**Experiment - 5**

**Aim:**

Merged sort

**PROBLEM:** Write a program that takes two sorted lists as inputs and merge them into one sorted list.

For example, if the first linked list A is 5 =>10 =>15, and the other linked list B is

2 =>3 => 20, then output should be 2 => 3 => 5 => 10 => 15 => 20.

# SOURCE CODE:

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* next;

};

struct Node\* newNode(int data) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

struct Node\* MergeLists(struct Node\* listA, struct Node\* listB) {

    struct Node\* mergedList = NULL;

    struct Node\* tail = NULL;

    while (listA != NULL && listB != NULL) {

        if(listA->data < listB->data) {

            if (mergedList == NULL) {

                mergedList = tail = listA;

            } else {

                tail->next = listA;

                tail = listA;

            }

            listA = listA->next;

        } else {

            if (mergedList == NULL) {

                mergedList = tail = listB;

            } else {

                tail->next = listB;

                tail = listB;

            }

            listB = listB->next;

        }

    }

    if (listA != NULL) {

        tail->next = listA;

    } else {

        tail->next = listB;

    }

    return mergedList;

}

void printList(struct Node\* list) {

    while (list != NULL) {

        printf("%d -> ", list->data);

        list = list->next;

    }

    printf("NULL\n");

}

int main() {

    struct Node\* list1 = newNode(5);

    list1->next = newNode(10);

    list1->next->next = newNode(15);

    struct Node\* list2 = newNode(2);

    list2->next = newNode(3);

    list2->next->next = newNode(20);

    printf("List 1: ");

    printList(list1);

    printf("List 2: ");

    printList(list2);

    struct Node\* mergedList = MergeLists(list1, list2);

    printf("Merged List: ");

    printList(mergedList);

    return 0;

}

# OUTPUT:

List 1: 5 -> 10 -> 15 -> NULL

List 2: 2 -> 3 -> 20 -> NULL

Merged List: 2 -> 3 -> 5 -> 10 -> 15 -> 20 -> NULL

Question-2

**PROBLEM:** . Write a program to insert a new node into the linked list. A node can be added into the

linked list using three ways: [Write code for all the three ways.]

a. At the front of the list

b. After a given node

c. At the end of the list.

# SOURCE CODE:

#include<stdio.h>

#include<stdlib.h>

struct Node{

    int data;

    struct Node \* next;

};

void linkedListTraversal(struct Node \*ptr)

{

    while (ptr != NULL)

    {

        printf("Element: %d\n", ptr->data);

        ptr = ptr->next;

    }

}

struct Node \* insertAtFirst(struct Node \*head, int data){

    struct Node \* ptr = (struct Node \*) malloc(sizeof(struct Node));

    ptr->data = data;

    ptr->next = head;

    return ptr;

}

struct Node \* insertAtEnd(struct Node \*head, int data){

    struct Node \* ptr = (struct Node \*) malloc(sizeof(struct Node));

    ptr->data = data;

    struct Node \* p = head;

    while(p->next!=NULL){

        p = p->next;

    }

    p->next = ptr;

    ptr->next = NULL;

    return head;

}

struct Node \* insertAfterNode(struct Node \*head, struct Node \*prevNode, int data){

    struct Node \* ptr = (struct Node \*) malloc(sizeof(struct Node));

    ptr->data = data;

    ptr->next = prevNode->next;

    prevNode->next = ptr;

    return head;

}

int main(){

    struct Node \*head;

    struct Node \*second;

    struct Node \*third;

    struct Node \*fourth;

    head = (struct Node \*)malloc(sizeof(struct Node));

    second = (struct Node \*)malloc(sizeof(struct Node));

    third = (struct Node \*)malloc(sizeof(struct Node));

    fourth = (struct Node \*)malloc(sizeof(struct Node));

    head->data = 7;

    head->next = second;

    second->data = 11;

    second->next = third;

    third->data = 41;

    third->next = fourth;

    fourth->data = 66;

    fourth->next = NULL;

    printf("Linked list before insertion\n");

    linkedListTraversal(head);

    head = insertAtFirst(head, 56);

    head = insertAtEnd(head, 56);

    head = insertAfterNode(head, third, 45);

    printf("\nLinked list after insertion\n");

    linkedListTraversal(head);

    return 0;

