USN:1BM19CS216
NAME:YASHASWINI SHAH
DATE:7/12/2020
LAB8

WAP Implement Single Link List with following operations

a) Sort the linked list.
b) Reverse the linked list.
c) Concatenation of two linked lists
d) implement Stack & Queues using Linked Representation

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct node
{
   int sem;
   struct node *next;
};
struct node *head= NULL;
struct node *head2= NULL;
int c=0;
void Insert()
{
          struct node *newnode;
          struct node *temp;
   int s;
   printf("Enter integer  :: ");
   scanf("%d",&s);
   newnode=(struct node*)malloc(sizeof(struct node));
   newnode->sem =s;
   if (head==NULL)
   {
     newnode->next=NULL;
     head=newnode;
     printf("First node of linked list created\n");
     c++;
   }
    else
    {
                    temp=head;
      while(temp->next!=NULL)
      {
```

```
                        temp=temp->next;
        }
                temp->next=newnode;
                newnode->next=NULL;
                c++;
                printf("Node created\n");
            }
}
void Insert2()
{
        struct node *newnode;
        struct node *temp;
    int s,y;
    printf("Enter elements to create list 2\n");
    do
    {
    printf("Enter integer  :: \n");
    scanf("%d",&s);
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->sem =s;
    if (head2==NULL)
    {
      newnode->next=NULL;
      head2=newnode;
      printf("first node of linked list created\n");
      c++;
    }
     else
     {
                temp=head2;
       while(temp->next!=NULL)
       {
                        temp=temp->next;
       }
                temp->next=newnode;
                newnode->next=NULL;
                c++;
                printf("Node created\n");
        }
        printf("Do you want to continue adding? ::0 or 1\n");
        scanf("%d",&y);
    }while(y!=0);
}
```

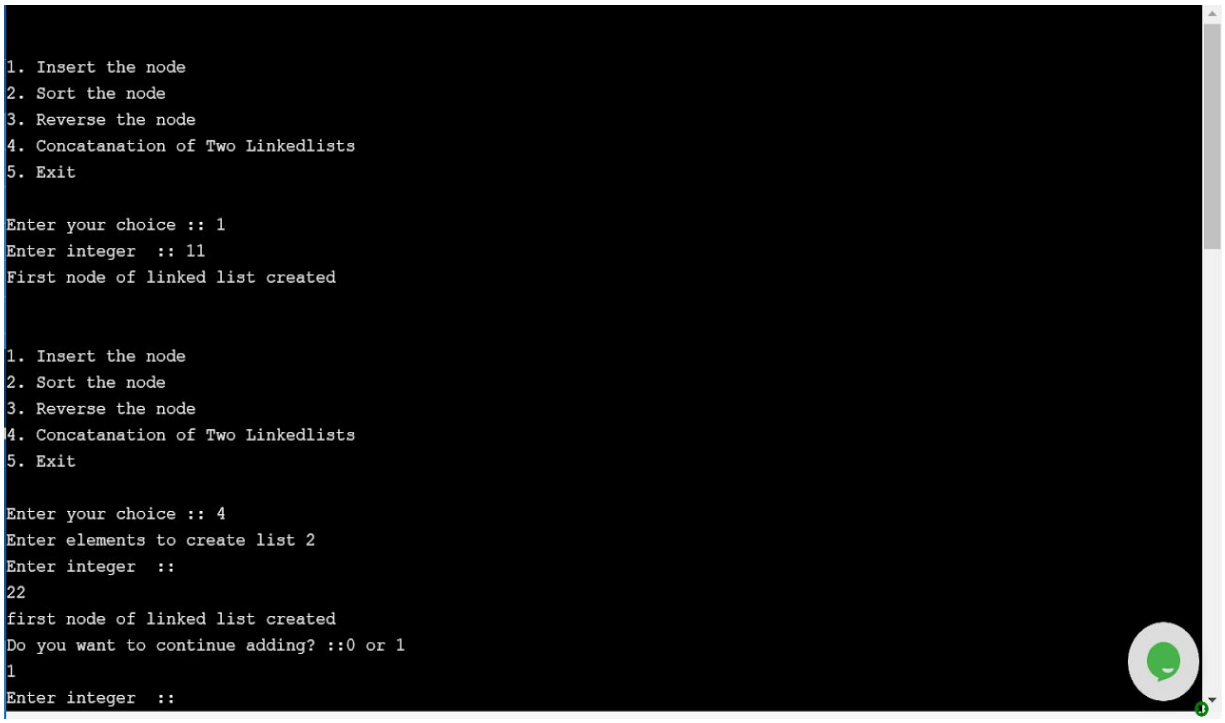
```
void bubbleSort()
{
    int swapped, i;
    struct node *ptr1;
    struct node *lptr = NULL;


