DS LAB RECORD
NAME: SAKSHI ROY
SECTION: C
USN: 1BM19CS140
ACADEMIC YEAR:2020-2021

LAB 1:
QUESTION:
Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow

CODE

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define STACK_SIZE 5
int top=-1,st[5];
void push();
void pop();
void display();
void main()
{
        int ch;
        for(;;)
        {
                printf("\nStack Menu");
                printf("\n\n1.Push\n2.Pop\n3.Display\n4.Exit");
                printf("\n\nEnter your choice: ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: push();
                                break;
                        case 2: pop();
                                break;
                        case 3: display();
                                break;
                        case 4: exit(0);
                        default:
                          printf("\nInvalid option!! Try again");
                }
```

```
        }
}

void push()
{
        int item;
        if(top==STACK_SIZE-1)
        {
                printf("\nStack Overflow\n");
        }
        else
        {
                printf("\nEnter element to be entered in stack:");
                scanf("%d",&item);
                top=top+1;
                st[top]=item;
        }
}

void pop()
{
        if(top==-1)
        {
                printf("\nStack Underflow\n");
        }
        else
        {
                printf("\nThe value deleted from stack is: %d",st[top]);
                top=top-1;
        }
}
void display()
{
        int i;
        if(top==-1)
        {
                printf("\nStack is empty\n");
        }
        else
        {
                printf("\nContents of the stack are: \n");
                for(i=top;i>=0;--i)
                        printf("%d\n",st[i]);
        }
}
```

OUTPUT-

```
Stack Menu

1.Push
2.Pop
3.Display
4.Exit

Enter your choice: 1

Enter element to be entered in stack:4

Stack Menu

1.Push
2.Pop
3.Display
4.Exit

Enter your choice: 1

Enter element to be entered in stack:5

Stack Menu
```

```
1.Push
2.Pop
3.Display
4.Exit

Enter your choice: 1

Enter element to be entered in stack:7

Stack Menu

1.Push
2.Pop
3.Display
4.Exit

Enter your choice: 1

Enter element to be entered in stack:9

Stack Menu
```

```
1.Push
2.Pop
3.Display
4.Exit

Enter your choice: 2

The value deleted from stack is: 9
Stack Menu

1.Push
2.Pop
3.Display
4.Exit

Enter your choice: 3

Contents of the stack are:
7
5
4
```

```
2.Pop
3.Display
4.Exit

Enter your choice: 3

Contents of the stack are:
7
5
4

Stack Menu

1.Push
2.Pop
3.Display
4.Exit

Enter your choice: 4
```

LAB 2:
QUESTION:
WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

CODE

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

#define SIZE 100


char stack[SIZE];
int top = -1;
```

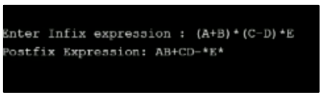
```
void push(char item)
{
	if(top >= SIZE-1)
	{
		printf("\nStack Overflow.");
	}
	else
	{
		top = top+1;
		stack[top] = item;
	}
}


char pop()
{
	char item;

	if(top <0)
	{
		printf("stack under flow: invalid infix expression");
		getchar();
		exit(1);
	}
	else
	{
		item= stack[top];
		top = top-1;
		return(item);
	}
}


int is_operator(char symbol)
{
	if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
	{
		return 1;
	}
	else
	{
	return 0;
	}
}
```

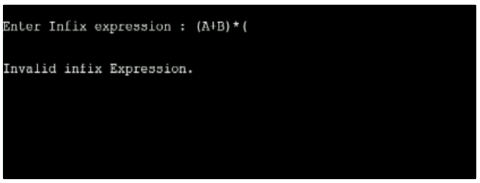
```
int precedence(char symbol)
{
	if(symbol == '^')
	{
		return(3);
	}
	else if(symbol == '*' || symbol == '/')
	{
		return(2);
	}
	else if(symbol == '+' || symbol == '-')
	{
		return(1);
	}
	else
	{
		return(0);
	}
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
	int i, j ;
	char item;
	char x;

	push('(');
	strcat(infix_exp,")");

	i=0;
	j =0;
	item=infix_exp[i];

	while(item!= '\0')
	{
		if(item== '(')
		{
			push(item);
		}
		else if( isdigit(item) || isalpha(item))
		{
			postfix_exp[j ] = item;
			j ++;
		}
```

```
        else if(is_operator(item) == 1)
        {
                x=pop();
                while(is_operator(x) == 1 && precedence(x)>= precedence(item))
                {
                        postfix_exp[j ] = x;
                        j ++;
                        x = pop();
                }
                push(x);


                push(item);
        }
        else if(item == ')')
        {
                x = pop();
                while(x != '(')
                {
                        postfix_exp[j ] = x;
                        j ++;
                        x = pop();
                }
        }
        else
        {
                printf("\nInvalid infix Expression.\n");
                getchar();
                exit(1);
        }
        i++;


        item= infix_exp[i];
}
if(top>0)
{
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
}
if(top>0)
{
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
```

```
        }


        postfix_exp[j] = '\0';


}


int main()
{
        char infix[SIZE], postfix[SIZE];

        printf("\nEnter Infix expression : ");
        gets(infix);

        InfixToPostfix(infix,postfix);
        printf("Postfix Expression: ");
        puts(postfix);

        return 0;
}
```

OUTPUT:





