TUTORIAL IX:
Linked List Implementation
Of
QuickSort and Binary Search
U19CS012 [D-12]

Implement the following operations in context to singly linked list:

1) Creation

2) Insertion (at beginning, middle and end)

3) Deletion (from beginning, middle and end)

---------- [Above Code Same as in Assignment VIII - DS] ----------

4) Sorting the Linked List using QuickSort Algorithm

5) Binary Search on Sorted Linked List

Code:

```c
#include <stdio.h>
// For Exit Function
#include <stdlib.h>

// Structure for Each Node
typedef struct node
{
    int data;
    struct node *next;
} node;

//Helper Functions

// 1 -> Creation of Linked List

// Creation of the Linked List
void CREATION_LL();
// Display of the Whole Linked List
void DISPLAY_LL();
// Returns the Length of Linked List
int LENGTH_LL();

// 2 -> Insertion in Linked List
```

```c
// Insert at the Beginning of Linked List
void Insert_Begin();
// Insert at the End of Linked List
void Insert_End();
// Insert in the Middle Of the Linked List
void Insert_Middle();

// 3 -> Deletion in the Linked List

// Delete at the Beginning of Linked List
void Delete_Begin();
// Delete at the End of Linked List
void Delete_End();
// Delete in the Middle Of the Linked List
void Delete_Middle();
// 1 -> Deletes Node at Particular Position
void Delete_Position();
// 2 -> Deletes all Nodes with Particular Value
void Delete_Value();

// 4 -> QuickSort Implementation

// Appends a single element at the head of the list
// [Similar to Insert at Beginning]
void list_insert_beg(node **list, node *node);

// Concanates Two Lists
void list_concat(node **list1, node *list2);

// QuickSort Implementation using Linked List
void My_QuickSort(node **list);

// 5 -> Binary Search Implementation

// Binary Search Implementation on Linked List
void Binary_Search(int value);

// head Pointer -> head of Linked List
struct node *head = NULL;

int main()
{
    int choice;
    printf("\nLINKED LIST\n");

    printf(" 1 -> Create a Linked List\n");
    printf(" 2 -> Display the Linked List\n");
    printf(" 3 -> Insert at the Beginning of Linked List\n");
    printf(" 4 -> Insert at the End of Linked List\n");
    printf(" 5 -> Insert at Middle of Linked List\n");
```

```
    printf(" 6 -> Delete from Beginning\n");
    printf(" 7 -> Delete from the End\n");
    printf(" 8 -> Delete at Middle of Linked List\n");
    printf(" 9 -> Sort the Linked List using QuickSort Algorithm\n");
    printf(" 10 -> Binary Search [Make Sure List is Sorted]\n");
    printf(" 11 -> Exit\n");

    while (1)
    {
        printf("Enter your choice : ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            CREATION_LL();
            break;
        case 2:
            DISPLAY_LL();
            break;
        case 3:
            Insert_Begin();
            break;
        case 4:
            Insert_End();
            break;
        case 5:
            // Insert at Middle of LL
            Insert_Middle();
            break;
        case 6:
            Delete_Begin();
            break;
        case 7:
            Delete_End();
            break;
        case 8:
            // Delete at Middle of ll
            Delete_Middle();
            break;
        case 9:
            printf("SORTING USING QUICKSORT ALGORITHM\n");
            My_QuickSort(&head);
            DISPLAY_LL();
            break;
        case 10:
            printf("BINARY SEARCH\n");
            int value;
            printf("Enter a Value to Search in Linked List : ");
            scanf("%d", &value);
```

```
            Binary_Search(value);
            break;
        default:
            printf("Enter a Valid Choice!");
            return 0;
            break;
        }
    }

    return 0;
}

// Creation of the Linked List
void CREATION_LL()
{
    struct node *temp, *ptr;

    // Allocate Memory for One Node
    temp = (struct node *)malloc(sizeof(struct node));

    if (temp == NULL)
    {
        printf("No Memory Space on Device!\n");
        exit(0);
    }

    // Creation of New Node { [?]->NULL }
    printf("Enter the Data to be stored in Node : ");
    scanf("%d", &temp->data);
    temp->next = NULL;

    if (head == NULL)
    {
        // If the Linked List is Empty
        head = temp;
    }
    else
    {
        // If the Linked List is Not Empty
        ptr = head;
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        // Insert the Node at the End
        ptr->next = temp;
    }
}

// Display of the Whole Linked List
```

```c
void DISPLAY_LL()
{
    struct node *ptr;

    if (head == NULL)
    {
        printf("List is Empty!!\n");
        return;
    }
    else
    {
        // Head Pointer
        ptr = head;

        printf("Elements of List : ");
        while (ptr != NULL)
        {
            printf("%d -> ", ptr->data);
            ptr = ptr->next;