    if (head == NULL)
        return;

    do
    {
        swapped = 0;
        ptr1 = head;

        while (ptr1->next != lptr)
        {
            if (ptr1->sem > ptr1->next->sem)
            {
                int temp = ptr1->sem;
                ptr1->sem = ptr1->next->sem;
                ptr1->next->sem = temp;
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    }
    while (swapped);
}

void reverse()
{
    struct node* prev = NULL;
    struct node* current = head;
    struct node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head= prev;
```

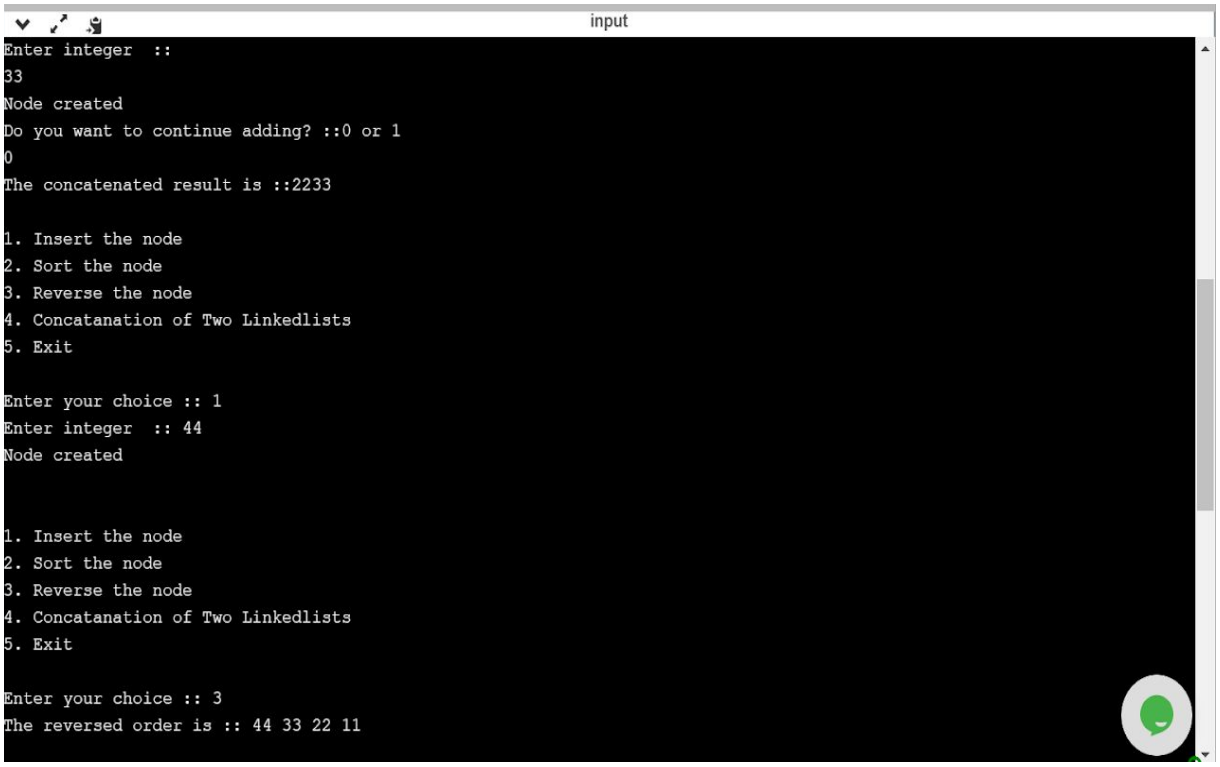
```
}

void concat()
{
        struct node *ptr;
        if(head==NULL)
        {
                head=head2;
        }
        if(head2==NULL)
              {
                      head2=head;
              }
        ptr=head;
        while(ptr->next!=NULL)
                ptr=ptr->next;
        ptr->next=head2;
}
void display1()
{
           struct node *ptr;
    ptr=head;
    int i=1;

    if(ptr==NULL)
    {
        printf("Linked list is empty!\n");
    }
    else
           {
        while(ptr!= NULL)
        {
                                printf(" %d",ptr->sem);
                                i++;
                                ptr=ptr->next;
        }