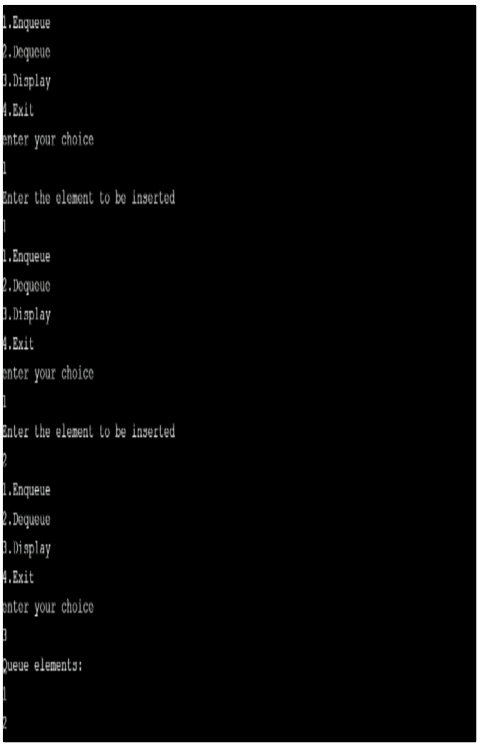
LAB 3:
QUESTION:
WAP to simulate the working of a queue of integers using an array. Provide the following operations a) Insert b) Delete c) Display The program should print appropriate messages for queue empty and queue overflow conditions

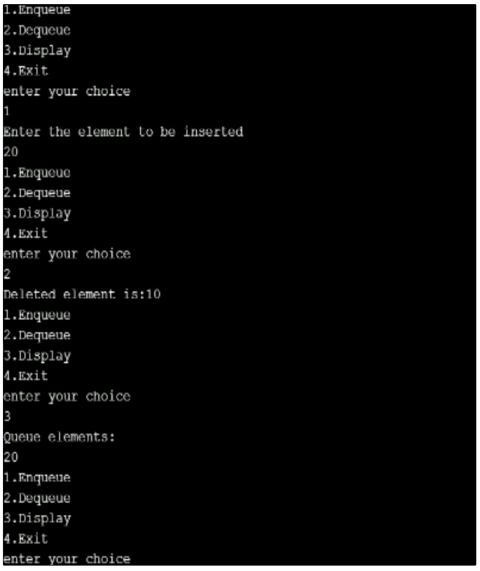
CODE

```
#include <stdio.h>
#define size 5
int arr[size];

int rare=-1;
int front=-1;
void Display();
void Enqueue();
void Dequeue();


void main()
{
  int choice;
  while(1)
  {
    printf("1.Enqueue\n");
    printf("2.Dequeue\n");
    printf("3.Display\n");
    printf("4.Exit \n");
    printf("enter your choice\n");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:Enqueue();
          break;
      case 2:Dequeue();
          break;
      case 3:Display();
          break;
      case 4:exit(1);
      default:printf("Invalid input\n");

    }
  }
}
void Display()
{
  int i;
  if(front==-1)
  printf("Queue is Empty\n");
  else{
    printf("Queue elements:\n");
    for(i=front;i<=rare;i++)
    {
```

```
      printf("%d",arr[i]);
      printf("\n");
      }
   }
}
 void Dequeue()
{
   if(front ==-1||front>rare)
   {
      printf("Queue underflow\n");
      return;
   }
   else{
      printf("Deleted element is:%d\n",arr[front]);
      front=front+1;
   }
}
void Enqueue()
{
   int item;
   if(rare==(size-1))
   printf("Queue Overflow\n");
   else
   {
      if(front ==-1)
      front=0;
      printf("Enter the element to be inserted\n");
      scanf("%d",&item);
      rare=rare+1;
      arr[rare]=item;
   }
}
```

OUTPUT-





LAB 4:

QUESTION:

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations. a) Insert b) Delete c) Display.

CODE

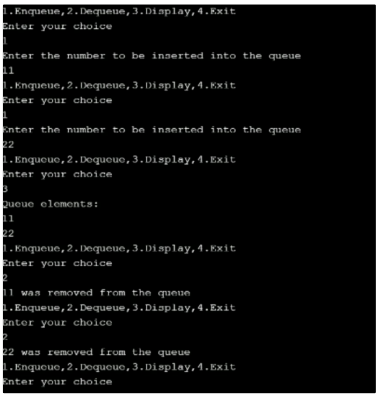
```
#include <stdio.h>
#include<stdlib.h>
```

```
#define size 5
int Q[size];
int rear=-1;
int front=-1;
int IsFull()
{
   if(front==(rear+1)%size)
   {
      return 0;
   }
   else
   {
      return -1;
   }

}
int IsEmpty()
{
   if(front==-1&&rear==-1)
   {
      return 0;
   }
   else
   {
      return -1;
   }
}
void Enqueue(int x)
{
   int item;
   if(IsFull()==0)
   {
   printf("Queue Overflow\n");

   }
   else if(IsEmpty()==0)
   {
      front=0;
      rear=0;
   }
   else
   {
     rear=(rear+1)%size;
   }
   Q[rear]=x;
```

```
}

int Dequeue()
{
   int x;
   if(IsEmpty()==0)
   {
      printf("Queue underflow\n");
   }
   else if(front==rear)
   {
      x=Q[front];
      front=-1;
      rear=-1;
   }
   else
   {
      x=Q[front];
      front=(front+1)%size;
   }
   return x;
}

void Display()
{
   int i;
   if(IsEmpty()==0)
   printf("Queue is Empty\n");
   else
   {
      printf("Queue elements:\n");
      for(i=front;i!=rear;i=((i+1)%size))
      {
      printf("%d\n",Q[i]);
      }
      printf("%d\n",Q[i]);
   }
}

void main()
{
   int choice,x,b;
   while(1)
   {
      printf("1.Enqueue,2.Dequeue,3.Display,4.Exit\n");
      printf("Enter your choice\n");
```

```
        scanf("%d",&choice);
        switch(choice)
        {
          case 1:printf("Enter the number to be inserted into the queue\n");
                 scanf("%d",&x);
                 Enqueue(x);
                 break;
          case 2:b=Dequeue();
                 printf("%d was removed from the queue\n",b);
                 break;
          case 3:Display();
                 break;
          case 4:exit(1);
          default:printf("Invalid input\n");

        }
      }
}
```

