LAB-6: (25/1/24)

a) Linked List operations:

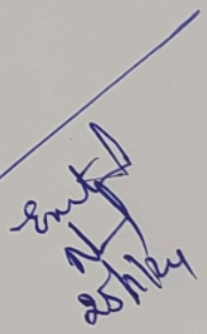
```
#include <stdio.h>
#include <stdlib.h>

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

Node *createNode(int value)
{
  Node *newNode = (Node*) malloc (sizeof(Node));
  newNode -> data = value;
  newNode -> next = NULL;
  return newNode;
}

void displayList(Node *head)
{
  while(head != NULL)
  {
    printf("%d -> ", head -> data);
    head = head -> next;
  }
  printf("NULL \n");
}

Node *sortList(Node *head)
{
  if(head == NULL || head -> next == NULL)
  {
    return head;
  }
```



```
    int swapped;
    Node *temp;
    Node * end = NULL;

    do
    {
      swapped = 0;
      temp = head;
      while (temp -> next != end)
      {
        if (temp -> data > temp -> next -> data)
        {
          int tempData = temp -> data;
          temp -> data = temp -> next -> data;
          temp -> next -> data = tempData;

          swapped = 1;
        }
        temp = temp -> next;
      }
    end = temp;
  } while (swapped);

  return head;

}

Node *reverseList (Node *head)
{
  Node *prev = NULL;
  Node *current = head;
  Node *nextNode = NULL;

  while (current != NULL)
  {
    nextNode = current -> next;
    current -> next = prev;
```

```
        prev = current;
        current = nextNode;
    }
    return prev;
}

Node *concatenateLists (Node *list1, Node *list2)
{
    if (list1 == NULL)
        return list2;

    Node *temp = list1;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = list2;

    return list1;
}

int main()
{
    Node *list1 = createNode(3);
    list1->next = createNode(1);
    list1->next->next = createNode(4);

    Node *list2 = createNode(2);
    list2->next = createNode(5);

    printf("Original list 1: ");
    displayList(list1);

    printf("Original list 2: ");
    displayList(list2);
```

```
list1 = sortList(list1);
printf("Sorted List1:");
displayList(list1);

list1 = reverseList(list1);
printf("Reversed List1:");
displayList(list1);

Node *concatenated = concatenatedLists(list1, list2);
printf("Concatenated List:");
displayList(concatenated);

return 0;
}
```

output:

original list 1: 3 → 1 → 4 → NULL
original list 2: 2 → 5 → NULL
Sorted list 1 : 1 → 3 → 4 → NULL
Reversed list 1 : 4 → 3 → 1 → NULL
Concatenated List : 4 → 3 → 1 → 2 → 5 → NULL

b) Implement single LL for stack operations (25/1/24)

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

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

```
typedef struct Node Node;
Node *createNode(int value)
{
  Node *newNode = (Node*)malloc(sizeof(Node));
  newNode->data = value;
  newNode->next = NULL;
  return newNode;
}
void displayList(Node *head)
{
  while(head != NULL)
  {
    printf("%d -> ", head->data);
    head = head->next;
  }
  printf("NULL\n");
}
typedef struct
{
  Node *top;
} LinkedList;
void push(LinkedList *stack, int value)
{
  Node *newNode = createNode(value);
  newNode->next = stack->top;
  stack->top = newNode;
}
int pop(LinkedList *stack)
{
  if(stack->top == NULL)
  {
```

Ended 25/11/24

```
        printf("Stack is empty\n");
        return -1;
    }
    int poppedValue = stack->top->data;
    Node *temp = stack->top;
    stack->top = stack->top->next;
    free(temp);
    return poppedValue;
}

int main()
{
    LinkedList stack;
    stack.top = NULL;

    printf("Stack operations:\n");
    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    displayList(stack.top);
    printf("Popped from stack: %d\n", pop(&stack));
    printf("Popped from stack: %d\n", pop(&stack));
    displayList(stack.top);

    return 0;
}
```

Output:

```
Stack operations:
30 -> 20 -> 10 -> NULL
Popped from stack: 30
Popped from stack: 20
20 -> NULL
```

Implement single LL for queue operations

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
  int data;
  struct Node *next;
};
typedef struct Node Node;
Node *createNode(int value)
{
  Node * newNode = (Node*) malloc (sizeof (Node));
  newNode -> data = value;
  newNode -> next = NULL;
  return newNode;
}
void displayList (Node *head)
{
  while (head != NULL)
  {
    printf ("%d ->", head -> data);
    head = head -> next;
  }
  printf ("NULL\n");
}
typedef struct
{
  Node *front;
  Node * rear;
} LinkedList;

void enqueue (LinkedList *queue, int value)
{
  Node * newNode = createNode (value);
```

Exit
25/11/24

```
    if (queue->front == NULL)
    {
        queue->front = newNode;
        queue->rear = newNode;
    }
    else
    {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}
int dequeue(LinkedList *queue)
{
    if (queue->front == NULL)
    {
        printf("Queue is empty\n");
        return -1;
    }
    int dequeuedvalue = queue->front->data;
    Node *temp = queue->front;
    queue->front = queue->front->next;
    free(temp);
    return dequeuedvalue;
}

int main()
{
    LinkedList queue;
    queue.front = NULL;
    queue.rear = NULL;

    printf("\n Queue operations:\n");
```

```
    enqueue (&queue, 40);
    enqueue (&queue, 50);
    enqueue (&queue, 60);
    displayList (queue.front);
    printf("Dequeued from queue: %d\n", dequeue(&queue));
    printf("Dequeued from queue: %d\n", dequeue(&queue));

    displayList (queue.front);
    return 0;
}
```

Output:

Queue operations:
40 → 50 → 60 → NULL
Popped from queue: 40
Popped from queue: 50
60 → NULL