# LAB-5

# Implementation and Operations on Singly Linked Lists

Q1) Write a program to reverse a singly linked list (implement both iterative and recursive methods).

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

struct simple {
   int x;
   struct simple *next;
};

struct simple* createL(struct simple *head, int x) {
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = x;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
   }
   else {
      struct simple *temp = head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
   return head;
}

struct simple* reverseL(struct simple *head) {
   struct simple *prev = NULL, *curr = head, *next = NULL;
   while (curr != NULL) {
      next = curr->next;
      curr->next = prev;
```

```
        prev = curr;
        curr = next;
    }
    head = prev;
    return head;
}

struct simple* recurssivereverseL(struct simple *curr,struct simple *prev){
    if(curr==NULL){
        return prev;
    }
    struct simple *next=curr->next;
    curr->next=prev;
    return recurssivereverseL(next,curr);
}

int main(){
    int n, val;
    struct simple *head = NULL;

    printf("Enter the number of nodes:\n");
    scanf("%d", &n);

    for(int i=0;i<n;i++){
        printf("Enter the value %d:\n",(i+1));
        scanf("%d", &val);
        head = createL(head, val);
    }

    printf("Existing Linked List :- \n");
    struct simple *temp = head;
    while(temp != NULL){
        printf("%d -> ", temp->x);
        temp = temp->next;
    }
    printf("NULL\n");
    struct simple *origHead = head;

    head = reverseL(head);
    printf("By Iterative, Reversed Linked List :- \n");
    temp = head;
    while(temp != NULL){
```

```
    printf("%d -> ", temp->x);
    temp = temp->next;
  }
  printf("NULL\n");

  head = reverseL(head);
  head = recurssivereverseL(head,NULL);
  printf("By Recursive, Reversed Linked List :- \n");
  temp = head;
  while(temp != NULL){
    printf("%d -> ", temp->x);
    temp = temp->next;
  }
  printf("NULL\n");

  return 0;
}
```

Input & Output

```
Enter the number of nodes:
4
Enter the value 1:
2034
Enter the value 2:
3420
Enter the value 3:
20
Enter the value 4:
34
Existing Linked List :-
2034 -> 3420 -> 20 -> 34 -> NULL
By Iterative, Reversed Linked List :-
34 -> 20 -> 3420 -> 2034 -> NULL
By Recursive, Reversed Linked List :-
34 -> 20 -> 3420 -> 2034 -> NULL
```

Q2) Write a program to search for a given element in a linked list and print its position.

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

struct simple {
   int x;
   struct simple *next;
};

struct simple* createL(struct simple *head, int x) {
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = x;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
   }
   else {
      struct simple *temp = head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
   return head;
}

int main(){
   int n, val, element;
   struct simple *head = NULL;

   printf("Enter the number of nodes:\n");
   scanf("%d", &n);
   printf("Enter the Element to find in Linked List:\n");
   scanf("%d", &element);

   for(int i=0;i<n;i++){
      printf("Enter the value %d:\n",(i+1));
      scanf("%d", &val);
```

```
        head = createL(head, val);
    }

    printf("Existing Linked List :- \n");
    struct simple *temp = head;
    while(temp != NULL){
        printf("%d -> ", temp->x);
        temp = temp->next;
    }
    printf("NULL\n");

    temp = head;
    int pos = 0;
    int found = 0;
    while(temp != NULL){
        if(temp->x == element){
            printf("We got Element %d at position %d in Linked List\n", element,
pos);
            found = 1;
            break;
        }
        temp = temp->next;
        pos++;
    }
    if(!found){
        printf("Element %d not found in Linked List\n", element);
    }

    return 0;
}
```

Input & output

```
Enter the number of nodes:

4

Enter the Element to find in Linked List:

20

Enter the value 1:

34

Enter the value 2:
```

```
2034
Enter the value 3:
3420
Enter the value 4:
20
Existing Linked List :-
34 -> 2034 -> 3420 -> 20 -> NULL
We got Element 20 at position 3 in Linked List
```

Q3) Create a singly linked list of n nodes and display all its elements.

