Week-05

25/01/24

Date
Page

// Sorting, Reversing & Concatination 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;
}
```

~~void~~

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

    list1 = sortList(list1);
    printf("sorted list1 : ");
    display(list1);

    list1 = reverseList(list1);
```

```
printf("reversed list 1: ");
display(list1);

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

output:-

original list 1: 3 → 1 → 4 → NULL
original list 2: 5 → 2 → NULL
sorted list 1 : 1 → 3 → 4 → NULL
reversed list 1 : 4 → 3 → 1 → NULL
concatenated list : 4 → 3 → 1 → 5 → 2 → NULL.

```
list2 = sortList(list2);
printf("sorted list 2: ");
display(list2);

list2 = reverseList(list2);
printf("reversed list 2: ");
display(list2);
}
```

output: sorted list 2 : 2 → 5 → NULL
reversed list 2 : 5 → 2 → NULL.

list1, list2);

→ // stack operation

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

struct Node
{
    int data;
    struct Node *next;
};
typedef struct Node Node;
Node * creatNode (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;
    }
    print("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\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 operations:\n");
    push(&stack, 5);
    push(&stack, 4);
    push(&stack, 3);
    push(&stack, 2);
    push(&stack, 1);
    display(stack.top);
    printf("popped value: %d", pop(&stack));
    printf("popped value: %d", pop(&stack));
```

```
    display(stack.top);
}
```

output:-
stack operations:
1 → 2 → 3 → 4 → 5 → NULL
popped value: 1 popped value: 2
3 → 4 → 5 → NULL

⇒ //Queue operation.

```
#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 dequeuedvalue = queue->front->data;
    Node *temp = queue->front;
    queue->front = queue->front->next;
    free(temp);
```

```
    return dequeued value;
}
void main()
{
    LinkedList queue;
    queue.front = NULL;
    queue.rear = NULL;

    printf("\n queue operations: \n");
    enqueue(&queue, 40);
    enqueue(&queue, 60);
    enqueue(&queue, 80);
    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 → 60 → 80 → Null
dequeued from queue: 40
dequeued from queue: 60
80 → NULL.