OUTPUT:



LAB 5 AND 6:

QUESTION:

WAP to Implement Singly Linked List with following operations:

a) Create a linked list

b) Insertion of a node at first position, at any position and at end of list

c) Deletion of first element, specified element and last element in the list

d) Display the contents of the linked list

CODE

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

void create();
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_begin();
void delete_end();
void delete_pos();

struct node
{
    int info;
    struct node *next;
};
struct node *start=NULL;
int main()
{
    int choice;
    while(1){

        printf("\n***MENU***\n");
        printf("\n 1. Create a list");
        printf("\n 2. Display the list");
        printf("\n 3. Insert node at the beginning");
        printf("\n 4. Insert node at the end");
        printf("\n 5. Insert node at any specified position");
        printf("\n 6. Delete node from the beginning");
        printf("\n 7. Delete node from the end");
        printf("\n 8. Delete node from any specified position");
        printf("\n 9. Exit");
        printf("\n Enter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                    create();
                    break;
            case 2:
                    display();
                    break;
            case 3:
                    insert_begin();
                    break;
```

```
                case 4:
                        insert_end();
                        break;
                case 5:
                        insert_pos();
                        break;
                case 6:
                        delete_begin();
                        break;
                case 7:
                        delete_end();
                        break;
                case 8:
                        delete_pos();
                        break;
                case 9:
                        exit(0);
                        break;

                default:
                        printf("\n Wrong Choice!\n");
                        break;
            }
        }
        return 0;
}
void create()
{
        struct node *temp,*ptr;
        temp=(struct node *)malloc(sizeof(struct node));
        printf("\nEnter the value for the node: ");
        scanf("%d",&temp->info);
        temp->next=NULL;
        if(start==NULL)
        {
            start=temp;
        }
        else
        {
            ptr=start;
            while(ptr->next!=NULL)
            {
                ptr=ptr->next;
            }
            ptr->next=temp;
        }
```

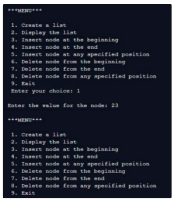
```
}
void display()
{
    struct node *ptr;
    if(start==NULL)
    {
        printf("\nList is empty!\n");
        return;
    }
    else
    {
        ptr=start;
        printf("\n The List elements are:\n");
        while(ptr!=NULL)
        {
            printf("%d  ",ptr->info);
            ptr=ptr->next ;
        }
    }
}
void insert_begin()
{
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    printf("\nEnter the value for the node: " );
    scanf("%d",&temp->info);
    temp->next =NULL;
    if(start==NULL)
    {
        start=temp;
    }
    else
    {
        temp->next=start;
        start=temp;
    }
}
void insert_end()
{
    struct node *temp,*ptr;
    temp=(struct node *)malloc(sizeof(struct node));
    printf("\nEnter the value for the node: " );
    scanf("%d",&temp->info );
    temp->next =NULL;
    if(start==NULL)
    {
```

```
            start=temp;
        }
        else
        {
            ptr=start;
            while(ptr->next !=NULL)
            {
                ptr=ptr->next ;
            }
            ptr->next =temp;
        }
}
void insert_pos()
{
        struct node *ptr,*temp;
        int i,pos;
        temp=(struct node *)malloc(sizeof(struct node));
        printf("\nEnter the position for the new node to be inserted: ");
        scanf("%d",&pos);
        printf("\nEnter the value of the node: ");
        scanf("%d",&temp->info) ;

        temp->next=NULL;
        if(pos==0)
        {
            temp->next=start;
            start=temp;
        }
        else
        {
            for(i=0,ptr=start;i<pos-1;i++)
            {
              ptr=ptr->next;
            }
            temp->next =ptr->next ;
            ptr->next=temp;
        }
}
void delete_begin()
{
        struct node *ptr;
        if(start==NULL)
        {
            printf("\nList is Empty!\n");
            return;
        }
```

```
    else
    {
        ptr=start;
        start=start->next ;
        printf("\nThe deleted element is: %d ",ptr->info);
        free(ptr);
    }
}
void delete_end()
{
    struct node *temp,*ptr;
    if(start==NULL)
    {
        printf("\nList is Empty!\n");
        exit(0);
    }
    else if(start->next ==NULL)
    {
        ptr=start;
        start=NULL;
        printf("\nThe deleted element is: %d ",ptr->info);
        free(ptr);
    }
    else
    {
        ptr=start;
        while(ptr->next!=NULL)
        {
            temp=ptr;
            ptr=ptr->next;
        }
        temp->next=NULL;
        printf("\nThe deleted element is: %d ",ptr->info);
        free(ptr);
    }
}
void delete_pos()
{
    int i,pos;
    struct node *temp,*ptr;
    if(start==NULL)
    {
        printf("\nThe List is Empty!\n");
        exit(0);
    }
    else
```

```
    {
        printf("\nEnter the position of the node to be deleted: \n");
        scanf("%d",&pos);
        if(pos==0)
        {
            ptr=start;
            start=start->next ;
            printf("\nThe deleted element is: %d ",ptr->info );
            free(ptr);
        }
        else
        {
            ptr=start;
            for(i=0;i<pos;i++)
            {
              temp=ptr;
              ptr=ptr->next ;
                if(ptr==NULL)
                {
                    printf("\nPosition not Found!\n");
                    return;
                }
            }
            temp->next =ptr->next ;
            printf("\nThe deleted element is: %d ",ptr->info);
            free(ptr);
        }
    }
}
```

OUTPUT:



