Program - 7

MERGE TWO SORTED LINKED LISTS

Algorithm Explanation:

1. The algorithm for merging two sorted linked lists is relatively straightforward:

2. Start with two sorted linked lists, list1 and list2.

3. Initialize a new empty linked list, which we'll call result.

4. Compare the first elements of list1 and list2. The smaller element becomes the first element of result.

5. Move to the next element in the list from which we took the smaller element (either list1 or list2), and repeat the comparison.

6. Continue this process until you have merged all the elements from both lists into result.

7. The result will be a single sorted linked list containing all the elements from list1 and list2.

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

These lines include some standard C libraries that we'll use in our program.

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

Here, we define a structure called Node that represents a single element (node) in our linked list. It has two parts: data, which stores some information, and next, which points to the next node in the list

```
struct Node* mergeSortedLists(struct Node* list1, struct Node* list2) {
    if (list1 == NULL)
        return list2;
    if (list2 == NULL)
        return list1;

    struct Node* result = NULL;

    if (list1->data <= list2->data) {
        result = list1;
        result->next = mergeSortedLists(list1->next, list2);
    } else {
        result = list2;
        result->next = mergeSortedLists(list1, list2->next);
    }

    return result;
}
```

The mergeSortedLists function takes two sorted linked lists as input (list1 and list2) and merges them into a single sorted linked list. It uses a recursive approach to do this.

- *If either list1 or list2 is NULL, it returns the other list because merging with a NULL list doesn't change anything.*
- *It compares the values of the first nodes in list1 and list2. Whichever node has a smaller value becomes the first node in the merged list (result).*
- *Then, it recursively calls mergeSortedLists to merge the rest of the lists, moving to the next nodes of the list with the smaller value while keeping the other list the same.*
- *This process continues until both list1 and list2 have been completely merged.*
- *Finally, it returns the merged list.*

```
void push(struct Node** headRef, int newData) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = newData;
    newNode->next = *headRef;
    *headRef = newNode;
}
```

The push function is used to insert a new node with the given newData at the beginning of a linked list. It takes a pointer to the head of the list (headRef) and updates it to point to the new node.

```
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
    printf("\n");
}
```

The printList function is used to print the elements of a linked list. It traverses the list from the head to the end, printing each node's data.

```
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    // Populate list1
    push(&list1, 9);
    push(&list1, 7);
    push(&list1, 5);

    // Populate list2
    push(&list2, 8);
    push(&list2, 6);
    push(&list2, 4);

    printf("List 1: ");
    printList(list1);

    printf("List 2: ");
    printList(list2);

    struct Node* mergedList = mergeSortedLists(list1, list2);

    printf("Merged List: ");
    printList(mergedList);

    return 0;
}
```

In the main function:

- *We create two sorted linked lists, list1 and list2, by using the push function to insert elements at the beginning of each list.*
- *We print the original lists.*
- *We call the mergeSortedLists function to merge list1 and list2 into mergedList.*
- *Finally, we print the merged list.*