Lab 7 11/12/25

⇒ Implementation of Double ended Queue Linked List

pseudo code:

```
Function (create node(int n):
    for loop till n:
        create a node (newnode);
        newnode->data = data;
        newnode->prev = newnod->next = NULL
        if (tail == NULL) head = tail = newnode;
        else
            tail->next = newnode;
            newnode->prev = tail;
            tail = newnode;
        End if
    End for
End function

Function insert at beginning(int data)
    if head is NULL
        head = tail = newnode;
    else
        head->prev = newnode;
        newnode->next = head;
        head = newnode;
    End if
End function

Function insert atEnd(int data);
    if (tail != NULL)
        tail->next = newnode;
        newnod->prev = tail;
        tail = newnode;
    End if
End function

Function delete by val (int val):
    struct temp = head;
    while (temp != NULL && temp->data != val) {
        temp = temp->next;
    }
    temp->prev->next = temp->next
    temp->next->prev = temp->prev
    free(temp)
```

```
Code:-
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *prev, *next;
};
struct node *head = NULL, *tail = NULL;
void createList(int n) {
    int data;
    for (int i=1; i<=n; i++) {
        printf("Enter node(%d) data: ", i);
        scanf("%d", &data);
        struct node *newnode = (struct node *) malloc(sizeof(struct node));
        if (newnode == NULL) {
            printf("Memory allocation failed\n");
            return;
        }
        newnode->data = data;
        newnode->next = newnode->prev = NULL;
        if (head == NULL) head = tail = newnode;
        else {
            tail->next = newnode;
            newnode->prev = tail;
            tail = newnode;
        }
    }
    printf("List created successfully\n");
}

void insertAtEnd(int val) {
    struct node *newnode = (struct node *) malloc(sizeof(struct node));
    newnode->data = val;
    newnode->next = NULL;
    if (head == NULL) {
        newnode->prev = NULL;
        head = tail = newnode;
    } else {
        newnode->prev = tail;
        tail->next = newnode;
        tail = newnode;
    }
    printf("Node added at end\n");
}
```

```
void insertAtBeginning(int val) {
    struct node *newnode = (struct node *) malloc(sizeof(struct node));
    newnode->data = val;
    newnode->prev = NULL;
    newnode->next = head;
    if (head == NULL) head = tail = newnode;
    else {
        head->prev = newnode;
        head = newnode;
    }
    printf("Node added at beginning\n");
}

void deleteByValue(int val) {
    struct node *temp = head;
    while (temp != NULL && temp->data != val) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Node not found\n"); return;
    }
    if (temp == NULL) {
        head = head->next;
        if (head) head->prev = NULL;
        else tail = NULL;
    }
    else if (temp == tail) {
        tail = tail->prev;
        tail->next = NULL;
    }
    else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }
    printf("Node with data %d deleted\n", val);
    free(temp);
}
```

```
void display() {
    struct node *temp = head;
    if(!head) {
        printf("List is empty\n");
        return;
    }
    printf("Current List :");
    while(temp != NULL) {
        printf("%d<-", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int ch, n, val;
    while(1) {
        printf("--- Double ll operation ---\n");
        printf("1. Create List\n2. Insert at Left\n3. Insert at Right\n
            4. Delete by value\n5. Display\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch(ch) {
            case 1:
                printf("Enter no. of Nodes: ");
                scanf("%d", &n);
                createList(n);
                break;
            case 2:
                printf("Enter value to Insert at Beginning: ");
                scanf("%d", &val);
                insertAtBeginning(val);
                break;
            case 3:
                printf("Enter value to Insert at End: ");
                scanf("%d", &val);
                insertAtEnd(val);
                break;
            case 4:
                printf("Enter Value to Delete: ");
                scanf("%d", &val);
                deleteByValue(val);
                break;
            case 5:
                display();
                break;
            case 6:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
}
```

```
o/p:
--Double ll operation--
1) Create List
2) Insert at left
3) Insert at Right
4) Delete by value
5) Display
6) Exit
Enter your choice: 1
Enter no. of nodes: 3
Enter node(1) data: 10
Enter node(2) data: 20
Enter node(3) data: 30
List Created Successfully
--- Double ll operations ---
1) Create List
2) Insert at left
3) Insert at Right
4) Delete by value
5) Display
6) Exit
Enter your choice: 2
Enter value to Insert at Beginning: 12
Node added at Beginning
```

```
Doubly Linked List Operations:
1. Create a node (Insert at beginning)
2. Insert a node at beginning
3. Insert a node at end
4. Delete node by value
5. Display linked list
6. Exit
Enter your choice: 1
Enter data for the node: 10
Node with data 10 inserted at the beginning.

Doubly Linked List Operations:
1. Create a node (Insert at beginning)
2. Insert a node at beginning
3. Insert a node at end
4. Delete node by value
5. Display linked list
6. Exit
Enter your choice: 2
Enter data for the node: 20
Node with data 20 inserted at the beginning.

Doubly Linked List Operations:
1. Create a node (Insert at beginning)
2. Insert a node at beginning
3. Insert a node at end
4. Delete node by value
5. Display linked list
6. Exit
Enter your choice: 3
Enter data for the node: 30
Node with data 30 inserted at the end.
```

```
1. Create a node (Insert at beginning)
2. Insert a node at beginning
3. Insert a node at end
4. Delete node by value
5. Display linked list
6. Exit
Enter your choice: 5
Doubly Linked List: 20 10 30

Doubly Linked List Operations:
1. Create a node (Insert at beginning)
2. Insert a node at beginning
3. Insert a node at end
4. Delete node by value
5. Display linked list
6. Exit
Enter your choice: 4
Enter value to delete: 10
Node with data 10 deleted.

Doubly Linked List Operations:
1. Create a node (Insert at beginning)
2. Insert a node at beginning
3. Insert a node at end
4. Delete node by value
5. Display linked list
6. Exit
Enter your choice: 5
Doubly Linked List: 20 30

Doubly Linked List Operations:
1. Create a node (Insert at beginning)
2. Insert a node at beginning
3. Insert a node at end
4. Delete node by value
5. Display linked list
6. Exit
Enter your choice: 6
Exiting...
```