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

struct simple {
   int x;
   struct simple *next;
};

struct simple* createL(struct simple *head, int x) {
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = x;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
   }
   else {
      struct simple *temp = head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
   return head;
}
```

```
int main(){
   int n, val;
   struct simple *head = NULL;

   printf("Enter the number of nodes:\n");
   scanf("%d", &n);

   for(int i=0;i<n;i++){
      printf("Enter the value %d:\n",(i+1));
      scanf("%d", &val);
      head = createL(head, val);
   }

   printf("Existing Linked List :- \n");
   struct simple *temp = head;
   while(temp != NULL){
      printf("%d -> ", temp->x);
      temp = temp->next;
   }
   printf("NULL\n");

   return 0;
}
```

Input & Output

```
Enter the number of nodes:
4
Enter the value 1:
34
Enter the value 2:
20
Enter the value 3:
3420
Enter the value 4:
2034
Existing Linked List :-
34 -> 20 -> 3420 -> 2034 -> NULL
```

Q4) Write a program to detect whether a linked list contains a loop using Floyd's Cycle Detection Algorithms.

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

struct simple {
   int x;
   struct simple *next;
};

struct simple* createL(struct simple *head, int x) {
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = x;
   newNode->next = NULL;

   if(head == NULL){
      head = newNode;
   }
   else{
      struct simple *temp = head;
      while(temp->next != NULL){
         temp = temp->next;
      }
      temp->next = newNode;
   }
   return head;
}

int detectLoop(struct simple *head){
   struct simple *slow = head;
   struct simple *fast = head;

   while(fast != NULL && fast->next != NULL){
      slow = slow->next;
      fast = fast->next->next;

      if(slow == fast){
         return 1;
      }
   }
```

```
    return 0;
}

int main(){
    struct simple *head = NULL;
    int n, val;

    printf("Enter the number of nodes:\n");
    scanf("%d", &n);

    for(int i=0;i<n;i++){
        printf("Enter value %d:\n", i+1);
        scanf("%d", &val);
        head = createL(head, val);
    }
    if(detectLoop(head)){
        printf("Linked List contains a loop.\n");
    } else {
        printf("Linked List does not contain a loop.\n");
    }

    return 0;
}
```

Input & Output

```
Enter the number of nodes:

6

Enter value 1:

20

Enter value 2:

34

Enter value 3:

2034

Enter value 4:

20

Enter value 5:

34

Enter value 6:
```

```
2034

Linked List does not contain a loop.
```

Q5) Implement insertion operations in a singly linked list to insert a node:
• At the beginning
• At the end
• At a given position

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

struct simple {
   int x;
   struct simple *next;
};

struct simple* createL(struct simple *head, int val){
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = val;
   newNode->next = NULL;
   if(head == NULL){
      head = newNode;
   } else {
      struct simple *temp = head;
      while(temp->next != NULL){
         temp = temp->next;
      }
      temp->next = newNode;
   }
   return head;
}

struct simple* insertAtBeginning(struct simple *head, int val){
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = val;
   newNode->next = head;
   head = newNode;
   return head;
```

```c
}

struct simple* insertAtEnd(struct simple *head, int val){
   return createL(head, val);
}

struct simple* insertAtPosition(struct simple *head, int val, int pos){
   if(pos == 0){
      return insertAtBeginning(head, val);
   }
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = val;
   struct simple *temp = head;
   for(int i=0;i<pos-1 && temp != NULL;i++){
      temp = temp->next;
   }
   if(temp == NULL){
      return insertAtEnd(head, val);
   }
   newNode->next = temp->next;
   temp->next = newNode;
   return head;
}

void displayList(struct simple *head){
   struct simple *temp = head;
   while(temp != NULL){
      printf("%d -> ", temp->x);
      temp = temp->next;
   }
   printf("NULL\n");
}

int main(){
   int n, val, pos;
   struct simple *head = NULL;

   printf("Enter number of initial nodes: ");
   scanf("%d",&n);
   for(int i=0;i<n;i++){
      printf("Enter value %d: ", i+1);
      scanf("%d",&val);
```

