DS LAB

NAME: SOUMYADIP MAITY

ROLL NO.: 22053029

SECTION: CSE 49

YEAR: 2022-23

ASSIGNMENT- 5

DATE: 08.09.2023

Q1: WAP to create a double linked list of n nodes and display the linked list by using suitable user defined functions for create and display operations.

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

// Define a structure for a double linked list node
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node at the end of the list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
}
```

```
// Function to display the double linked list
void display(struct Node* head) {
    printf("Double Linked List: ");
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int n, data;

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

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

    display(head);

    return 0;
}
```

Q2: WAP to reverse the sequence of elements in a double linked list.

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

// Define a structure for a double-linked list node
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to reverse the sequence of elements in the double-linked list
void reverse(struct Node** head) {
    struct Node* current = *head;
    struct Node* temp = NULL;

    while (current != NULL) {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }

    if (temp != NULL) {
        *head = temp->prev; // Update the new head
    }
}

// Function to display the double-linked list
void display(struct Node* head) {

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

int main() {
    // Create predefined nodes
    struct Node* head = createNode(1);
    struct Node* second = createNode(2);
    struct Node* third = createNode(3);
```

```
    // Connect the nodes to form a double-linked list
    head->next = second;
    second->prev = head;
    second->next = third;
    third->prev = second;

    // Display the original double-linked list
    printf("Original double linked list is:-\n");
    display(head);

    // Reverse the sequence of elements in the double-linked list
    reverse(&head);

    // Display the reversed double-linked list
    printf("Reversed double linked list is:-\n");
    display(head);

    return 0;
}
```

Q3: Write a menu driven program to perform the following operations in a double linked list by using suitable user defined functions for each case.
a) Traverse the list forward, b) Traverse the list backward, c) Check if the list is empty
d) Insert a node at the certain position (at beginning/end/any position)
e) Delete a node at the certain position (at beginning/end/any position)
f) Delete a node for the given key,

g) Count the total number of nodes,
h) Search for an element in the linked list
Verify & validate each function from main method

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

// Define a structure for a double-linked list node
```

```
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
}
void insertAtPosition(struct Node** head, int data, int position) {
    if (position <= 0) {
        printf("Invalid position.\n");
        return;
    }
    struct Node* newNode = createNode(data);
    if (*head == NULL && position != 1) {
        printf("List is empty, cannot insert at position %d.\n", position);
        return;
    } else if (position == 1) {
        insertAtBeginning(head, data);
    } else {
        struct Node* current = *head;
```

```
        int currentPosition = 1;
        while (currentPosition < position - 1 && current->next != NULL) {
            current = current->next;
            currentPosition++;
        }
        if (currentPosition < position - 1) {
            printf("Position %d is out of range.\n", position);
        } else {
            newNode->next = current->next;
            newNode->prev = current;
            if (current->next != NULL) {
                current->next->prev = newNode;
            }
            current->next = newNode;
        }
    }
}
void deleteAtBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty, cannot delete.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    if (*head != NULL) {
        (*head)->prev = NULL;
    }
    free(temp);
}
// Function to delete a node at the end of the list
void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty, cannot delete.\n");
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    if (current->prev != NULL) {
        current->prev->next = NULL;
    } else {
        *head = NULL;
```

```
    }
    free(current);
}

// Function to delete a node at a specific position
void deleteAtPosition(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty, cannot delete.\n");
        return;
    }
    if (position <= 0) {
        printf("Invalid position.\n");
        return;
    }
    if (position == 1) {
        deleteAtBeginning(head);
        return;
    }
    struct Node* current = *head;
    int currentPosition = 1;
    while (currentPosition < position && current->next != NULL) {
        current = current->next;
        currentPosition++;
    }
    if (currentPosition < position) {
        printf("Position %d is out of range.\n", position);
        return;
    }
    current->prev->next = current->next;
    if (current->next != NULL) {
        current->next->prev = current->prev;
    }
    free(current);
}

// Function to delete a node with a specific key
void deleteByKey(struct Node** head, int key) {
    if (*head == NULL) {
        printf("List is empty, cannot delete.\n");
        return;
    }
    struct Node* current = *head;
    while (current != NULL && current->data != key) {
```

```
        current = current->next;
    }
    if (current == NULL) {
        printf("Key not found in the list.\n");
        return;
    }
    if (current->prev != NULL) {
        current->prev->next = current->next;
    } else {
        *head = current->next;
    }
    if (current->next != NULL) {
        current->next->prev = current->prev;
    }
    free(current);
}
// Function to check if the list is empty
int isEmpty(struct Node* head) {
    return head == NULL;
}