}

# OUTPUT:

# 

Linked list before insertion

Element: 7

Element: 11

Element: 41

Element: 66

Linked list after insertion

Element: 56

Element: 7

Element: 11

Element: 41

Element: 45

Element: 66

Element: 56

Question-3

**PROBLEM:**  Write a program to delete a node from the linked list. A node can be deleted from the linked list using three ways: [Write code for all the three ways.]

a. Delete from the beginning

b. Delete from the end

c. Delete from the middle

SOURCE CODE:

#include<stdio.h>

#include<stdlib.h>

struct node{

    int data;

    struct node \* next;

};

void traversal(struct node\*ptr)

{

    while (ptr!=NULL){

         printf("element:%d\n",ptr->data);

         ptr= ptr->next;

    }

}

struct node \* deletatbeginnig(struct node \*head){

   struct node\* ptr = head;

   head=head->next;

   free(ptr);

   return head;

}

struct node \* deletatlast(struct node \*head){

    struct node\*ptr= head;

    struct node \*ptr2=head->next;

    while(ptr2->next != NULL){

        ptr=ptr->next;

        ptr2=ptr2->next;

    }

    ptr->next=NULL;

    free(ptr2);

    return head;

}

struct node \* deleteByValue(struct node \* head, int value){

    struct node \*ptr = head;

    struct node \*ptr2= head->next;

    while(ptr2->data!=value && ptr2->next!= NULL)

    {

        ptr = ptr->next;

        ptr2 = ptr2->next;

    }

    if(ptr2->data == value){

        ptr->next = ptr2->next;

        free(ptr2);

    }

    return head;

}

int main()

{

    struct node \*head;

    struct node \*sec;

    struct node \*third;

    struct node \*fourth;

    head = (struct node \*)malloc(sizeof(struct node));

    sec = (struct node \*)malloc(sizeof(struct node));

    third = (struct node \*)malloc(sizeof(struct node));

    fourth = (struct node \*)malloc(sizeof(struct node));

    head->data = 3;

    head->next = sec;

    sec->data = 6;

    sec->next = third;

    third->data = 8;

    third->next = fourth;

    fourth->data = 2;

    fourth->next = NULL;

    printf("Linked list before deletion\n");

    traversal(head);

    head = deletatbeginnig(head); // For deleting first element of the linked list

    head = deleteatindex(head, 2);

    printf("Linked list after deletion\n");

    traversal(head);

    return 0;

}

# OUTPUT:

# 

Linked list before deletion

element:3

element:6

element:8

element:2

Linked list after deletion

element:6

element:8

question-4

**PROBLEM:**

Implement the circular linked list and perform the operation of traversal on it. In a conventional linked list, we traverse the list from the head node and stop the traversal when we reach NULL. In a circular linked list, we stop traversal when we reach the first node again.

SOURCE CODE:

#include<stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

void linkedListTraversal(struct Node \*head){

    struct Node \*ptr = head;

    do{

        printf("Element is %d\n", ptr->data);

        ptr = ptr->next;

    }while(ptr!=head);

}

struct Node \* insertAtFirst(struct Node \*head, int data){

    struct Node \* ptr = (struct Node \*) malloc(sizeof(struct Node));

    ptr->data = data;

    struct Node \* p = head->next;

    while(p->next != head){

        p = p->next;

    }

    p->next = ptr;

    ptr->next = head;

    head = ptr;

    return head;

}

int main(){

    struct Node \*head;

    struct Node \*second;

    struct Node \*third;

    struct Node \*fourth;

    head = (struct Node \*)malloc(sizeof(struct Node));

    second = (struct Node \*)malloc(sizeof(struct Node));

    third = (struct Node \*)malloc(sizeof(struct Node));

    fourth = (struct Node \*)malloc(sizeof(struct Node));

    head->data = 2;

    head->next = second;

    second->data = 3;

    second->next = third;

    third->data = 1;

    third->next = fourth;

    fourth->data = 5;

    fourth->next = head;

    printf("Circular Linked list before insertion\n");

    linkedListTraversal(head);

    head = insertAtFirst(head, 6);

    head = insertAtFirst(head, 8);

    head = insertAtFirst(head, 9);

    printf("Circular Linked list after insertion\n");

    linkedListTraversal(head);

    return 0;

}

# OUTPUT:

# 

Circular Linked list before insertion

Element is 2

Element is 3

Element is 1

Element is 5

Circular Linked list after insertion

Element is 9

Element is 8

Element is 6

Element is 2

Element is 3

Element is 1

Element is 5