Lab - 6

6a) WAP to Implement Single Link list with following operations: Sort the linked list, Reverse Linked list, concatenation of two linked lists.

(b) WAP to Implement Single Link list to simulate stack & Queue operations.

Pseudo code

sorted

Compare ~~head~~each ~~element~~ Node will the remaining nodes.

→ ~~for~~ for initialize i=head and condition i->next != NULL and update condn i = i->next

for each i ⇒ for initialise j = i->next , condition j != NULL, update j = j->next

if i->data > j->data ⇒ ~~replace~~ swap

tempData = i->data
i->data = j->data
j->data = tempData

} swap

Reverse Linked list (

→ Parameter as struct Node *head

Assign prevNod = NULL, *curr = head, *next = NULL

while (curr != NULL)

next = curr->next

curr->next = prev

prev = curr

curr = next

Concatinate

Assign ~~temp~~ = head

while (temp->next != NULL)

{ temp = temp->next }

temp -> next = head2

printf("Linked list concated successfully\n");

& return head1

Code

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

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

struct Node *head1 = NULL;
struct Node *head2 = NULL;

struct Node* CreateList(int n);
void displayList(struct Node *head);
void displaysortLinkedList(struct Node*head);
struct Node* reverseLinkedList(struct Node *head);
struct Node* concatenateLinkedlist(struct Node *head1, struct
                                   Node *head2);

struct Node *CreateList(int n){
    struct Node *head=NULL, *newNode, *temp;
    int data;
    if(n<=0){
        printf("Number of nodes should be greater than 0\n");
        return NULL;
    }

    for(int i=1; i<=n; i++){
        newNode = (struct Node*)malloc(sizeof(struct Node));
        if(newNode==NULL){
            printf("Memory allocation failed\n");
            return head;
    }
```

```
        printf("Enter data for node %d : ", i);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;
        if (head == NULL)
            head = newNode;
        else
            temp->next = newNode;
        temp = newNode;
    }
    printf("Linked List created successfully\n");
    return head;
}
void displayList(struct Node *head){
    struct Node *temp = head;
    if (head == NULL){
        printf("List is empty\n");
        return;
    }
    printf("Linked List : ");
    while (temp != NULL){
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
void sortLinkedList(struct Node *head){
    struct Node *i, *j;
    int tempData;
    if (head == NULL){
        printf("List is empty, cannot sort\n");
        return;
    }
```

```
for (i=head; i->next!=NULL ; i=i->next){
    for(j=i->next; j!=NULL ; j=j->next){
        if(i->data > j->data){
            tempData = i->data;
            i->data = j->data;
            j->data = tempData;
        }
    }
}
printf("Linked list sorted successfully\n");
}
struct Node * reverseLinkedList(struct Node *head){
    struct Node *prev=NULL, *curr=head, *next=NULL;
    while (curr!=NULL){
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    printf("Linked list reversed successfully\n");
    return prev;
}
struct Node *concatenateLinkedList(struct Node *head1,
                                   struct Node *head2){
    struct Node *temp;
    if(head1==NULL)
        return head2;
    temp=head1;
    while (temp->next!=NULL)
        temp=temp->next;
    temp->next = head2;
    printf("Linked lists concatenated successfully\n");
    return head1;
}
```

```
int main(){
int choice, n;
while (1){
    printf("\n==========\n");
    printf("    LINKED LIST MENU \n");
    printf("==========\n");
    printf("1. Create List 1\n");
    printf("2. Create List 2\n");
    printf("3. Display List 1\n");
    printf("4. Display List 2\n");
    printf("5. sort List 1\n");
    printf("6. Reverse List 1\n");
    printf("7. Concatenate List1+List2\n");
    printf("8. Exit\n");
    printf("Enter choice: ");
    scanf("%d", &choice);

    switch(choice){

        case 1:
            printf("Enter number of nodes for list1: ");
            scanf("%d", &n);
            head1 = createList(n);
            break;

        case 2:
            printf("Enter number of nodes for list2: ");
            scanf("%d", &n);
            head2 = createList(n);
            break;

        case 3:
            displayList(head1);
            break;
```

```
case 4 : display List(head2);
         break;
case 5 : sort LinkedList(head1);
         break;
case 6 : head1 = reverseLinkedList(head1);
         break;
case 7 : head1 = concatenateLinkedList(head1, head2);
         printf("After concatenation:\n");
         displayList(head1);
         break;
case 8 : printf("Exiting program...\n");
         exit(0);
default : printf("Invalid choice. Try again\n");
    }
}
return 0;
}
```

o/p