    }

}
void display2()
{
           struct node *ptr;
    ptr=head2;
```

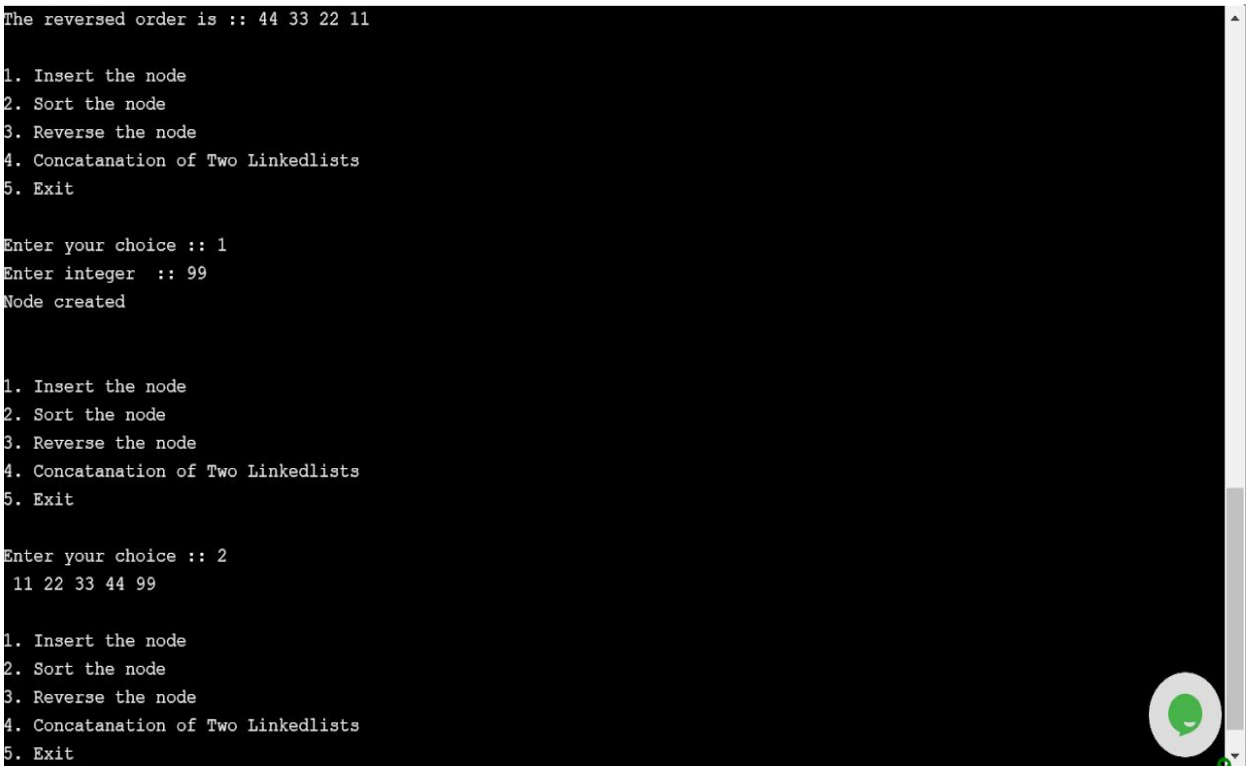
```
    int i=1;

    if(ptr==NULL)
    {
       printf("Linked list is empty!\n");
    }
    else
          {
       while(ptr!= NULL)
       {


                              printf("%d",ptr->sem);
                              i++;
                              ptr=ptr->next;
       }

    }

}

int main()
{
    int choice,pos;
    do
    {
                  printf("\n\1. Insert the node \n2. Sort the node\n3. Reverse the node\n4.
Concatanation of Two Linkedlists \n5. Exit\n");
                  printf("\nEnter your choice :: ");
                  scanf("%d",&choice);
                  switch(choice)
                  {
                              case 1:
                              Insert();
                              break;

                              case 2:
                              bubbleSort();
                              display1();
                              break;

                              case 3:
                              reverse();
                              printf("The reversed order is ::");
```