```
 3. Insert node at the beginning
 4. Insert node at the end
 5. Insert node at any specified position
 6. Delete node from the beginning
 7. Delete node from the end
 8. Delete node from any specified position
 9. Exit
 Enter your choice: 1

Enter the value for the node: 45

***MENU***

 1. Create a list
 2. Display the list
 3. Insert node at the beginning
 4. Insert node at the end
 5. Insert node at any specified position
 6. Delete node from the beginning
 7. Delete node from the end
 8. Delete node from any specified position
 9. Exit
 Enter your choice: 2

 The List elements are:
23  45
```

```
  8. Delete node from any specified position
  9. Exit
  Enter your choice: 1

 Enter the value for the node: 56

 ***MENU***

  1. Create a list
  2. Display the list
  3. Insert node at the beginning
  4. Insert node at the end
  5. Insert node at any specified position
  6. Delete node from the beginning
  7. Delete node from the end
  8. Delete node from any specified position
  9. Exit
  Enter your choice: 8

 Enter the position of the node to be deleted:
 1

 The deleted element is:  45
 ***MENU***

  1. Create a list
```

LAB 7:

QUESTION:

WAP Implement Single Link List with following operations a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists

CODE

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

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

typedef struct node *Node;

Node getNode(){
   Node x;
   x = (Node)malloc(sizeof(struct node));
```

```
    if(x == NULL){
        printf("mem full\n");
        exit(0);
    }
    return x;
}

void freeNode(Node x){
    free(x);
}

Node insertRear(Node first,int item){
    Node temp,current;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL){
        return temp;
    }
    current = first;
    while(current->next != NULL){
        current = current->next;
    }
    current->next = temp;
    return first;
}


Node deleteRear(Node first){
    Node current,previous;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return first;
    }
    if(first->next == NULL){
        printf("Item deleted is %d\n",first->data);
        free(first);
        return NULL;
    }
    previous = NULL;
    current = first;
    while(current->next != NULL){
        previous = current;
        current = current->next;
    }
    printf("item deleted at rear end is %d\n",current->data);
```

```
    free(current);
    previous->next = NULL;
    return first;
}

Node concat(Node first, Node second){
    Node cur;
    if(first == NULL){
        return second;
    }
    if(second == NULL){
        return first;
    }
    cur = first;
    while(cur->next != NULL){
        cur = cur->next;
    }
    cur->next = second;
    return first;
}
Node swap(Node a, Node b){
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}
void sort(Node first){
    int swapped, i;
    Node cur;

    if(first == NULL){
        printf("List is empty\n");
        return;
    }

    do{
        swapped = 0;
        cur = first;

        while(cur->next != NULL){
            if(cur->data > cur->next->data){
                swap(cur,cur->next);
                swapped = 1;
            }
            cur = cur->next;
        }
    }
```

```
    while(swapped);
}

Node reverse(Node first){
    Node cur,temp;
    cur = NULL;
    while(first!=NULL){
        temp = first;
        first = first->next;
        temp->next = cur;
        cur = temp;
    }
    return cur;
}

void display(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return;
    }
    for(temp=first;temp!=NULL;temp = temp->next){
        printf("%d ",temp->data);
    }
    printf("\n");
}

int main(){
    int item,choice,flag = 1,n,i;
    Node first = NULL;
    Node a,b;

    while(flag == 1){
        printf("\n1. Insert Rear\n2. Delete Rear\n3. reverse\n4.Concat\n5. Sort\n6. Display\n7. Exit\n");
        printf("\nEnter Your choice: ");
        scanf("%d",&choice);
        printf("\n");
        switch(choice){
            case 1: printf("Enter the item to be inserted in the rear: \n");
                    scanf("%d",&item);
                    first = insertRear(first,item);
                    break;
            case 2: first = deleteRear(first);
                    break;
            case 3: first = reverse(first);
                    display(first);
```

```
            break;
        case 4: {
            printf("Enter number of node in List 1: \n");
            scanf("%d",&n);
            a = NULL;
            for(i=0;i<n;i++){
                printf("Enter the item: \n");
                scanf("%d",&item);
                a = insertRear(a,item);
            }

            printf("Enter number of node in List 2: \n");
            scanf("%d",&n);
            b = NULL;
            for(i=0;i<n;i++){
                printf("Enter the item: \n");
                scanf("%d",&item);
                b = insertRear(b,item);
            }

            a = concat(a,b);
            display(a);
            break;
        }
        case 5: sort(first);
                display(first);
                break;
        case 6: display(first);
                break;
        case 7: exit(0);
                break;
        default: printf("Enter correct option!\n");
      }
   }
}

//MODIFIED CONCATINATION
#include <stdio.h>
#include <stdlib.h>

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

typedef struct node *Node;
```

```
Node getNode(){
   Node x;
   x = (Node)malloc(sizeof(struct node));
   if(x == NULL){
      printf("mem full\n");
      exit(0);
   }
   return x;
}

void freeNode(Node x){
   free(x);
}

Node insertRear(Node first,int item){
   Node temp,current;
   temp = getNode();
   temp->data = item;
   temp->next = NULL;
   if(first == NULL){
      return temp;
   }
   current = first;
   while(current->next != NULL){
      current = current->next;
   }
   current->next = temp;
   return first;
}


Node deleteRear(Node first){
   Node current,previous;
   if(first == NULL){
      printf("List is empty cannot delete\n");
      return first;
   }
   if(first->next == NULL){
      printf("Item deleted is %d\n",first->data);
      free(first);
      return NULL;
   }
   previous = NULL;
   current = first;
   while(current->next != NULL){
```