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

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

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 insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }
    printf("Node with data %d inserted at the beginning.\n", data);
}
```

```c
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
    printf("Node with data %d inserted at the end.\n", data);
}

void deleteByValue(struct Node** head, int value) {
    if (*head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return;
    }
    struct Node* temp = *head;
    if (temp->data == value) {
        *head = temp->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        free(temp);
        printf("Node with data %d deleted.\n", value);
        return;
    }
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
```

```
64      while (temp != NULL && temp->data != value) {
67      if (temp == NULL) {
68          printf("Node with data %d not found.\n", value);
69          return;
70      }
71      if (temp->next != NULL) {
72          temp->next->prev = temp->prev;
73      }
74      if (temp->prev != NULL) {
75          temp->prev->next = temp->next;
76      }
77      free(temp);
78      printf("Node with data %d deleted.\n", value);
79  }
80
81  void displayList(struct Node* head) {
82      if (head == NULL) {
83          printf("List is empty.\n");
84          return;
85      }
86      struct Node* temp = head;
87      printf("Doubly Linked List: ");
88      while (temp != NULL) {
89          printf("%d ", temp->data);
90          temp = temp->next;
91      }
92      printf("\n");
93  }
94
95  int main() {
96      struct Node* head = NULL;
97      int choice, data;
```

```
        printf("\nDoubly Linked List Operations:\n");
        printf("1. Create a node (Insert at beginning)\n");
        printf("2. Insert a node at beginning\n");
        printf("3. Insert a node at end\n");
        printf("4. Delete node by value\n");
        printf("5. Display linked list\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data for the node: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter data for the node: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 3:
                printf("Enter data for the node: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 4:
                printf("Enter value to delete: ");
                scanf("%d", &data);
                deleteByValue(&head, data);
                break;
```

```
                break;
            case 5:
                displayList(head);
                break;
            case 6:
                printf("Exiting...\n");
                while (head != NULL) {
                    struct Node* temp = head;
                    head = head->next;
                    free(temp);
                }
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}

```