```
        head = createL(head,val);
    }

    printf("Initial Linked List:\n");
    displayList(head);

    printf("Insert at beginning, enter value: ");
    scanf("%d",&val);
    head = insertAtBeginning(head,val);
    displayList(head);

    printf("Insert at end, enter value: ");
    scanf("%d",&val);
    head = insertAtEnd(head,val);
    displayList(head);

    printf("Insert at position, enter value: ");
    scanf("%d",&val);
    printf("Enter position (0-based): ");
    scanf("%d",&pos);
    head = insertAtPosition(head,val,pos);
    displayList(head);

    return 0;
}
```

Input & Output

```
Enter number of initial nodes: 4

Enter value 1: 20

Enter value 2: 34

Enter value 3: 0 2034

Enter value 4: 3420

Initial Linked List:

20 -> 34 -> 2034 -> 3420 -> NULL

Insert at beginning, enter value: 20  680

680 -> 20 -> 34 -> 2034 -> 3420 -> NULL

Insert at end, enter value: 14

680 -> 20 -> 34 -> 2034 -> 3420 -> 14 -> NULL
```

```
Insert at position, enter value: 1320

Enter position (0-based): 3

680 -> 20 -> 34 -> 1320 -> 2034 -> 3420 -> 14 -> NULL
```

Q6) Write a program to count and display the total number of nodes present in a linked list.

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

struct simple {
   int x;
   struct simple *next;
};

struct simple* createL(struct simple *head, int val){
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = val;
   newNode->next = NULL;
   if(head == NULL){
      head = newNode;
   } else {
      struct simple *temp = head;
      while(temp->next != NULL){
         temp = temp->next;
      }
      temp->next = newNode;
   }
   return head;
}

void displayList(struct simple *head){
   struct simple *temp = head;
   while(temp != NULL){
      printf("%d -> ", temp->x);
      temp = temp->next;
   }
   printf("NULL\n");
}

int countNodes(struct simple *head){
   struct simple *temp = head;
   int count = 0;
   while(temp != NULL){
      count++;
      temp = temp->next;
   }
   return count;
}

int main(){
   int n, val;
   struct simple *head = NULL;

   printf("Enter number of nodes: ");
   scanf("%d",&n);

   for(int i=0;i<n;i++){
      printf("Enter value %d: ", i+1);
      scanf("%d",&val);
      head = createL(head,val);
   }

   printf("Linked List:\n");
   displayList(head);

   int total = countNodes(head);
   printf("Total number of nodes in the Linked List: %d\n", total);

   return 0;
}
```

Input & Output

```
Enter number of nodes: 4

Enter value 1: 20

Enter value 2: 34

Enter value 3: 3420

Enter value 4: 2034
```

```
Linked List:
20 -> 34 -> 3420 -> 2034 -> NULL
Total number of nodes in the Linked List: 4
```

Q7) WWrite a program to merge two sorted linked lists into one sorted linked list.

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

struct simple {
   int x;
   struct simple *next;
};

struct simple* createL(struct simple *head, int val){
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = val;
   newNode->next = NULL;
   if(head == NULL){
      head = newNode;
   } else {
      struct simple *temp = head;
      while(temp->next != NULL){
         temp = temp->next;
      }
      temp->next = newNode;
   }
   return head;
}

void displayList(struct simple *head){
   struct simple *temp = head;
   while(temp != NULL){
      printf("%d -> ", temp->x);
      temp = temp->next;
   }
   printf("NULL\n");
```