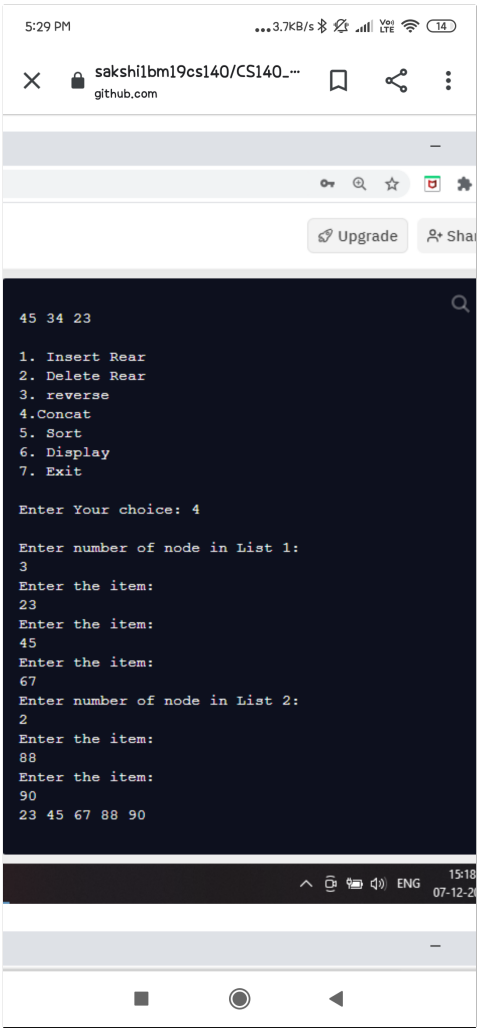
```
        previous = current;
        current = current->next;
    }
    printf("item deleted at rear end is %d\n",current->data);
    free(current);
    previous->next = NULL;
    return first;
}

Node concat(Node first, Node second){
    Node cur;
    if(first == NULL){
        return second;
    }
    if(second == NULL){
        return first;
    }
    cur = first;
    while(cur->next != NULL){
        cur = cur->next;
    }
    cur->next = second;
    return first;
}
Node swap(Node a, Node b){
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}
void sort(Node first){
    int swapped, i;
    Node cur;

    if(first == NULL){
        printf("List is empty\n");
        return;
    }

    do{
        swapped = 0;
        cur = first;

        while(cur->next != NULL){
            if(cur->data > cur->next->data){
                swap(cur,cur->next);
                swapped = 1;
```

```
        }
        cur = cur->next;
      }
    }
    while(swapped);
}

Node reverse(Node first){
    Node cur,temp;
    cur = NULL;
    while(first!=NULL){
        temp = first;
        first = first->next;
        temp->next = cur;
        cur = temp;
    }
    return cur;
}

void display(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return;
    }
    for(temp=first;temp!=NULL;temp = temp->next){
        printf("%d ",temp->data);
    }
    printf("\n");
}

int main(){
    int item,choice,flag = 1,n,i;
    Node first = NULL;
    Node a,b;

    while(flag == 1){
        printf("\n1. Insert Rear\n2. Delete Rear\n3. reverse\n4.Concat\n5. Sort\n6. Display\n7. Exit\n");
        printf("\nEnter Your choice: ");
        scanf("%d",&choice);
        printf("\n");
        switch(choice){
            case 1: printf("Enter the item to be inserted in the rear: \n");
                    scanf("%d",&item);
                    first = insertRear(first,item);
                    break;
```

```
        case 2: first = deleteRear(first);
                break;
        case 3: first = reverse(first);
                display(first);
                break;
        case 4: {

           printf("Enter number of node in List 2: \n");
           scanf("%d",&n);
           b = NULL;
           for(i=0;i<n;i++){
              printf("Enter the item: \n");
              scanf("%d",&item);
              b = insertRear(b,item);
           }

           a = concat(first,b);
           display(a);
           break;
        }
        case 5: sort(first);
                display(first);
                break;
        case 6: display(first);
                break;
        case 7: exit(0);
                break;
        default: printf("Enter correct option!\n");
     }
   }
}
```

OUTPUT:

```
Enter Your choice: 1

Enter the item to be inserted in the rear:
34

1. Insert Rear
2. Delete Rear
3. reverse
4.Concat
5. Sort
6. Display
7. Exit

Enter Your choice: 1

Enter the item to be inserted in the rear:
45

1. Insert Rear
2. Delete Rear
3. reverse
4.Concat
5. Sort
6. Display
7. Exit

Enter Your choice: 3

45 34 23
```



```
45 34 23

1. Insert Rear
2. Delete Rear
3. reverse
4.Concat
5. Sort
6. Display
7. Exit

Enter Your choice: 4

Enter number of node in List 1:
3
Enter the item:
23
Enter the item:
45
Enter the item:
67
Enter number of node in List 2:
2
Enter the item:
88
Enter the item:
90
23 45 67 88 90
```

```
Enter number of node in List 2:
2
Enter the item:
88
Enter the item:
90
23 45 67 88 90

1. Insert Rear
2. Delete Rear
3. reverse
4.Concat
5. Sort
6. Display
7. Exit

Enter Your choice: 5

23 34 45

1. Insert Rear
2. Delete Rear
3. reverse
4.Concat
5. Sort
6. Display
7. Exit

Enter Your choice:
```

MODIFIED CONCATENATION::

```
2. Delete Rear
3. reverse
4.Concat
5. Sort
6. Display
7. Exit

Enter Your choice: 1

Enter the item to be inserted in the rear:
45

1. Insert Rear
2. Delete Rear
3. reverse
4.Concat
5. Sort
6. Display
7. Exit

Enter Your choice: 4

Enter number of node in List 2:
2
Enter the item:
66
Enter the item:
90
23 45 66 90
```

