ASSIGNMENT-7

1. Write a menu driven program to perform Multiple Operations on a Linked List

SOLUTION:

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

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

struct Node* create_linked_list(struct Node* head) {
    int n, i;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    struct Node* tail = NULL;
    for (i = 0; i < n; i++) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &newNode->data);
        newNode->next = NULL;
        if (head == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}

struct Node* display_list(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return head;
    }
    printf("Linked List: ");
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("NULL\n");
    return head;
}

struct Node* add_node_at_beginning(struct Node* head) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data for the new node: ");
    scanf("%d", &newNode->data);
    newNode->next = head;
    head = newNode;
    return head;
}

struct Node* add_node_at_end(struct Node* head) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data for the new node: ");
    scanf("%d", &newNode->data);
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* current = head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
    return head;
}

struct Node* add_node_at_specified_pos(struct Node* head) {
    int pos, data;
    printf("Enter the position where you want to add a node: ");
    scanf("%d", &pos);
    if (pos < 1) {
        printf("Invalid position. Please enter a positive integer.\n");
        return head;
    }
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data for the new node: ");
    scanf("%d", &newNode->data);
    newNode->next = NULL;
    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        return head;
    }
    struct Node* current = head;
    int currentPos = 1;
    while (current != NULL && currentPos < pos) {
        current = current->next;
        currentPos++;
    }
    if (current == NULL) {
        printf("Position %d is beyond the end of the list.\n", pos);
        free(newNode);
    } else {
        newNode->next = current->next;
        current->next = newNode;
    }
    return head;
}

struct Node* delete_node_from_beginning(struct Node* head) {
    if (head == NULL) {
        printf("The list is already empty.\n");
        return head;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
    return head;
}

struct Node* delete_node_from_end(struct Node* head) {
    if (head == NULL) {
        printf("The list is already empty.\n");
        return head;
    }
    if (head->next == NULL) {
        free(head);
        head = NULL;
        return head;
    }
    struct Node* current = head;
    while (current->next->next != NULL) {
        current = current->next;
    }
    free(current->next);
    current->next = NULL;
    return head;
}

struct Node* delete_node_from_specified_pos(struct Node* head) {
    int pos;
    printf("Enter the position from which you want to delete a node: ");
    scanf("%d", &pos);
    if (pos < 1) {
        printf("Invalid position. Please enter a positive integer.\n");
        return head;
    }
    if (pos == 1) {
        if (head == NULL) {
            printf("The list is already empty.\n");
        } else {
            struct Node* temp = head;
            head = head->next;
            free(temp);
        }
        return head;
    }
    struct Node* current = head;
    int currentPos = 1;
    while (current != NULL && currentPos < pos - 1) {
        current = current->next;
        currentPos++;
    }
    if (current == NULL || current->next == NULL) {
        printf("Position %d is beyond the end of the list.\n", pos);
    } else {
        struct Node* temp = current->next;
        current->next = current->next->next;
        free(temp);
    }
    return head;
}

int count_nodes(struct Node* head) {
    int c=0;
    struct Node* current = head;
    while (current != NULL) {
        c++;
        current = current->next;
    }
    return c;
}

struct Node* sort_list(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty. Nothing to sort.\n");
        return head;
    }
    struct Node* current = head;
    struct Node* index = NULL;
    int temp;
    while (current != NULL) {
        index = current->next;
        while (index != NULL) {
            if (current->data > index->data) {
                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
            index = index->next;
        }
        current = current->next;
    }
    printf("List sorted successfully.\n");
    return head;
}

struct Node* rev(struct Node* head){
            if (head == NULL) {
        printf("The list is empty!");
        return head;
    }
            struct Node* temp=head;
            struct Node* prev=NULL;
            struct Node* nextn=NULL;
            while(temp!=NULL){
                        nextn=temp->next;
                        temp->next=prev;
                        prev=temp;
                        temp=nextn;
            }
            printf("List reversed successfully.\n");
            return prev;
}

int main() {
    int option;
    struct Node* head = NULL;
    struct Node* tail = NULL;
    do {
        printf("\nMAIN MENU:\n");
        printf("\n 1: Create a list");
        printf("\n 2: Display the list");
        printf("\n 3: Add a node at the beginning");
        printf("\n 4: Add a node at the end");
        printf("\n 5: Add a node at a Specified Position");
        printf("\n 6: Delete a node from the beginning");
        printf("\n 7: Delete a node from the end");
        printf("\n 8: Delete a node from a Specified Position");
        printf("\n 9: Node Count");
        printf("\n 10: Sort the list");
        printf("\n 11: Reverse the List");
        printf("\n 12: EXIT");
        printf("\n\n Enter your option: ");
        scanf("%d", &option);
        switch (option) {
            case 1:
                head = create_linked_list(head);
                break;
            case 2:
                head = display_list(head);
                break;
            case 3:
                head = add_node_at_beginning(head);
                break;
            case 4:
                head = add_node_at_end(head);
                break;
            case 5:
                head = add_node_at_specified_pos(head);
                break;
            case 6:
                head = delete_node_from_beginning(head);
                break;
            case 7:
                head = delete_node_from_end(head);
                break;
            case 8:
                head = delete_node_from_specified_pos(head);
                break;
            case 9:
                head = display_list(head);
                int nodeCount = count_nodes(head);
                printf("Number of nodes in the list: %d\n", nodeCount);
                break;
            case 10:
                head = sort_list(head);
                break;
            case 11:
                head = rev(head);
                break;
            case 12:
                printf("Exiting the program.\n");
                exit(0);
            default:
```

```
            printf("Invalid option. Please try again.\n");
            break;
      }
   } while (1);
   return 0;
}
```

OUTPUT:

```
MAIN MENU:

 1: Create a list
 2: Display the list
 3: Add a node at the beginning
 4: Add a node at the end
 5: Add a node at a Specified Position
 6: Delete a node from the beginning
 7: Delete a node from the end
 8: Delete a node from a Specified Position
 9: Node Count
 10: Sort the list
 11: Reverse the List
 12: EXIT

 Enter your option: 1
Enter the number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30

MAIN MENU:

 1: Create a list
 2: Display the list
 3: Add a node at the beginning
 4: Add a node at the end
 5: Add a node at a Specified Position
 6: Delete a node from the beginning
 7: Delete a node from the end
 8: Delete a node from a Specified Position
 9: Node Count
 10: Sort the list
 11: Reverse the List
 12: EXIT

 Enter your option: 2
Linked List: 10 20 30 NULL
```

```
MAIN MENU:

 1: Create a list
 2: Display the list
 3: Add a node at the beginning
 4: Add a node at the end
 5: Add a node at a Specified Position
 6: Delete a node from the beginning
 7: Delete a node from the end
 8: Delete a node from a Specified Position
 9: Node Count
 10: Sort the list
 11: Reverse the List
 12: EXIT

 Enter your option: 5
Enter the position where you want to add a node: 2
Enter data for the new node: 40

MAIN MENU:

 1: Create a list
 2: Display the list
 3: Add a node at the beginning
 4: Add a node at the end
 5: Add a node at a Specified Position
 6: Delete a node from the beginning
 7: Delete a node from the end
 8: Delete a node from a Specified Position
 9: Node Count
 10: Sort the list
 11: Reverse the List
 12: EXIT

 Enter your option: 2
Linked List: 10 20 40 30 NULL
```