```
}

struct simple* mergeSorted(struct simple *l1, struct simple *l2){
   struct simple *head = NULL, *tail = NULL;

   if(l1 == NULL) return l2;
   if(l2 == NULL) return l1;

   if(l1->x <= l2->x){
      head = tail = l1;
      l1 = l1->next;
   } else {
      head = tail = l2;
      l2 = l2->next;
   }

   while(l1 != NULL && l2 != NULL){
      if(l1->x <= l2->x){
         tail->next = l1;
         tail = l1;
         l1 = l1->next;
      } else {
         tail->next = l2;
         tail = l2;
         l2 = l2->next;
      }
   }

   if(l1 != NULL) tail->next = l1;
   if(l2 != NULL) tail->next = l2;

   return head;
}

int main(){
   struct simple *head1 = NULL, *head2 = NULL;
   int n1, n2, val;

   printf("Enter number of nodes for first sorted list: ");
   scanf("%d",&n1);
   for(int i=0;i<n1;i++){
      printf("Enter value %d: ", i+1);
```

```
        scanf("%d",&val);
        head1 = createL(head1,val);
    }

    printf("Enter number of nodes for second sorted list: ");
    scanf("%d",&n2);
    for(int i=0;i<n2;i++){
        printf("Enter value %d: ", i+1);
        scanf("%d",&val);
        head2 = createL(head2,val);
    }

    printf("First Sorted List:\n");
    displayList(head1);
    printf("Second Sorted List:\n");
    displayList(head2);

    struct simple *merged = mergeSorted(head1, head2);
    printf("Merged Sorted List:\n");
    displayList(merged);

    return 0;
}
```

Input & Output

```
Enter number of nodes for first sorted list: 3 4

Enter value 1: 20

Enter value 2: 34

Enter value 3: 2034

Enter value 4: 342- 0

Enter number of nodes for second sorted list: 4

Enter value 1: 10

Enter value 2: 17

Enter value 3: 1017

Enter value 4: 1710

First Sorted List:

20 -> 34 -> 2034 -> 3420 -> NULL

Second Sorted List:
```

```
10 -> 17 -> 1017 -> 1710 -> NULL

Merged Sorted List:

10 -> 17 -> 20 -> 34 -> 1017 -> 1710 -> 2034 -> 3420 -> NULL
```

Q8) Delete nodes from a singly linked list from:
- The beginning
- The end
- A given position

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

struct simple {
   int x;
   struct simple *next;
};

struct simple* createL(struct simple *head, int val){
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = val;
   newNode->next = NULL;
   if(head == NULL) {
      head = newNode;
   } else {
      struct simple *temp = head;
      while(temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
   return head;
}

struct simple* deleteBeginning(struct simple *head){
   if(head == NULL) {
      return NULL;
   } else {
      struct simple *temp = head;
```

```
        head = head->next;
        free(temp);
        return head;
    }
}

struct simple* deleteEnd(struct simple *head){
    if(head == NULL) {
        return NULL;
    } else if(head->next == NULL) {
        free(head);
        return NULL;
    } else {
        struct simple *temp = head;
        while(temp->next->next != NULL) {
            temp = temp->next;
        }
        free(temp->next);
        temp->next = NULL;
        return head;
    }
}

struct simple* deletePosition(struct simple *head, int pos){
    if(head == NULL) {
        return NULL;
    } else if(pos == 0) {
        return deleteBeginning(head);
    } else {
        struct simple *temp = head;
        for(int i=0;i<pos-1 && temp->next != NULL;i++) {
            temp = temp->next;
        }
        if(temp->next == NULL) {
            return head;
        } else {
            struct simple *del = temp->next;
            temp->next = del->next;
            free(del);
            return head;
        }
    }
```

```
}

void displayList(struct simple *head){
   struct simple *temp = head;
   while(temp != NULL) {
      printf("%d -> ", temp->x);
      temp = temp->next;
   }
   printf("NULL\n");
}

int main(){
   int n, val, pos;
   struct simple *head = NULL;

   printf("Enter number of nodes: ");
   scanf("%d",&n);

   for(int i=0;i<n;i++) {
      printf("Enter value %d: ", i+1);
      scanf("%d",&val);
      head = createL(head,val);
   }

   printf("Initial Linked List:\n");
   displayList(head);

   head = deleteBeginning(head);
   printf("After deleting from beginning:\n");
   displayList(head);

   head = deleteEnd(head);
   printf("After deleting from end:\n");
   displayList(head);

   printf("Enter position to delete (0-based): ");
   scanf("%d",&pos);
   head = deletePosition(head,pos);
   printf("After deleting from position %d:\n", pos);
   displayList(head);

   return 0;
```

```
}
```