LAB 8:
QUESTION:
implement Stack & Queues using Linked Representation

CODE

```
#include <stdio.h>

#include <stdlib.h>


typedef struct node{

int info;

struct node *link;

}NODE;


NODE* create(NODE *start)

{

 NODE *temp1,*temp2;

 int n,i;
```

```
printf("Enter number of nodes:\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
 temp1=(NODE*)malloc(sizeof(NODE));
 printf("Enter value for node %d:\n",i+1);
 scanf("%d",&temp1->info);
 temp1->link=NULL;
 if(start==NULL)
  start=temp1;
 else
 {
  temp2=start;
  while(temp2->link!=NULL)
   temp2=temp2->link;
  temp2->link=temp1;
 }
}
return start;
}

void disp(NODE *start)
{
```

```
 NODE *t;
 printf("Elements of the list:\n");
 for(t=start;t!=NULL;t=t->link)
 {
  if(t->link!=NULL)
   printf("%d  ",t->info);
  else
   printf("%d",t->info);
 }
 printf("\n");
}

void push();
void pop();
void display_S();



struct node *head;


void push ()
{
   int info;
   NODE *ptr = (NODE*)malloc(sizeof(NODE));
```

```
if(ptr == NULL)
{
   printf("Stack Empty!\n");
}
else
{
   printf("Enter the value:\n ");
   scanf("%d",&info);
   if(head==NULL)
   {
      ptr->info = info;
      ptr -> link = NULL;
      head=ptr;
   }
   else
   {
      ptr->info = info;
      ptr->link = head;
      head=ptr;

   }
   printf("Item is pushed\n");
```

```
    }
}

void pop()
{
    int item;
    NODE *ptr;
    if (head == NULL)
    {
        printf("Stack Underflow\n");
    }
    else
    {
        item = head->info;
        ptr = head;
        head = head->link;
        free(ptr);
        printf("Item popped\n ");

    }
}
void display_S()
{
```

```
    int i;
    NODE *ptr;
    ptr=head;
    if(ptr == NULL)
    {
       printf("Stack is empty\n");
    }
    else
    {
       printf("The Stack elements are: \n");
       while(ptr!=NULL)
       {
          printf("%d  ",ptr->info);
          ptr = ptr->link;
       }
    }
}

void enQueue();
void deQueue();
void display_Q();
struct node *front;
struct node *rear;
```

```
void enQueue()
{
   NODE *ptr;
   int item;

   ptr = (NODE *) malloc (sizeof(NODE));
   if(ptr == NULL)
   {
      printf("Queue Overflow\n");
      return;
   }
   else
   {
      printf("Enter value\n");
      scanf("%d",&item);
      ptr->info = item;
      if(front == NULL)
      {
         front = ptr;
         rear = ptr;
         front->link = NULL;
         rear->link = NULL;
```

```
        }
        else
        {
            rear->link=ptr;
            rear=ptr;
            rear->link=NULL;
        }
    }
}
void deQueue()
{
    NODE *ptr;
    if(front == NULL)
    {
        printf("Queue Underflow\n");
        return;
    }
    else
    {
        ptr=front;
        front=front->link;
        free(ptr);
        printf("Itemdeleted\n");
```

```
    }
}
void display_Q()
{
    NODE *ptr;
    ptr = front;
    if(front == NULL)
    {
        printf("Queue is Empty\n");
    }
    else
    {   printf("The Queue elements are:\n");
        while(ptr != NULL)
        {
            printf("%d  ",ptr -> info);
            ptr = ptr -> link;
        }
    }
}

int main()
{
 NODE *s1,*s2;
```

```
int ch;
s1=NULL;
s2=NULL;
while(1)
{

 printf("\n***MENU***\n");
 printf("\n1.Create\n2.Stack (Push)\n3.Stack (Pop)\n4.Stack (Display)\n5.Queue (Enqueue)\n6.Queue (Dequeue)\n7.Queue (Display)\n8.Exit\n");
 printf("Enter choice:\n");
 scanf("%d",&ch);
 switch(ch)
 {
    case 1: s1=create(s1);
            break;
    case 2: push();
            break;
    case 3: pop();
            break;
    case 4: display_S();
            break;
    case 5: enQueue();
            break;
    case 6: deQueue();
```

```
            break;
        case 7: display_Q();
            break;
        case 8: exit(0);
            break;
        default: printf("Wrong choice!");
             break;
    }


}
}
```

OUTPUT:

```
***MENU***

1.Create
2.Stack (Push)
3.Stack (Pop)
4.Stack (Display)
5.Queue (Enqueue)
6.Queue (Dequeue)
7.Queue (Display)
8.Exit
Enter choice:
1
Enter number of nodes:
4
Enter value for node 1:
34
Enter value for node 2:
23
Enter value for node 3:
45
Enter value for node 4:
90

***MENU***

1.Create
```

```
3.Stack (Pop)
4.Stack (Display)
5.Queue (Enqueue)
6.Queue (Dequeue)
7.Queue (Display)
8.Exit
Enter choice:
2
Enter the value:
 77
Item is pushed

***MENU***

1.Create
2.Stack (Push)
3.Stack (Pop)
4.Stack (Display)
5.Queue (Enqueue)
6.Queue (Dequeue)
7.Queue (Display)
8.Exit
Enter choice:
3
Item popped

***MENU***

1.Create
```

```
***MENU***

1.Create
2.Stack (Push)
3.Stack (Pop)
4.Stack (Display)
5.Queue (Enqueue)
6.Queue (Dequeue)
7.Queue (Display)
8.Exit
Enter choice:
4
The Stack elements are:
44
***MENU***

1.Create
2.Stack (Push)
3.Stack (Pop)
4.Stack (Display)
5.Queue (Enqueue)
6.Queue (Dequeue)
7.Queue (Display)
8.Exit
Enter choice:
5
Enter value
12
```

```
Enter choice:
5
Enter value
12

***MENU***

1.Create
2.Stack (Push)
3.Stack (Pop)
4.Stack (Display)
5.Queue (Enqueue)
6.Queue (Dequeue)
7.Queue (Display)
8.Exit
Enter choice:
5
Enter value
23

***MENU***

1.Create
2.Stack (Push)
3.Stack (Pop)
4.Stack (Display)
5.Queue (Enqueue)
6.Queue (Dequeue)
7.Queue (Display)
```

```
8.Exit
Enter choice:
6
Item deleted

***MENU***

1.Create
2.Stack (Push)
3.Stack (Pop)
4.Stack (Display)
5.Queue (Enqueue)
6.Queue (Dequeue)
7.Queue (Display)
8.Exit
Enter choice:
7
The Queue elements are:
23  45
***MENU***

1.Create
2.Stack (Push)
3.Stack (Pop)
4.Stack (Display)
5.Queue (Enqueue)
6.Queue (Dequeue)
7.Queue (Display)
8.Exit
```