        }
        printf("NULL\n");
    }
}

// Display the Length of Linked List
int LENGTH_LL()
{
    struct node *ptr;

    if (head == NULL)
    {
        return 0;
    }
    else
    {
        // Head Pointer
        int cnt = 0;
        ptr = head;
        while (ptr != NULL)
        {
            cnt++;
            ptr = ptr->next;
        }
        return cnt;
    }
}

// Insert at the Beginning of Linked List
void Insert_Begin()
{
```

```
    struct node *temp;

    // Allocate Memory for One Node
    temp = (struct node *)malloc(sizeof(struct node));

    if (temp == NULL)
    {
        printf("No Memory Space on Device!\n");
        return;
    }

    // Creation of New Node { [?]->NULL }
    printf("Enter the Data to be stored in Node : ");
    scanf("%d", &temp->data);
    temp->next = NULL;

    if (head == NULL)
    {
        // If the Linked List is Empty
        head = temp;
    }
    else
    {
        // If the Linked List is Not Empty
        temp->next = head;
        // head is Pointing to temp NOW
        head = temp;
    }
}

// Insert at the End
void Insert_End()
{
    struct node *temp, *ptr;

    temp = (struct node *)malloc(sizeof(struct node));

    if (temp == NULL)
    {
        printf("No Memory Space on Device!\n");
        return;
    }

    // Creation of New Node { [?]->NULL }
    printf("Enter the Data to be stored in Node : ");
    scanf("%d", &temp->data);
    temp->next = NULL;

    if (head == NULL)
    {
```

```
        head = temp;
    }
    else
    {
        // Intialize ptr to head
        ptr = head;
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        // ptr will be pointing to the Last Element of Linked List
        ptr->next = temp;
    }
}

// Deletion at the Front of Linked List
void Delete_Begin()
{
    // temporary pointer to store old head
    struct node *ptr;

    if (ptr == NULL)
    {
        printf("List is Empty! No Deletion Possible!!\n");
        return;
    }
    else
    {
        ptr = head;
        // head is pointing to second element NOW
        head = head->next;
        printf("The Deleted Element : %d\n", ptr->data);
        free(ptr);
    }
}

// Deletion at the End of Linked List
void Delete_End()
{
    struct node *temp, *ptr;

    if (head == NULL)
    {
        printf("List is Empty! No Deletion Possible!!\n");
        exit(0);
    }
    else if (head->next == NULL)
    {
        // Only One Element in the Linked List
        ptr = head;
```

```
        head = NULL;
        printf("The Deleted Element is : %d\n", ptr->data);
        free(ptr);
    }
    else
    {
        ptr = head;
        while (ptr->next != NULL)
        {
            temp = ptr;
            ptr = ptr->next;
        }

        // temp is Pointing to Second Last Element
        temp->next = NULL;

        printf("The Deleted Element is : %d\n", ptr->data);
        free(ptr);
    }
}

// Insertion at Middle of Linked List
void Insert_Middle()
{
    struct node *ptr, *temp;

    int i, pos;

    temp = (struct node *)malloc(sizeof(struct node));

    if (temp == NULL)
    {
        printf("No Memory Space on Device!\n");
        return;
    }

    printf("Enter the Position for the New Node to be Inserted : ");
    scanf("%d", &pos);

    // pos = 1 -> Insertion at Beginning of LL
    // pos = len + 1 -> Insertion at Ending of LL

    // Length of the Linked Lst
    int len = LENGTH_LL();

    if (pos <= 0 || pos > len + 1)
    {
        printf("Enter Valid Postion for Insertion!\n");
        return;
    }
```

```
    // Creation of New Node { [?]->NULL }
    printf("Enter the Data to be stored in Node : ");
    scanf("%d", &temp->data);
    temp->next = NULL;

    if (pos == 1)
    {
        // At the Beginning of Linked List
        temp->next = head;
        head = temp;
    }
    else
    {
        for (i = 1, ptr = head; i < pos - 1; i++)
        {
            ptr = ptr->next;
        }
        // temp is also pointing to next of "pos" to be inserted
        temp->next = ptr->next;
        // Make ptr Point to Temp
        ptr->next = temp;
    }
}

// Deletion at Middle of Linked List
void Delete_Middle()
{
    if (head == NULL)
    {
        printf("List is Empty! No Deletion Possible!!\n");
        exit(0);
    }
    else
    {
        int ch = 0;
        printf("Delete A Node By : \n");
        printf(" 1 -> Position\n");
        printf(" 2 -> Value\n");
        printf("Enter Your Choice : ");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            Delete_Position();
            break;
        case 2:
            Delete_Value();
            break;
```

```
        default:
            printf("Enter a Valid Choice!\n");
            break;
        }
    }
}

//Deletes Node at Particular Position
void Delete_Position()
{
    int i, pos;
    struct node *temp, *ptr;

