CSLL

Circular Singly Linked List

→Without head node

→ With head node

CSLL without head node

1. Insertion at the beginning:

```
#include <stdio.h>
#include <stdlib.h>
// Structure of a linked list node
struct node {
            int info;
            struct node* next;
};
// Pointer to last node in the list
struct node* last = NULL;
```

```
// Function to insert a node in the starting of the list
void insertAtFront(int data)
{
	// Initialize a new node
	struct node* temp;
	temp = (struct node*)malloc(sizeof(struct node));
	// If the new node is the only node in the list
	if (last == NULL) {
		temp->info = data;
		temp->next = temp;
		last = temp;
	}

	// Else last node contains the reference of the new node and new node contains the reference of the previous first node
	else {
		temp->info = data;
		temp->next = last->next;
		last->next = temp;			// last node now has reference of the new node temp
	}
}
```

2. Insertion at the end:

```
void addatlast(int data)
{
        // Initialize a new node
        struct node* temp;
        temp = (struct node*)malloc(sizeof(struct node));

        if (last == NULL) {                    // If the new node is the only node in the list
                temp->info = data;
                temp->next = temp;
                last = temp;
        }

        // Else the new node will be the last node and will contain the reference of first node
        else {
                temp->info = data;
                temp->next = last->next;
                last->next = temp;
                last = temp;
        }
}
```

3. Insertion after a specific element:

```
void insertafter()
{
        int data, value,flag=0;
        struct node *temp, *n;
        printf("\nEnter number after which you want to enter number: \n");        scanf("%d", &value);
        temp = last->next; //first
        do {        if (temp->info == value) {
                        flag=1;
                        n = (struct node*)malloc(sizeof(struct node));
                        printf("\nEnter data to be inserted : \n");                scanf("%d", &data);
                        n->info = data;
                        n->next = temp->next;
                        temp->next = n;
                        if (temp == last)            last = n;
                        break;
                }
                else
                        temp = temp->next;
        } while (temp != last->next);
        If(flag==0) printf(“%d is not present in list”,value);
}
```

4. Delete the first element:

```
void deletefirst()
{
        struct node* temp;

        // If list is empty
        if (last == NULL)
                printf("\nList is empty.\n");

        // Else last node now contains
        // reference of the second node
        // in the list because the
        // list is circular
        else {
                temp = last->next;
                last->next = temp->next;
                free(temp);
        }
}
```

5. Delete the last element:

```
void deletelast()
{
        struct node* temp;

        // If list is empty
        if (last == NULL)
                printf("\nList is empty.\n");

        temp = last->next;

        // Traverse the list till the second last node
        while (temp->next != last) temp = temp->next;

        // Second last node now contains the reference of the first node in the list
        temp->next = last->next;
        last = temp;
}
```

6. Delete at a given position:

```
void deleteAtIndex()
{		int pos, i = 1;
		struct node *temp, *position;
		temp = last->next;
		if (last == NULL) printf("\nList is empty.\n");
		else {
				// Input position
				printf("\nEnter index : ");		scanf("%d", &pos);
				// Traverse till the node to be deleted is reached
				while (i <= pos - 1) {
						temp = temp->next;
						i++;
				}
				// After the loop ends, temp points at a node just before the node to be deleted
				// Reassigning links
				position = temp->next;
				temp->next = position->next;
				free(position);
		}
}
```

```
// Function to print the list
void viewList()
{
            // If list is empty
            if (last == NULL)
                        printf("\nList is empty\n");

            // Else print the list
            else {
                        struct node* temp;
                        temp = last->next;
                        do {
                                    printf("\nData = %d", temp->info);
                                    temp = temp->next;
                        } while (temp != last->next);
            }
}
```

CSLL with head node