```
                display1();
                break;

                case 4:
            Insert2();
            concat();
            printf("The concatenated result is ::");
            display2();
                break;

                case 5:
                break;

                default:
                printf("Wrong choice!\n");
                break;
            }
    }while(choice!=5);
    return 0;
}
```

OUTPUT:




input
Enter integer ::
33
Node created
Do you want to continue adding? ::0 or 1
0
The concatenated result is ::2233
1. Insert the node
2. Sort the node
3. Reverse the node
4. Concatanation of Two Linkedlists
5. Exit
Enter your choice :: 1
Enter integer :: 44
Node created
1. Insert the node
2. Sort the node
3. Reverse the node
4. Concatanation of Two Linkedlists
5. Exit
Enter your choice :: 3
The reversed order is :: 44 33 22 11


The reversed order is :: 44 33 22 11
1. Insert the node
2. Sort the node
3. Reverse the node
4. Concatanation of Two Linkedlists
5. Exit
Enter your choice :: 1
Enter integer :: 99
Node created
1. Insert the node
2. Sort the node
3. Reverse the node
4. Concatanation of Two Linkedlists
5. Exit
Enter your choice :: 2
11 22 33 44 99
1. Insert the node
2. Sort the node
3. Reverse the node
4. Concatanation of Two Linkedlists
5. Exit

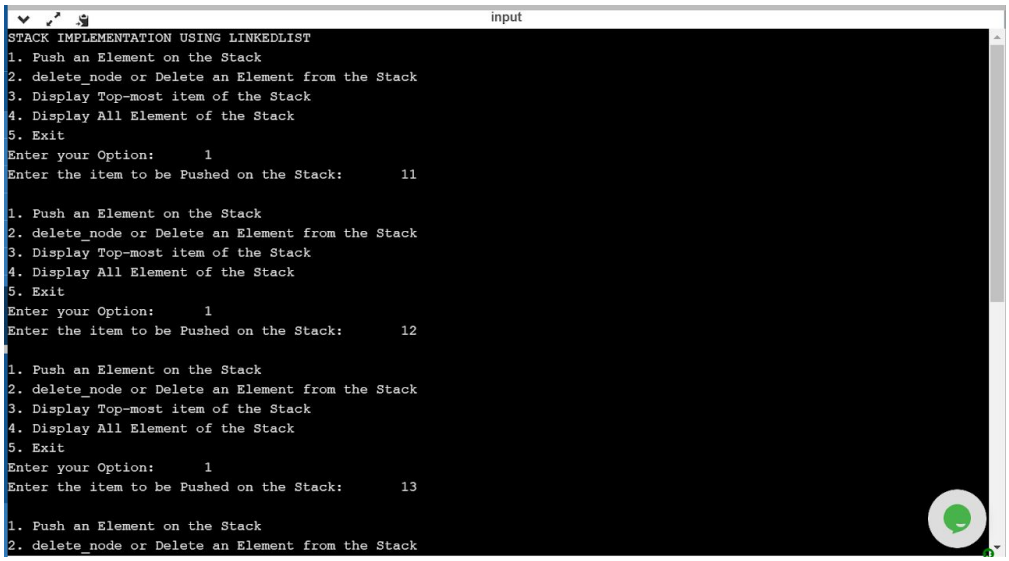
PART D

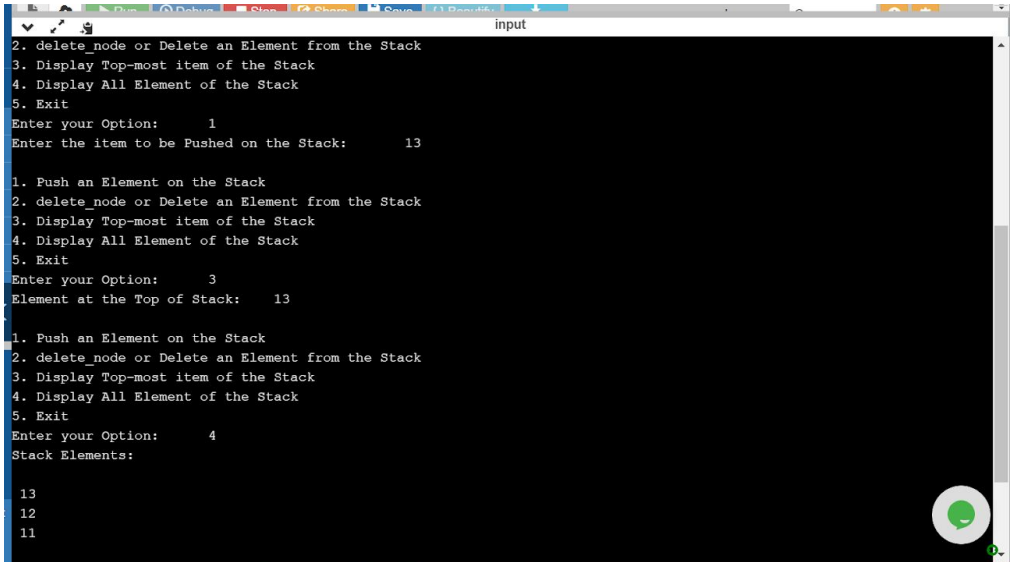
i)

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

struct node
{
    int data;
    struct node *link;
}*top = NULL;

int isEmpty()
{
    if(top == NULL)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void push(int item)
{
    struct node *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    if(temp == NULL)
    {
        printf("Stack is Full\n");
        return;
    }
    temp->data = item;
    temp->link = top;
    top = temp;
}

int delete_node()
{
    struct node *temp;
    int item;
    if(isEmpty())
    {
        printf("Stack is Empty\n");
```

```
            exit(1);
        }
        temp = top;
        item = temp->data;
        top = top->link;
        free(temp);
        return item;
}

void display()
{
        struct node *ptr;
        if(isEmpty())
        {
            printf("Stack is Empty\n");
            return;
        }
        printf("Stack Elements:\n\n");
        for(ptr = top; ptr != NULL; ptr = ptr->link)
        {
            printf(" %d\n", ptr->data);
        }
        printf("\n");
}

int peek()
{
        if(isEmpty())
        {
            printf("Stack is Empty\n");
            exit(1);
        }
        return top->data;
}

int main()
{
    printf("STACK IMPLEMENTATION USING LINKEDLIST");
        int option, element;
        while(1)
        {
            printf("\1. Push an Element on the Stack\n");
            printf("2. delete_node or Delete an Element from the Stack\n");
            printf("3. Display Top-most item of the Stack\n");
```

```
        printf("4. Display All Element of the Stack\n");
        printf("5. Exit\n");
        printf("Enter your Option:\t");
        scanf("%d", &option);
        switch(option)
        {
            case 1:
                printf("Enter the item to be Pushed on the Stack:\t");
                scanf("%d", &element);
                push(element);
                break;
            case 2:
                element = delete_node();
                printf("Deleted Element:\t%d\n", element);
                break;
            case 3:
                printf("Element at the Top of Stack:\t%d\n", peek());
                break;
            case 4:
                display();
                break;
            case 5:
                exit(1);
            default:
                printf("Wrong Option Selected\n");
        }
    }
    return 0;
}
```