LAB 9:
QUESTION:
WAP Implement doubly link list with primitive operations a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value d) Display the contents of the list

CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *prev;
   struct node *next;
   int data;
};
struct node *head;
void insertion_beginning();
void deletion_specified();
void display();
void main ()
{
int choice =0;
   while(choice != 9)
   {
      printf("\n***Menu***\n");
      printf("\n1.Insert at the beginning \n2.Delete from any specified location\n3.Display\n4.Exit\n");
      printf("\nEnter your choice: \n");
      scanf("\n%d",&choice);
      switch(choice)
      {
         case 1:
         insertion_beginning();
         break;
         case 2:
         deletion_specified();
         break;
         case 3:
         display();
         break;
         case 4:
         exit(0);
         break;
         default:
         printf("\n Invalid choice!");
      }
   }
```

```
}
void insertion_beginning()
{
  struct node *ptr;
  int item;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\n OVERFLOW!\n");
  }
  else
  {
   printf("\nEnter value to be inserted: \n");
   scanf("%d",&item);

  if(head==NULL)
  {
     ptr->next = NULL;
     ptr->prev=NULL;
     ptr->data=item;
     head=ptr;
  }
  else
  {
     ptr->data=item;
     ptr->prev=NULL;
     ptr->next = head;
     head->prev=ptr;
     head=ptr;
  }
  printf("\nNode inserted\n");
}

}
void deletion_specified()
{
   struct node *ptr, *temp;
   int val;
   printf("\nEnter the data after which the node is to be deleted: \n");
   scanf("%d", &val);
   ptr = head;
   while(ptr -> data != val)
   ptr = ptr -> next;
   if(ptr -> next == NULL)
   {
      printf("\nCan't be deleted!\n");
```

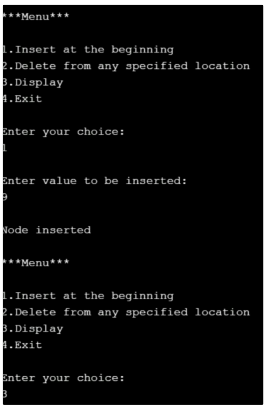
```
    }
    else if(ptr->next->next == NULL)
    {
        ptr->next = NULL;
        printf("\nNode deleted\n");
    }
    else
    {
        temp = ptr->next;
        ptr->next = temp->next;
        temp->next->prev = ptr;
        free(temp);
        printf("\nNode deleted\n");
    }
}
void display()
{
    struct node *ptr;
  if (head == NULL)
{
printf("\nThe List is Empty!\n");
}
else
{
    if(head == NULL)

    printf("\nThe values in the Doubly Linked list are:\n");
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d\t",ptr->data);
        ptr=ptr->next;
    }
}
}
```

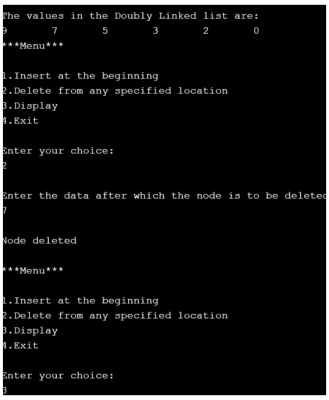
OUTPUT:

```
***Menu***

1.Insert at the beginning
2.Delete from any specified location
3.Display
4.Exit

Enter your choice:
1

Enter value to be inserted:
0

Node inserted

***Menu***

1.Insert at the beginning
2.Delete from any specified location
3.Display
4.Exit

Enter your choice:
1
```

```
Enter value to be inserted:
2

Node inserted

***Menu***

1.Insert at the beginning
2.Delete from any specified location
3.Display
4.Exit

Enter your choice:
1

Enter value to be inserted:
3

Node inserted

***Menu***

1.Insert at the beginning
2.Delete from any specified location
3.Display
4.Exit
```

```
Enter your choice:
1

Enter value to be inserted:
5

Node inserted

***Menu***

1.Insert at the beginning
2.Delete from any specified location
3.Display
4.Exit

Enter your choice:
1

Enter value to be inserted:
7

Node inserted
```





LAB 10:

QUESTION:

Write a program to: a) to construct a binary search tree. b) to traverse the tree using all the methods i.e., in-order, preorder and post order c) to display the elements in the tree.

CODE

```
#include <stdio.h>
#include <stdlib.h>
struct node{
 int data;
 struct node *llink;
 struct node *rlink;
};
typedef struct node *Node;
Node getnode(){
 Node x;
 x=(Node)malloc(sizeof(struct node));
 if(x==NULL){
  printf("Memory full\n");
  exit(0);
 }
```