    printf("Enter the Position of the Node to be Deleted : ");
    scanf("%d", &pos);

    // pos = 1 -> Deletion at Beginning of LL
    // pos = len -> Deletion at Ending of LL

    // Length of the Linked Lst
    int len = LENGTH_LL();

    if (pos <= 0 || pos > len)
    {
        printf("Enter Valid Postion for Deletion!\n");
        return;
    }

    if (pos == 1)
    {
        ptr = head;
        head = head->next;
        printf("The Deleted Element is : %d\n", ptr->data);
        free(ptr);
    }
    else
    {
        ptr = head;
        for (i = 1; i < pos; i++)
        {
            temp = ptr;
            ptr = ptr->next;
        }
        // point the prev of {element to be deleted} to "next of deleted"
        // []     []     []
        //temp    ptr
        temp->next = ptr->next;

        printf("The Deleted Element is : %d\n", ptr->data);
        free(ptr);
```

```
    }
}

// Deletes all Nodes with Particular Value
void Delete_Value()
{
    int value;
    struct node *temp, *ptr;

    printf("Enter the Value of the Node to be Deleted : ");
    scanf("%d", &value);

    int flag = 0;

    if (head == NULL)
    {
        printf("List is Empty!No Deletions Possible\n");
        return;
    }
    else
    {
        // Head Pointer
        ptr = head;

        while (ptr != NULL)
        {

            // If the Value of Node = Value of Node to be Deleted
            if (ptr->data == value)
            {
                if (ptr == head)
                {
                    // head is pointing to second element NOW
                    head = head->next;
                    // printf("The Deleted Element : %d\n", ptr->data);
                    flag = 1;
                }
                else
                {
                    temp->next = ptr->next;
                    flag = 1;
                    // printf("The Deleted Element is : %d\n", ptr->data);
                }
            }
            // temp stored old node's address
            temp = ptr;
            // ptr now points to next node
            ptr = ptr->next;
        }
```

```
        if (flag == 0)
        {
            printf("Node with Given Value Does Not Exist! OR Deleted Earlier!\n");
        }
        else
        {
            printf("Node with Given Value Found and Deleted Succesfully!\n");
        }
    }
}

// Appends a single element at the head of the list
// [Similar to Insert at Beginning]
void list_insert_beg(node **list, node *node)
{
    // node->next should point to Starting of List
    node->next = *list;
    // Now, List Should Point to Node
    *list = node;
}

// Concanates Two Lists
void list_concat(node **list1, node *list2)
{
    // traverse complete list to get address of last node of list1
    while (*list1)
        list1 = &((*list1)->next);

    // Point the End of List 1 to Starting of List2
    *list1 = list2;
}

// QuickSort Implementation using Linked List
void My_QuickSort(node **list)
{
    // Sorting an empty list is trivial
    if (!*list)
    {
        // printf("Linked List is Empty! So Can't Sort Further!\n");
        return;
    }

    //Extract the pivot
    node *pivot = *list;
    int data = pivot->data;

    node *p = pivot->next;
    pivot->next = NULL;

    // Construct left and right lists in place in a single pass
```

```
    node *left = NULL;
    node *right = NULL;

    // Divide the List into Two Parts
    // left -> Elements Smaller than Pivot
    // Right -> Elements Larger than Pivot
    while (p)
    {
        node *n = p;
        p = p->next;
        if (n->data < data)
            list_insert_beg(&left, n);
        else
            list_insert_beg(&right, n);
    }

    //We now sort left and right
    My_QuickSort(&left);
    My_QuickSort(&right);

    // We now concatenate Left "List + Pivot + Right" List
    node *result = NULL;
    list_concat(&result, left);
    list_concat(&result, pivot);
    list_concat(&result, right);

    // Now Point it to New Sorted List
    *list = result;
}

// Binary Search Implementation on Linked List
void Binary_Search(int value)
{
    int flag = 0;
    int low = 1;
    int high = LENGTH_LL();

    // printf("Length %d", high);

    struct node *tmp = head;
    int mid;

    while (low <= high)
    {
        //To Avoid OverFlow
        mid = low + (high - low) / 2;
        tmp = head;
        for (int i = 1; i <= mid - 1; i++)
        {
            tmp = tmp->next;
```

```
        }

        if (tmp->data == value)
        {
            printf("Found at Position %d in Linked List!\n", mid);
            flag = 1;
            break;
        }
        else if (tmp->data > value)
        {
            high = mid - 1;
        }
        else if (tmp->data < value)
        {
            low = mid + 1;
        }
    }
    if (flag == 0)
    {
        printf("Not Found in Linked List!\n");
    }
}
```