il)

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

struct node
{
    int data;
    struct node *link;
}*front = NULL, *rear = NULL;

int isEmpty()
{
    if(front == NULL)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void insert(int item)
{
    struct node *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    if(temp == NULL)
```

```
    {
        printf("Queue is not Allocated\n");
        return;
    }
    temp->data = item;
    temp->link = NULL;
    if(front == NULL)
    {
        front = temp;
    }
    else
    {
        rear->link = temp;
    }
    rear = temp;
}

int delete_node()
{
    struct node *temp;
    int item;
    if(isEmpty())
    {
        printf("Queue is Empty\n");
        exit(1);
    }
    temp = front;
    item = temp->data;
    front = front->link;
    free(temp);
    return item;
}

int peek()
{
    if(isEmpty())
    {
        printf("Queue is Empty\n");
        exit(1);
    }
    return front->data;
}

void display()
```

```
{
    struct node *ptr;
    if(isEmpty())
    {
        printf("Queue is Empty\n");
        return;
    }
    printf("Queue Elements (or Nodes):\n");
    for(ptr = front; ptr != NULL; ptr = ptr->link)
    {
        printf("%d ", ptr->data);
    }
    printf("\n\n");
}

int main()
{
   printf("QUEUE IMPLEMENTATION USING LINKED LIST");
    int option, item;
    while(1)
    {
        printf("\n1. Insert an Element (Node) in the Queue\n");
        printf("2. Delete an Element from the Queue\n");
        printf("3. Display Element at the Front position\n");
        printf("4. Display All Elements of the queue\n");
        printf("5. Exit\n");
        printf("Enter your option:\t");
        scanf("%d", &option);
        switch(option)
        {
            case 1:
                printf("Enter the Element to Add in Queue:\t");
                scanf("%d", &item);
                insert(item);
                break;
            case 2:
                printf("The Deleted Element fromt the Queue:\t%d\n", delete_node());
                break;
            case 3:
                printf("Element at the Front:\t%d\n", peek());
                break;
            case 4:
                display();
                break;
```

```
            case 5:
                exit(1);
            default:
                printf("Wrong option\n");
        }
    }
    return 0;
}
```