**Aim ↬ To perform basic operations on Linked List.**

**Objectives ↬**

i. Write a program to print the given Linked List in Reverse Order
ii. Write a program to sort the given Linked List.
iii. Write a program to create a circularly linked list of integers entered by user and insert a new node:
    a. At the beginning
    b. At the end.

**Software Required ↬** Visual Studio Code

**Code 1 ↬**

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
   int data;
   struct Node* next;
};
typedef struct Node Node;
void reverse(struct Node** head){
   Node *curr = *head;
   Node *back = NULL, *front = NULL;
   while(curr!=NULL) {
      front = curr->next;
      curr->next = back;
      back = curr;
      curr = front;
   }
   *head = back;
}
void printList(struct Node* head){
   struct Node* temp = head;
   while(temp!=NULL){
      printf("%d ",temp->data);
      temp = temp->next;
   }
}
```

```
int main(){
    struct Node* head = NULL;
    int num;
    printf("Enter Integers: [Enter alphabet to exit]\n");
    while(scanf("%d",&num)==1){
        Node* newNode = (Node*)malloc(sizeof(Node));
        if(newNode==NULL){
            printf("Memory Allocation Failed!");
            return 1;
        }
        newNode->data=num;
        newNode->next=head;
        head = newNode;
    }
    printf("Linked List Elements: ");
    Node* current = head;
    while(current != NULL){
        printf("%d ",current->data);
        current = current->next;
    }
    printf("\n");
    current = head;
    while(current=NULL){
        Node* temp = current;
        current = current->next;
        free(temp);
    }
    reverse(&head);
    printf("Reverse Linked List: ");
    printList(head);
    return 0;
}
```

**Output ↬**

```
Enter Integers: [Enter alphabet to exit]
1
2
3
4
5
p
Linked List Elements: 5 4 3 2 1
Reverse Linked List: 1 2 3 4 5
```

**Code 2 ↬**

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
   int data;
   struct Node* next;
};

typedef struct Node Node;

void sort(struct Node** head){
   Node *curr = *head;
   Node *index = NULL;
   int temp;
   while(curr!=NULL){
      index = curr->next;
      while(index!=NULL){
         if(curr->data > index->data){
            temp = curr->data;
            curr->data = index->data;
            index->data = temp;
```

```
            }
            index = index->next;
        }
        curr = curr->next;
    }
}

void printList(struct Node* head){
    struct Node* temp = head;
    while(temp!=NULL){
        printf("%d ",temp->data);
        temp = temp->next;
    }
}

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

    printf("Enter Integers: [Enter alphabet to exit]\n");
    while(scanf("%d",&num)==1){
        Node* newNode = (Node*)malloc(sizeof(Node));
        if(newNode==NULL){
            printf("Memory Allocation Failed!");
            return 1;
        }
        newNode->data=num;
        newNode->next=head;
        head = newNode;
    }
    printf("Linked List Elements: ");
    Node* current = head;
    while(current != NULL){
        printf("%d ",current->data);
        current = current->next;
    }
    printf("\n");
```

```
    current = head;

    while(current=NULL){
        Node* temp = current;
        current = current->next;
        free(temp);
    }

    sort(&head);
    printf("Sorted Linked List: ");
    printList(head);
    return 0;
}
```

**Output ↬**

```
Enter Integers: [Enter alphabet to exit]
5
1
2
-6
3
-4
t
Linked List Elements: -4 3 -6 2 1 5
Sorted Linked List: -6 -4 1 2 3 5
```

**Code 3 ↬**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node* next;
};
struct Node* createNode(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
if (newNode == NULL) {
printf("Memory allocation failed.\n");
exit(1);
}
newNode->data = value;
newNode->next = NULL;
return newNode;
}
struct Node* insertAtBeginning(struct Node* head, int value) {
struct Node* newNode = createNode(value);
if (head == NULL) {
newNode->next = newNode; // Circular link to itself
return newNode;
}
newNode->next = head->next;
head->next = newNode;
return head;
}
struct Node* insertAtEnd(struct Node* head, int value) {
struct Node* newNode = createNode(value);
if (head == NULL) {
newNode->next = newNode; // Circular link to itself
return newNode;
}
newNode->next = head->next;
head->next = newNode;
return newNode;
}
void displayList(struct Node* head) {
if (head == NULL) {
printf("List is empty.\n");
return;
}
struct Node* current = head->next;
do {
printf("%d -> ", current->data);
current = current->next;
```

```
} while (current != head->next);
printf("(head)\n");
}
int main() {
struct Node* head = NULL;
int choice, value;
do {
printf("\nCircular Linked List Operations:\n");
printf("1. Insert at the beginning\n");

printf("2. Insert at the end\n");
printf("3. Display the list\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter the value to insert at the beginning: ");
scanf("%d", &value);
head = insertAtBeginning(head, value);
break;
case 2:
printf("Enter the value to insert at the end: ");
scanf("%d", &value);
head = insertAtEnd(head, value);
break;
case 3:
displayList(head);
break;
case 4:
printf("Exiting the program.\n");
break;
default:
printf("Invalid choice. Please try again.\n"); }
} while (choice != 4);
return 0;
}
```

## Output ↬

```
Circular Linked List Operations:
1. Insert at the beginning
2. Insert at the end
3. Display the list
4. Exit
Enter your choice: 1
Enter the value to insert at the beginning: -8

Circular Linked List Operations:
1. Insert at the beginning
2. Insert at the end
3. Display the list
4. Exit
Enter your choice: 1
Enter the value to insert at the beginning: 7

Circular Linked List Operations:
1. Insert at the beginning
2. Insert at the end
3. Display the list
4. Exit
Enter your choice: 2
Enter the value to insert at the end: -5

Circular Linked List Operations:
1. Insert at the beginning
2. Insert at the end
3. Display the list
4. Exit
Enter your choice: 3
7 -> -8 -> -5 -> (head)

Circular Linked List Operations:
1. Insert at the beginning
2. Insert at the end
3. Display the list
4. Exit
```

## Result ↬

The programs successfully managed linked list operations, including reversing and sorting a linked list, and creating a circular linked list with insertions. They effectively demonstrated:

- **Reversal:** Displayed the linked list elements in reverse order.
- **Sorting:** Sorted the linked list elements in ascending order.
- **Circular Linked List Operations:** Inserted nodes at the beginning and end of a circular linked list and displayed its contents.

## Conclusion ↬

The experiment effectively showcased fundamental linked list operations in C, including reversal, sorting, and circular linked list manipulations. The programs illustrated proper techniques for managing linked lists, improving understanding of dynamic memory management, and list manipulations.

## Precautions ↬

- Provide valid inputs to prevent errors.
- Properly manage memory allocation and deallocation.
- Handle empty list cases for all operations.
- Implement error handling for missing nodes or keys.