// Function to count the total number of nodes
int countNodes(struct Node* head) {
    int count = 0;
    struct Node* current = head;
    while (current != NULL) {
        count++;
        current = current->next;
    }
    return count;
}

// Function to search for an element in the linked list
int searchElement(struct Node* head, int key) {
    struct Node* current = head;
    int position = 1;
    while (current != NULL) {
        if (current->data == key) {
            return position;
        }
        current = current->next;
        position++;
    }
```

```
    return -1; // Element not found
}

// Function to traverse the list forward
void traverseForward(struct Node* head) {
    printf("Forward Traversal: ");
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

// Function to traverse the list backward
void traverseBackward(struct Node* head) {
    printf("Backward Traversal: ");
    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->prev;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data, position, key, result;
    while (1) {
        printf("\n---------- Menu ----------\n");
        printf("1. Traverse the list forward\n");
        printf("2. Traverse the list backward\n");
        printf("3. Check if the list is empty\n");
        printf("4. Insert a node\n");
        printf("5. Delete a node\n");
        printf("6. Delete a node by key\n");
        printf("7. Count the total number of nodes\n");
        printf("8. Search for an element\n");
        printf("9. Exit\n");
        printf("Enter your choice: ");
```

```
scanf("%d", &choice);
switch (choice) {
    case 1:
        traverseForward(head);
        break;
    case 2:
        traverseBackward(head);
        break;
    case 3:
        if (isEmpty(head)) {
            printf("The list is empty.\n");
        } else {
            printf("The list is not empty.\n");
        }
        break;
    case 4:
        printf("Enter data to insert: ");
        scanf("%d", &data);
        printf("1. Insert at the beginning\n");
        printf("2. Insert at the end\n");
        printf("3. Insert at a specific position\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                insertAtBeginning(&head, data);
                break;
            case 2:
                insertAtEnd(&head, data);
                break;
            case 3:
                printf("Enter the position to insert: ");
                scanf("%d", &position);
                insertAtPosition(&head, data, position);
                break;
            default:
                printf("Invalid choice.\n");
                break;
        }
        traverseForward(head);
        break;
    case 5:
        if (isEmpty(head)) {
```

```
            printf("The list is empty, cannot delete.\n");
        } else {
            printf("1. Delete at the beginning\n");
            printf("2. Delete at the end\n");
            printf("3. Delete at a specific position\n");
            printf("Enter your choice: ");
            scanf("%d", &choice);
            switch (choice) {
                case 1:
                    deleteAtBeginning(&head);
                    break;
                case 2:
                    deleteAtEnd(&head);
                    break;
                case 3:
                    printf("Enter the position to delete: ");
                    scanf("%d", &position);
                    deleteAtPosition(&head, position);
                    break;
                default:
                    printf("Invalid choice.\n");
                    break;
            }
        }
        traverseForward(head);
        break;
    case 6:
        printf("Enter the key to delete: ");
        scanf("%d", &key);
        deleteByKey(&head, key);
        traverseForward(head);
        break;
    case 7:
        result = countNodes(head);
        printf("Total number of nodes: %d\n", result);
        break;
    case 8:
        printf("Enter the element to search: ");
        scanf("%d", &key);
        result = searchElement(head, key);
        if (result != -1) {
            printf("Element %d found at position %d.\n", key, result);
        } else {
```

```
                printf("Element %d not found in the list.\n", key);
            }
            break;
        case 9:
            exit(0);
    }
  }

  return 0;
}
```

Q4: WAP to create a circular double linked list of n nodes and display the linked list by using suitable user defined functions for create and display operations.

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

// Define a structure for a circular double-linked list node
struct Node {
  int data;
  struct Node* prev;
  struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  if (newNode == NULL) {
    printf("Memory allocation failed.\n");
    exit(1);
  }
  newNode->data = data;
  newNode->prev = NULL;
  newNode->next = NULL;
  return newNode;
}

// Function to insert a new node at the end of the list
void insertAtEnd(struct Node** head, int data) {
  struct Node* newNode = createNode(data);
  if (*head == NULL) {
    *head = newNode;
```

```
        (*head)->next = newNode;
        (*head)->prev = newNode;
    } else {
        struct Node* tail = (*head)->prev;
        tail->next = newNode;
        newNode->prev = tail;
        newNode->next = *head;
        (*head)->prev = newNode;
    }
}

// Function to display the circular double-linked list
void display(struct Node* head) {
    struct Node* current = head;
    printf("Circular Double Linked List: ");
    if (head != NULL) {
        do {
            printf("%d -> ", current->data);
            current = current->next;
        } while (current != head);
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int n, data;

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

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

    display(head);

    return 0;
}
```