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));

	if (newNode == NULL) {

		printf("Memory allocation failed!\n");

		return NULL;

	}

	newNode->data = data;

	newNode->next = NULL;

	return newNode;

}

void insertAtBeginning(struct Node **head, int data) {

	struct Node *newNode = createNode(data);

	if (newNode == NULL) {

		return;	}

	newNode->next = *head;

	*head = newNode;

	printf("Inserted %d at the beginning.\n", data);
```

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

	while (temp != NULL && temp->data != key) {
		prev = temp;
		temp = temp->next;
```

```
    }

    if (temp == NULL) {
        printf("Key %d not found in the list. Deletion failed.\n", key);
        return;
    }

    if (prev == NULL) {
        *head = temp->next;
    } else {
        prev->next = temp->next;
    }

    printf("Deleted %d from the list.\n", key);
    free(temp);
}
void printList(struct Node *head) {
    struct Node *temp = head;
    if (temp == NULL) {
        printf("Linked list is empty.\n");
        return;
    }
    printf("Linked list: ");
    while (temp != NULL) {
```

```
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
}
void freeList(struct Node **head) {
        struct Node *current = *head;
        struct Node *next;

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

        *head = NULL;
}
int main() {
        struct Node *head = NULL;
        insertAtEnd(&head, 1);
        insertAtEnd(&head, 2);
        insertAtEnd(&head, 3);
        insertAtBeginning(&head, 4);
        printList(head);
```

```
    deleteNode(&head, 3);
    printList(head);
    freeList(&head);

    return 0;
}
```

Output:

```
inserted 1 at the end.
inserted 2 at the end.
inserted 3 at the end.
inserted 4 at the beginning.
linked list: 4->1->2->3->NULL
deleted 3 from the list.
linked list:4->1->2->NULL
```

Double Linked List:

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

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

    if (temp == NULL) {
```

```
        printf("Key %d not found in the list. Deletion failed.\n", key);
        return;
    }

    if (temp->prev == NULL) {
        *head = temp->next;
    } else {
        temp->prev->next = temp->next;
    }

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

    printf("Deleted %d from the list.\n", key);
    free(temp);
}
void printForward(struct Node *head) {
    struct Node *temp = head;
    if (temp == NULL) {
        printf("Doubly linked list is empty.\n");
        return;
    }
    printf("Doubly linked list (forward): ");
```

```
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
void printBackward(struct Node *head) {
    struct Node *temp = head;
    if (temp == NULL) {
        printf("Doubly linked list is empty.\n");
        return;
    }
    while (temp->next != NULL) {
        temp = temp->next;
    }
    printf("Doubly linked list (backward): ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->prev;
    }
    printf("NULL\n");
}
void freeList(struct Node **head) {
    struct Node *current = *head;
```

```
    struct Node *next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
    *head = NULL;
}
int main() {
    struct Node *head = NULL;
    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);
    insertAtBeginning(&head, 4);
    printForward(head);
    printBackward(head);
    deleteNode(&head, 3);
    printForward(head);
    printBackward(head);
    freeList(&head);
    return 0;
}
```

Output:

```
Inserted 1 at the end.
```

Inserted 2 at the end.

Inserted 3 at the end.

Inserted 4 at the beginning.Inserted 3 at the end.

Doubly linked list (forward): 4 -> 1 -> 2 -> 3 -> NULL

Doubly linked list (backward): 3 -> 2 -> 1 -> 4 -> NULL

deleted 3 from linked list.

Doubly linked list (forward): 4 -> 1 -> 2 -> NULL

Doubly linked list (backward): 2 -> 1 -> 4 -> NULL

Circular linked list:

```
#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));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return NULL;
    }
    newNode->data = data;
    newNode->next = NULL;
```

```
    return newNode;
}
void insertAtBeginning(struct Node **head, int data) {
    struct Node *newNode = createNode(data);
    if (newNode == NULL) {
        return; // Exit if memory allocation failed
    }
    if (*head == NULL) {
        *head = newNode;
        newNode->next = *head; // Point to itself to form circular link
    } else {
        struct Node *temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;
        *head = newNode;
    }
    printf("Inserted %d at the beginning.\n", data);
}
void insertAtEnd(struct Node **head, int data) {
    struct Node *newNode = createNode(data);
    if (newNode == NULL) {
```

```
        return; // Exit if memory allocation failed
    }
    if (*head == NULL) {
        *head = newNode;
        newNode->next = *head; // Point to itself to form circular link
    } else {
        struct Node *temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;
    }
    printf("Inserted %d at the end.\n", data);
}
void deleteNode(struct Node **head, int key) {
    if (*head == NULL) {
        printf("Circular linked list is empty. Deletion failed.\n");
        return;
    }

    struct Node *temp = *head, *prev = NULL;
    while (temp->data != key) {
        if (temp->next == *head) {
```

```
            printf("Key %d not found in the list. Deletion failed.\n", key);
            return;
        }
        prev = temp;
        temp = temp->next;
    }
    if (temp->next == *head && prev == NULL) {
        *head = NULL;
    } else if (temp == *head) { // If the node to be deleted is the head node
        prev = *head;
        while (prev->next != *head) {
            prev = prev->next;
        }
        *head = (*head)->next;
        prev->next = *head;
    } else if (temp->next == *head) { // If the node to be deleted is the last node
        prev->next = *head;
    } else { // If the node to be deleted is in between
        prev->next = temp->next;
    }
    free(temp);
    printf("Deleted %d from the list.\n", key);
}
void printList(struct Node *head) {
```

```
    struct Node *temp = head;
    if (temp == NULL) {
        printf("Circular linked list is empty.\n");
        return;
    }
    printf("Circular linked list: ");
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(back to the beginning)\n");
}
void freeList(struct Node **head) {
    if (*head == NULL) {
        return;
    }
    struct Node *current = *head;
    struct Node *temp;

    do {
        temp = current;
        current = current->next;
        free(temp);
    } while (current != *head);
```

```
    *head = NULL;
}
int main() {
    struct Node *head = NULL;
    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);
    insertAtBeginning(&head, 4);
    printList(head);
    deleteNode(&head, 3);
    printList(head);
    freeList(&head);
    return 0;
}
```

Output:

```
Inserted 1 at the end.
Inserted 2 at the end.
Inserted 3 at the end.
Inserted 4 at the beginning.
Circular linked list: 4 -> 1 -> 2 -> 3 -> (back to the beginning)
Deleted 3 from the list.
Circular linked list: 4 -> 1 -> 2 -> (back to the beginning)
```