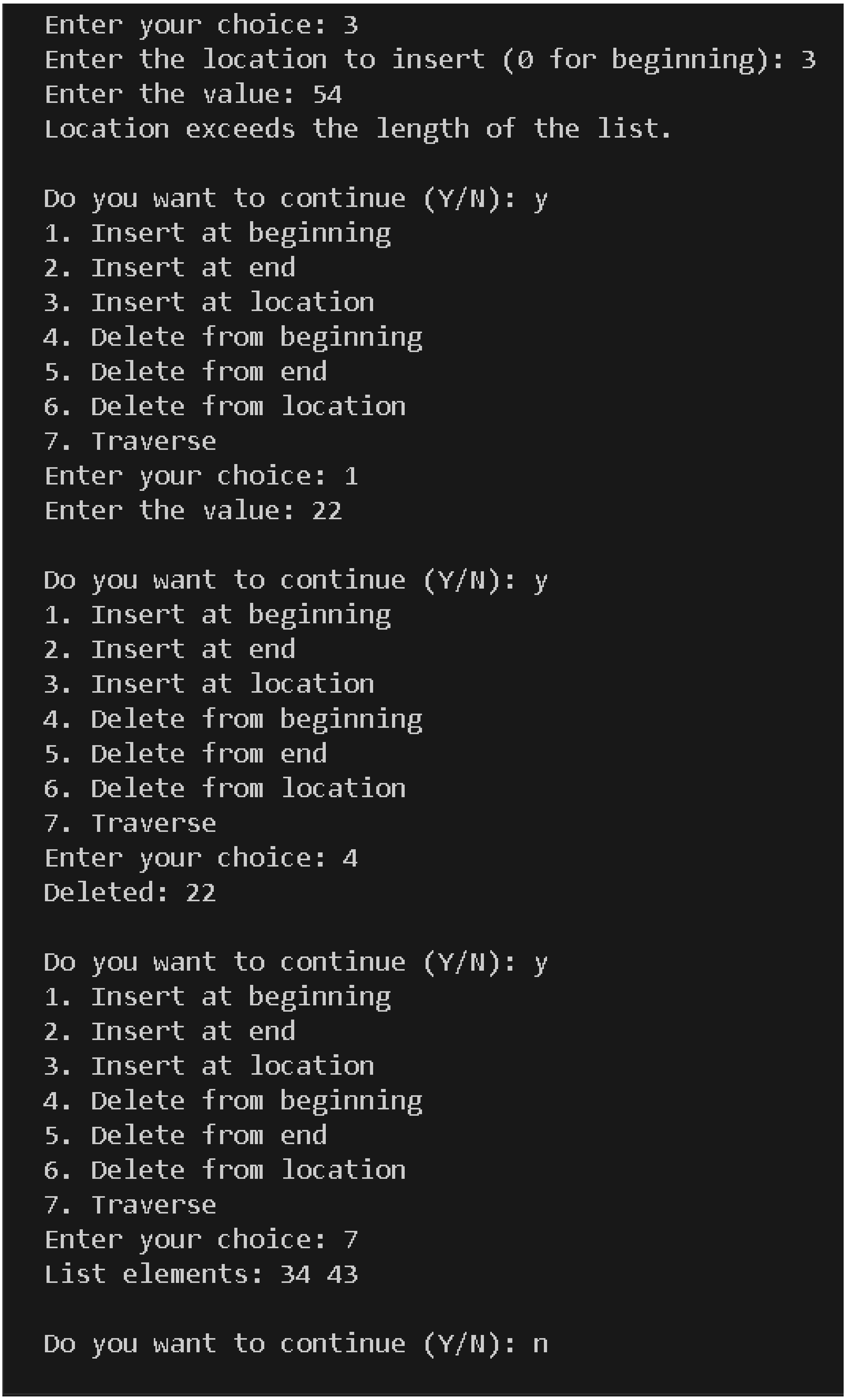
```
printf("Enter the location to insert (0 for beginning): ");
scanf("%d", &loc);


if (loc < 0) {
    printf("Invalid location!\n");
    return;
}


if (loc == 0) {
    insertAtBeginning();
    return;
}


struct node *ptr = (struct node *)malloc(sizeof(struct node));
if (ptr == NULL) {
    printf("Overflow \n");
    return;
}
printf("Enter the value: ");
scanf("%d", &ptr->info);  // Get the value to insert


struct node *temp = head;
for (i = 0; i < loc - 1 && temp != NULL; i++) {
    temp = temp->next;
}


if (temp == NULL) {
    printf("Location exceeds the length of the list.\n");
    free(ptr);
} else {
    ptr->next = temp->next;  // Link the new node to the next node
```

```
        temp->next = ptr;  // Link the previous node to the new node
    }
}


// Function to delete a node from the beginning
void deleteFromBeginning() {
    if (head == NULL) {
        printf("Underflow \n");
        return;
    }
    struct node *temp = head;
    head = head->next;  // Move head to the next node
    printf("Deleted: %d\n", temp->info);
    free(temp);  // Free the memory of the deleted node
}

// Function to delete a node from the end
void deleteFromEnd() {
    if (head == NULL) {
        printf("Underflow \n");
        return;
    }
    struct node *temp = head;
    if (temp->next == NULL) {
        // If there's only one node
        printf("Deleted: %d\n", temp->info);
        free(temp);
        head = NULL;
    } else {
        // Traverse to the second last node
        while (temp->next->next != NULL) {
            temp = temp->next;
        }
        printf("Deleted: %d\n", temp->next->info);
        free(temp->next);  // Free the last node
        temp->next = NULL;  // Set the second last node's next to NULL
    }
}


// Function to delete a node from a specific location
void deleteFromLocation() {
    int loc, i;
    printf("Enter the location to delete (0 for beginning): ");
    scanf("%d", &loc);



    if (loc < 0) {
        printf("Invalid location!\n");
        return;
    }

    if (loc == 0) {
        deleteFromBeginning();
        return;
    }

    struct node *temp = head;
    struct node *prev = NULL;

    for (i = 0; i < loc && temp != NULL ; i++) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Location exceeds the length of the list.\n");
    } else {
        printf("Deleted: %d\n", temp->info);
        prev->next = temp->next;  // Link the previous node to the next node
        free(temp);  // Free the memory of the deleted node
    }
}


// Function to traverse the list
void traverse() {
    struct node *ptr = head;
    if (ptr == NULL) {
        printf("List is empty.\n");
        return;
    }
    printf("List elements: ");
    while (ptr != NULL) {  // Traverse until the end of the list
        printf("%d ", ptr->info);
        ptr = ptr->next;
    }
