Week - 5
Linked list

```
// Linked Lists 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 display (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 *concatLists (Node *list1, Node *list2)
{
    if (list1 == list2)
        return list2;
    Node *temp = list1;
    while (temp -> next != NULL)
    {
        temp = temp -> next;
    }
    temp -> next = list2;
    return list1;
}
```



```
void 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: ");
    display (list1);
    printf ("original list 2: ");
    display (list2);

    list1 = sortList (list1);
    printf ("Sorted list 1: ");
    display (list1);

    list1 = sortList (list1);
    printf ("Sorted list 1: ");
    display (list1);

    list1 = reverseList (list1);
    printf ("reversed list 1: ");
    display (list1);

    Node *concatenated = concatLists (list1, list2);
    printf ("Concatenated List: ");
    display (concatenated);
}
```

URBAN EDGE 25/1/24

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

URBAN EDGE 25/1/24

Week-5

Stack implementation using Linked lists

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

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

Node *create Node (int value)
{
    Node *newNode = (Node*) malloc (Size of (Node));
    newNode -> data = value;
    newNode -> next = NULL;
    Return newNode;
}

Void display (Node *head)
{
    while (head != NULL)
    {
        printf ("%d ->", head -> data);
        head = head -> next;
    }
    printf ("NULL \n");
}
typedef Stouct {
    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)
    {
        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;
}

void main()
{
    LinkedList Stack;
    Stack top = NULL;
    printf ("Stack operation : \n");
        push (&stack, 3);
        push (&stack, 4);
        push (&stack, 5);
        push (&stack, 6);
    printf ("popped value : ", pop (&stack));
    printf ("popped value : ", pop (&stack));
      display (stack. top);
}
```


Output :

Stack operations :

6 -> 5 -> 4 -> 3 -> NULL

popped value : popped value : 4 -> 3 -> NULL

Week - 5

Queue implementation using linked list

```
#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 display (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);
    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 dequeuevalue = queue -> front -> data;
    Node *temp = queue -> front;
    queue -> front = queue -> front -> next;
    free (temp);
    return dequeuevalue;
}
```

```
Void main()
{
    Linked List queue;
    queue.front = NULL;
    queue.rear = NULL;

    printf("\n queue operation :\n");
    enqueue (& queue, 40);
    enqueue (& queue, 50);
    enqueue (& queue, 60);
    display (queue.front)                    (& queue));
    printf ("dequeued from queue : %d\n", dequeue
                                        (& dequeue));
    printf ("dequeued from queue : %d\n", dequeue
                                        (& dequeue));
    display (queue.front);
}
```

output :

queue operations :
40 -> 50 -> 60 -> NULL
dequeued from queue : 40
dequeued from queue : 50
60 -> NULL

25/1/24