25-1-24

Week-5

(Q) Linked list operations:- sort, reverse, concatenate, swappin

```
#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) {
            ~~while~~ 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 = create Node(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);
    list 1 = sortList(list1);
    printf("sorted list : ");
    display (list1);
    list 1 = reverseList (list1);
    printf("reversed List 1: ");
    display(list1);

    Node *concaterated = concatLists(list1, list2);
    printf(" concaterated list: ");
    display (concaterated);
}
```

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
concaterated list :    4 → 3 → 1 → 2 → 5 → NULL
```

2) Implement stack 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 *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 : ");
        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 operations : \n");
    push( & stack, 10);
    push (& stack, 20);
    push (& stack, 25);
    push (&stack, 30);
    display (stack.top);
    printf(" popped value :  ", pop(&stack));
    printf(" popped value :  ", pop (&stack));
    display (stack.top);
}
```

Output :

```
stack operations:
30 -> 25 -> 20 -> 10 -> NULL
popped value : 30
popped value : ~~20~~25.
20 -> 10 -> NULL
```

3) Implement queue 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 = new Node;
        }
}
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;
}

void main()
{
        LinkedList queue;
        queue. front = NULL;
        queue. rear = NULL;
        printf ("\n queue operations: \n");
        enqueue (&queue, 40);
        enqueue (&queue, 50);
        enqueue (&queue, 60);
        display (queue. front);
        printf (" dequeued from queue : %d \n", dequeue (&queue));
        printf ("dequeued from queue : %d \n", dequeue(&queue));
        display(queue.front);
}
```

output:

```
    queue operations :
    40 → 50 → 60 → NULL
    dequeued from enqueue : 40
    dequeued from enqueue : 50
    60 → NULL
```