    printf("\n");
}


// Main function
int main() {
    int choice;
    char ch;
    do {
        printf("1. Insert at beginning \n");
        printf("2. Insert at end \n");
        printf("3. Insert at location \n");
        printf("4. Delete from beginning \n");
        printf("5. Delete from end \n");
        printf("6. Delete from location \n");
        printf("7. Traverse \n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

           switch (choice) {
}

        case 1:
            insertAtBeginning();
            break;
        case 2:
            insertAtEnd();
            break;
        case 3:
            insertAtLocation();
            break;
        case 4:
            deleteFromBeginning();
            break;
        case 5:
            deleteFromEnd();
            break;
        case 6:
            deleteFromLocation();
            break;
        case 7:
            traverse();
            break;
        default:
            printf("You entered a wrong choice!!\n");
        }


        printf(" \nDo you want to continue (Y/N): ");
        getchar();  // Clear newline character from buffer
        scanf("%c", &ch);  // Get user's choice to continue
    } while (ch == 'Y' || ch == 'y');


    return 0;
}
```

## **Output:**



```
1. Insert at beginning
2. Insert at end
3. Insert at location
4. Delete from beginning
5. Delete from end
6. Delete from location
7. Traverse
Enter your choice: 1
Enter the value: 34

Do you want to continue (Y/N): y
1. Insert at beginning
2. Insert at end
3. Insert at location
4. Delete from beginning
5. Delete from end
6. Delete from location
7. Traverse
Enter your choice: 2
Enter the value: 43

Do you want to continue (Y/N): y
1. Insert at beginning
2. Insert at end
3. Insert at location
4. Delete from beginning
5. Delete from end
6. Delete from location
7. Traverse
Enter your choice: 7
List elements: 34 43

Do you want to continue (Y/N): y
1. Insert at beginning
2. Insert at end
3. Insert at location
4. Delete from beginning
5. Delete from end
6. Delete from location
7. Traverse
```



```
Enter your choice: 3
Enter the location to insert (0 for beginning): 3
Enter the value: 54
Location exceeds the length of the list.

Do you want to continue (Y/N): y
1. Insert at beginning
2. Insert at end
3. Insert at location
4. Delete from beginning
5. Delete from end
6. Delete from location
7. Traverse
Enter your choice: 1
Enter the value: 22

Do you want to continue (Y/N): y
1. Insert at beginning
2. Insert at end
3. Insert at location
4. Delete from beginning
5. Delete from end
6. Delete from location
7. Traverse
Enter your choice: 4
Deleted: 22

Do you want to continue (Y/N): y
1. Insert at beginning
2. Insert at end
3. Insert at location
4. Delete from beginning
5. Delete from end
6. Delete from location
7. Traverse
Enter your choice: 7
List elements: 34 43

Do you want to continue (Y/N): n
```