```
=========
LINKED LIST
==========
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List1 + List2
8. Exit
Enter choice: 1
Enter number of nodes for list 1: 3
Enter data for node 1 : 5
Enter data for node 2 : 2
Enter data for node 3 : 9
Linked list created successfully
```

LINKED LIST MENU

1. Create list 1
2. Create list 2
3. Display list 1
4. Display list 2
5. Sort list 1
6. Reverse list 1
7. Concatenate list1 + list 2
8. Exit

Enter choice : 3

Linked list : 5 → 2 → 9 → NULL

LINKED LIST MENU

1. Create list 1
2. Create list 2
3. Display list 1
4. Display list 2
5. Sort list 1
6. Reverse list 1
7. Concatenate list1 + list 2
8. Exit

Enter choice : 5

Linked list sorted successfully

LINKED LIST MENU

1. Create list 1
2. Create list 2
3. Display list 1
4. Display list 2
5. Sort list 1
6. Reverse list 1

3. Concatenate list1 + list2
4. Exit
Enter choice : 3
Linked list : 2 → 5 → 9 → NULL

(b) Write to implement single linked list for Stack and Queue operations.

```
Create node ( )
Create new node
set node.data = value
set node - next = null
return node
```

Stacks operations :

```
push( )
new node = create node (value)
new node → next = top
top = new node
print (pushed element)
pop( )
if (top = null
    printf (empty)
    return
temp = top
print (popped element)
top = top → next
free (temp)
display( )
if top = = null
    printf (empty)
    return
temp = top
printf (Stack elements :)
while (temp! = value)
    printf (temp → data)
    temp = temp → next
```

Queue Operations

```
enqueue ( )
new node = Create node (value)
if rear = = null)
    front = rear = new node
else
    rear → next = new node
    rear = new node
printf ("enqueued element)
dequeue ( )
if front = = null)
    printf (empty)
    return
temp = front
printf (dequeued element)
front = front → next
if front = = null
    rear = null
free (temp)
display ( )
if front = null
    printf (empty)
    return
temp = front
printf ("queue elements)
while (temp != null)
    printf (temp → data)
    temp = temp → next
```

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};
struct Node *top = NULL;
struct Node *front = NULL;
struct Node *rear = NULL;
struct Node *createNode(int value);
{ struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}
void push(int value){
    struct node *newNode = createNode(value);
    newNode->next = top;
    top = newNode;
    printf("%d pushed to stack \n", value);
}
void pop() {
    if (top == NULL)
{   printf("stack is empty. Nothing to pop \n");
    return;
}
    struct node *temp = top;
    printf("%d popped from stack \n", top->data);
    top = top->next;
    free(temp);
}
```

```
void displayStack(){
    struct Node *temp = top;
    if (temp == NULL){
        printf("Stack is empty\n");
        return;
    }
printf("Stack (Top to bottom): ");
while (temp != NULL){
        printf("%d ", temp->data);
        temp = temp->next;
}
printf("\n");
}
void enqueue(int value){
    struct Node *newNode = createNode(value);
    if (rear == NULL){
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    } printf("%d enqueued to the queue\n", value);
}
void dequeue(){
    if (front == NULL)
    { printf("Queue is empty nothing to dequeue\n");
      return;
    }
    struct node *temp = front;
    printf("%d dequeued from the queue\n", front->data);
    front = front->next;
    if (front == NULL)
        rear = NULL;
    free(temp);
}
```

```
void display Queue()
{ struct node * temp = front;
  if (temp == NULL){
      printf ("Queue is empty\n");
      return;
}
printf ("Queue (front to Rear): ");
while (temp != NULL){
      printf("%d ", temp->data);

  } printf("\n");

}
int main() {
   int choice, value, ch;
    while (1){
       printf("\n-- Singly Linked list simulation --\n");
      printf ("%d 1. stack operations\n");
      printf("2. Queue Operations\n");
       printf ("3. Exit \n");
       printf ("Enter your choice");
      scanf ("%d", &choice);
      switch(choice){

case 1 :
       while(1){
        printf("\n -- stack Menu --\n");
         printf ("1. push\n");
         printf("2. pop \n");
         printf ("3. Display stack\n");
         printf("4. Back to main menu \n");
         printf("Enter your choice:");
         scanf ("%d", &ch);
         switch(ch){
            case 1: printf ("Enter value to push:")
                    scanf %d %value);
                  push(value)
```

```
case 2: pop()
        break;
case 3: display stack();
        break;
case 4: go to main menu;
default: printf("Invalid choice\n");
    }
}
break;
case 2; while(1){
    printf("\n --Queue menu--\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Display queue");
    printf("4. Back to main menu\n");
    printf("Enter your choice:");
    scanf("%d",&ch);
    switch(ch){
        case 1:
            printf("Enter value to enqueue:");
            scanf("%d",&value);
            enqueue(value);
            break;
        case 2:
            dequeue();
            break;
        case 3: display queue(); break;
        case 4: go to main_menu;
        default: printf("invalid choice\n");
    }
}
    break;
```

```
case 3:
    printf("Existing program\n");
    exit(0);
default:
    printf("Invalid choice. Try again\n");
}
main_menu();
}
return 0;
}
```

