DATA STRUCTURE

DAY - 2 , 25/07/2024 CSA - 0390

1) WRITE C PROGRAM FOR LINKED LIST SINGLY.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}
void deleteNode(struct Node** head, int key) {
    struct Node* temp = *head;
    struct Node* prev = NULL;
    if (temp != NULL && temp->data == key) {
        *head = temp->next;
        free(temp);
        return;
    }
    while (temp != NULL && temp->data != key) {
```

```
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) return;
    prev->next = temp->next;
    free(temp);
}
void displayList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node* head = NULL;
    insertAtEnd(&head, 10);
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    displayList(head);
    insertAtBeginning(&head, 5);
    displayList(head);
    deleteNode(&head, 20);
    displayList(head);
    return 0;
}
```

OUTPUT :

```
10 -> 20 -> 30 -> NULL
5 -> 10 -> 20 -> 30 -> NULL
5 -> 10 -> 30 -> NULL
```

2) WRITE A C PROGRAM FOR DOUBLE N CIRCULAR.

```
#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));
```

```
    newNode->data = data;
    newNode->prev = newNode->next = newNode;
    return newNode;
}
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* last = (*head)->prev;
        newNode->next = *head;
        newNode->prev = last;
        last->next = (*head)->prev = newNode;
        *head = newNode;
    }
}
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* last = (*head)->prev;
        newNode->next = *head;
        newNode->prev = last;
        last->next = (*head)->prev = newNode;
    }
}
void insertAtPosition(struct Node** head, int data, int position) {
    if (position == 0) {
        insertAtBeginning(head, data);
        return;
    }
    struct Node* newNode = createNode(data);
    struct Node* temp = *head;
    for (int i = 0; i < position - 1; i++) {
        temp = temp->next;
        if (temp == *head) {
            printf("Position out of bounds\n");
            return;
        }
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    temp->next->prev = newNode;
    temp->next = newNode;
}
void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
```

```
    struct Node* temp = *head;
    if ((*head)->next == *head) {
        *head = NULL;
    } else {
        struct Node* last = (*head)->prev;
        *head = (*head)->next;
        (*head)->prev = last;
        last->next = *head;
    }
    free(temp);
}
void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* last = (*head)->prev;
    if ((*head)->next == *head) {
        *head = NULL;
    } else {
        struct Node* prev = last->prev;
        prev->next = *head;
        (*head)->prev = prev;
    }
    free(last);
}
void deleteFromPosition(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;
    for (int i = 0; i < position; i++) {
        temp = temp->next;
        if (temp == *head) {
            printf("Position out of bounds\n");
            return;
        }
    }
    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;
    if (temp == *head) {
        *head = temp->next;
    }
    free(temp);
}
int search(struct Node* head, int data) {
    struct Node* temp = head;
    int position = 0;
    do {
        if (temp->data == data) {
```

```
            return position;
        }
        temp = temp->next;
        position++;
    } while (temp != head);
    return -1; // Data not found
}

0
void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(head)\n");
}

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

    insertAtBeginning(&head, 10);
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    insertAtPosition(&head, 25, 2);
    display(head); // 10 -> 20 -> 25 -> 30 -> (head)

    deleteFromBeginning(&head);
    deleteFromEnd(&head);
    deleteFromPosition(&head, 1);
    display(head); // 20 -> (head)

    int position = search(head, 20);
    if (position != -1) {
        printf("Element found at position: %d\n", position);
    } else {
        printf("Element not found\n");
    }

    return 0;
}
```

OUTPUT :

```
10 -> 20 -> 25 -> 30 -> (head)
```

20 -> (head)
Element found at position: 0