```
 return x;
}
void freenode(Node x){
free(x);
}
Node insert(Node root,int item){
 Node temp,cur,prev;
 temp=getnode();
 temp->data=item;
 temp->rlink=temp->llink=NULL;
 if(root==NULL){
  return temp;
 }
 cur=root;
 prev=NULL;
 while(cur!=NULL){
  prev=cur;
  cur=(item<cur->data)?cur->llink:cur->rlink;
 }
 if(item<prev->data)
 prev->llink=temp;
 else
 prev->rlink=temp;
 return root;
}
Node deletion(Node root,int item){
 Node cur,parent,suc,q;
 if(root==NULL){
  printf("Empty\n");
  return root;
 }
 cur=root;
 parent=NULL;
 while(cur!=NULL&&item!=cur->data){
  parent=cur;
  cur=(item<cur->data)?cur->llink:cur->rlink;
 }
 if(cur==NULL){
  printf("Not found\n");
  return root;
 }
 if(cur->llink==NULL)
 q=cur->rlink;
 else if(cur->rlink==NULL)
 q=cur->llink;
 else{
```

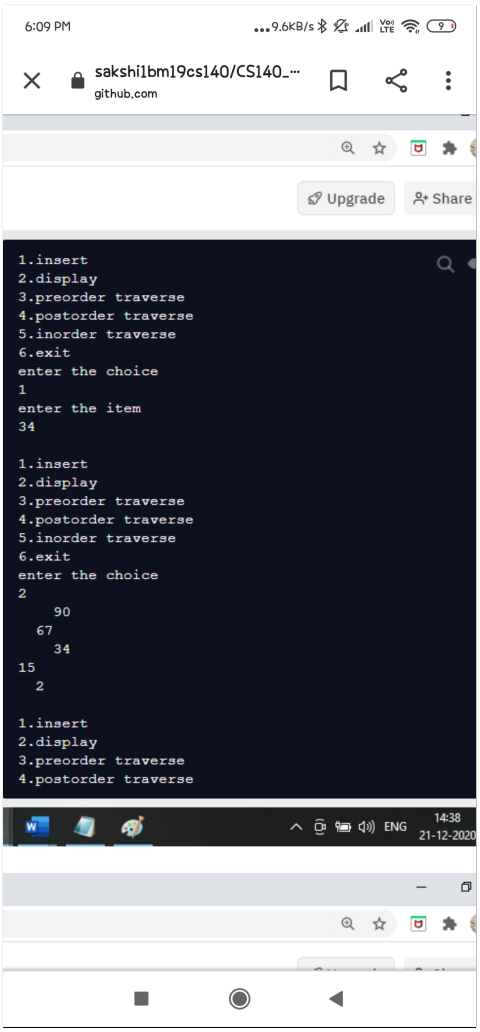
```
    suc=cur->rlink;
    while(suc->llink!=NULL)
    suc=suc->llink;
    suc->llink=cur->llink;
    q=cur->rlink;
  }
  if(parent==NULL){
    return q;
  }
  if(parent->llink==cur)
  parent->llink=q;
  else
  parent->rlink=q;
  free(cur);
  return root;
}
void display(Node root,int i){
  int j ;
  if(root!=NULL){
    display(root->rlink,i+1);
    for(j =0;j <i;j ++)
    printf("  ");
    printf("%d\n",root->data);
    display(root->llink,i+1);
  }
}
void preorder(Node root){
  if(root!=NULL){
    printf("%d\n",root->data);
    preorder(root->llink);
    preorder(root->rlink);
  }
}
void postorder(Node root){
  if(root!=NULL){
    postorder(root->llink);
    postorder(root->rlink);
    printf("%d\n",root->data);
  }
}
void inorder(Node root){
  if(root!=NULL){
    inorder(root->llink);
    printf("%d\n",root->data);
    inorder(root->rlink);
  }
```

```
}
int main()
{
int item,choice;
Node root=NULL;
for(;;)
{
printf("\n1.insert\n2.display\n3.pre\n4.post\n5.in\n6.delete\n7.exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item\n");
                        scanf("%d",&item);
                        root=insert(root,item);
                        break;
  case 2:display(root,0);
                        break;
  case 3:preorder(root);
                        break;
  case 4:postorder(root);
                        break;
  case 5:inorder(root);
                        break;
  case 6:printf("enter the item\n");
                        scanf("%d",&item);
                        root=deletion(root,item);
                        break;
  default:exit(0);
                         break;
              }
            }
}
```

OUTPUT:

```
1.insert
2.display
3.preorder traverse
4.postorder traverse
5.inorder traverse
6.exit
enter the choice
1
enter the item
15

1.insert
2.display
3.preorder traverse
4.postorder traverse
5.inorder traverse
6.exit
enter the choice
1
enter the item
2

1.insert
2.display
3.preorder traverse
4.postorder traverse
5.inorder traverse
6.exit
```

```
1.insert
2.display
3.preorder traverse
4.postorder traverse
5.inorder traverse
6.exit
enter the choice
1
enter the item
67

1.insert
2.display
3.preorder traverse
4.postorder traverse
5.inorder traverse
6.exit
enter the choice
1
enter the item
90

1.insert
2.display
3.preorder traverse
4.postorder traverse
5.inorder traverse
6.exit
enter the choice
```




```
3
15
2
67
34
90

1.insert
2.display
3.preorder traverse
4.postorder traverse
5.inorder traverse
6.exit
enter the choice
4
2
34
90
67
15

1.insert
2.display
3.preorder traverse
4.postorder traverse
5.inorder traverse
6.exit
enter the choice
```