O/P ---singly linked list simulation---

1. stack operations
2. queue operations
3. Exit

Enter your choice: 1

--stack menu---

1. push
2. pop
3. Display stack
4. Back to main menu

Enter your choice: 1

Enter value to push: 10

10 pushed to stack

---stack menu---

1. push
2. pop
3. Display stack
4. Back to main menu

Enter your choice: 1

Enter the value to push: 20

20 pushed to stack

---stack menu---

1. push
2. pop
3. Display stack
4. Back to main menu

Enter your choice: 2

20 popped from stack

---stack menu---

1. push
2. pop
3. Display stack
4. Back to main menu

Enter your choice: 3

stack (top to bottom): 10

---stack menu---

1. push
2. pop
3. Display stack
4. Back to main menu

Enter your choice 4.

--- Singly linked list simulation ---

1. Stack operations
2. Queue operations
3. Exit

Enter your choice : 2

--- Queue menu ---

1. enqueue
2. dequeue
3. Display enqueue
4. Back to main menu

Enter your choice : 1

Enter value to enqueue : 10

10 enqueued to the queue

--- Queue Menu ---

1. Enqueue
2. Dequeue
3. Display Queue
4. Back to main menu

Enter your choice : 1

Enter value to enqueue : 20

20 enqueued to the queue.

20 enqueued to the queue

--- Queue Menu ---

1. Enqueue
2. Dequeue
3. Display Queue
4. Back to main menu

Enter your choice : 1

Enter value to enqueue : 30

30 enqueued to the queue

Enter your choice : 3

Queue (front to rear) : 20 30

--- Queue Operation ---

1. Enqueue
2. Dequeue
3. Display enqueue
4. Back to main menu

Enter your choice

--- Singly linked list simulation ---

1. Stack operations
2. Queue operations
3. Exit

Enter your choice : 3

Exiting program.

7. WAP to implement doubly linked list with primitive operations

a) Create doubly linked list

2) Insert a new node to the left of the node

c) delete a node based on specified value

d) display the contents of list

pseudocode

```
struct node{
    struct node *prev,*next;
    int data
};
insertatfront
void insertfront(int data){
    struct node *newNode;
    newNode->data=data;
    newNode->prev=null
    newNode->next=head
    if(head==null) head=tail=newNode
    else
        head->prev=newNode
        head=newNode
void insertatend(int data){
    struct node *newNode;
    newNode->data=data
    newNode->next=null
    newNode->prev=tail
    if(tail=null) head=tail=newNode
    else tail->next=newNode, tail=newNode
void insertatpos(int data, pos)
    int i
    struct node *newnode
case 1: if pos=1
        insert at front at(data)
case 2: for(i=1; i<pos-1 & temp!=null; i++) temp=temp->next
        if(temp==null || temp->next==null)
            insert at end(data)
```