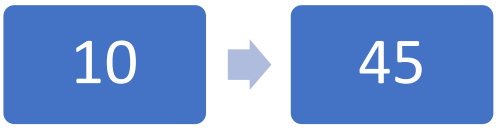
Test Cases:

A.) Creation of Linked List

45

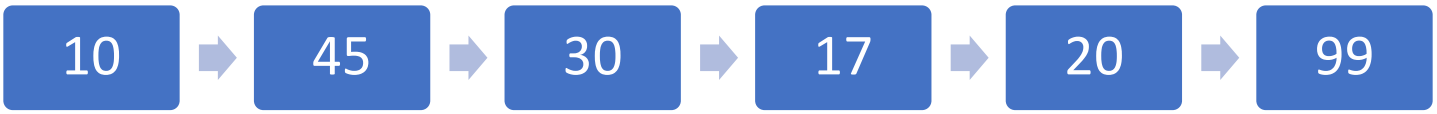
```
LINKED LIST
 1 -> Create a Linked List
 2 -> Display the Linked List
 3 -> Insert at the Beginning of Linked List
 4 -> Insert at the End of Linked List
 5 -> Insert at Middle of Linked List
 6 -> Delete from Beginning
 7 -> Delete from the End
 8 -> Delete at Middle of Linked List
 9 -> Sort the Linked List using QuickSort Algorithm
 10 -> Binary Search [Make Sure List is Sorted]
 11 -> Exit
Enter your choice : 1
Enter the Data to be stored in Node : 45
Enter your choice : 2
Elements of List : 45 -> NULL
```

B.) Insert 10 at Front of Linked List




```
Enter your choice : 3
Enter the Data to be stored in Node : 10
Enter your choice : 2
Elements of List : 10 -> 45 -> NULL
```

C.) Insert 30, 17, 20 & 99 at End of Linked List



D.) Sort the Linked List using QuickSort Algorithm

E.) Binary Search for 45 [Element Present] & 25 [Element Not Present]

```
Enter your choice : 4
Enter the Data to be stored in Node : 30
Enter your choice : 4
Enter the Data to be stored in Node : 17
Enter your choice : 4
Enter the Data to be stored in Node : 20
Enter your choice : 4
Enter the Data to be stored in Node : 99
Enter your choice : 2
Elements of List : 10 -> 45 -> 30 -> 17 -> 20 -> 99 -> NULL
Enter your choice : 9
SORTING USING QUICKSORT ALGORITHM
Elements of List : 10 -> 17 -> 20 -> 30 -> 45 -> 99 -> NULL
Enter your choice : 10
BINARY SEARCH
Enter a Value to Search in Linked List : 45
Found at Position 5 in Linked List!
Enter your choice : 10
BINARY SEARCH
Enter a Value to Search in Linked List : 25
Not Found in Linked List!
Enter your choice : 11
Enter a Valid Choice!
```