Input & Output

```
Enter number of nodes: 4
Enter value 1: 20
Enter value 2: 34
Enter value 3: 2034
Enter value 4: 3420
Initial Linked List:
20 -> 34 -> 2034 -> 3420 -> NULL
After deleting from beginning:
34 -> 2034 -> 3420 -> NULL
After deleting from end:
34 -> 2034 -> NULL
Enter position to delete (0-based): 1
After deleting from position 1:
34 -> NULL
```

Q9) Find and print the middle element of a linked list using the two-pointer technique.

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

struct simple {
   int x;
   struct simple *next;
};

struct simple* createL(struct simple *head, int val){
   struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
   newNode->x = val;
   newNode->next = NULL;
   if(head == NULL) {
      head = newNode;
```

```
    } else {
        struct simple *temp = head;
        while(temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    return head;
}

void displayList(struct simple *head){
    struct simple *temp = head;
    while(temp != NULL) {
        printf("%d -> ", temp->x);
        temp = temp->next;
    }
    printf("NULL\n");
}

void printMiddle(struct simple *head){
    if(head == NULL) {
        printf("List is empty\n");
        return;
    } else {
        struct simple *slow = head;
        struct simple *fast = head;
        while(fast != NULL && fast->next != NULL) {
            slow = slow->next;
            fast = fast->next->next;
        }
        printf("Middle element is: %d\n", slow->x);
    }
}

int main(){
    int n, val;
    struct simple *head = NULL;

    printf("Enter number of nodes: ");
    scanf("%d",&n);

    for(int i=0;i<n;i++) {
        printf("Enter value %d: ", i+1);
        scanf("%d",&val);
        head = createL(head,val);
    }

    printf("Linked List:\n");
    displayList(head);

    printMiddle(head);

    return 0;
}
```

Input & Output

```
Enter number of nodes: 4

Enter value 1: 20

Enter value 2: 34

Enter value 3: 2034

Enter value 4: 3420

Linked List:

20 -> 34 -> 2034 -> 3420 -> NULL

Middle element is: 2034
```

Q10) Remove duplicate elements from a sorted linked list so that each element appears only once.

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

struct simple {
    int x;
    struct simple *next;
};

struct simple* createL(struct simple *head, int val){
    struct simple *newNode = (struct simple*)malloc(sizeof(struct simple));
    newNode->x = val;
    newNode->next = NULL;
    if(head == NULL) {
        head = newNode;
    } else {
        struct simple *temp = head;
        while(temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    return head;
}

void displayList(struct simple *head){
    struct simple *temp = head;
    while(temp != NULL) {
        printf("%d -> ", temp->x);
        temp = temp->next;
    }
    printf("NULL\n");
}

struct simple* removeDuplicates(struct simple *head){
    if(head == NULL) {
        return NULL;
    } else {
        struct simple *current = head;
        while(current->next != NULL) {
            if(current->x == current->next->x) {
                struct simple *dup = current->next;
                current->next = current->next->next;
                free(dup);
            } else {
                current = current->next;
            }
        }
        return head;
    }
}

int main(){
    int n, val;
    struct simple *head = NULL;

    printf("Enter number of nodes (sorted list): ");
    scanf("%d",&n);

    for(int i=0;i<n;i++) {
        printf("Enter value %d: ", i+1);
        scanf("%d",&val);
        head = createL(head,val);
    }

    printf("Original Linked List:\n");
    displayList(head);

    head = removeDuplicates(head);
    printf("Linked List after removing duplicates:\n");
    displayList(head);

    return 0;
}
```

Input & Output

```
Enter number of nodes (sorted list): 4

Enter value 1: 20

Enter value 2: 34

Enter value 3: 2034

Enter value 4: 2034

Original Linked List:

20 -> 34 -> 2034 -> 2034 -> NULL

Linked List after removing duplicates:

20 -> 34 -> 2034 -> NULL
```