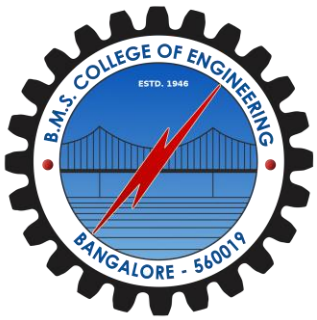


B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)

BENGALURU-560019

Sep-2020 to Dec-2020

DATA STRUCTURES

(19CS3PCDST)

LAB REPORT

Submitted by:

NAME: SAABIR SADIK

USN: 1BM19CS209

LAB Program 1

WAP for the below given scenario:

A university wants to automate their admission process. Students are admitted based on the marks scored in a qualifying exam. A student is identified by student id, age and marks in qualifying exam.

Data are valid, if:

- Age is greater than 20
- Marks is between 0 and 100 (both inclusive)
- A student qualifies for admission, if
- Age and marks are valid and
- Marks is 65 or more

Write a program to represent the students seeking admission in the university.

CODE:

```
#include <stdio.h>
  int main(int argc, char **argv)
 {
   int n,i;
   printf("Enter the total no of students:");
   scanf("%d",&n);
   struct student
 {
   int id;
   int age;
   int marks;

 }s[n];
 for(i=0;i<n;i++)
 {
   printf("Student %d - student id:",i+1);
   scanf("%d",& s[i].id);
```

```
  printf("Student %d - student age:",i+1);
  scanf("%d",& s[i].age);
  printf("Student %d - student marks:",i+1);;
  scanf("%d",& s[i].marks);
}
printf("The students who are eligible are:\n");
 for(i=0; i<n;i++)
 {
   if(s[i].age>20)
   {
   if(s[i].marks>=65)
   {
     printf("student %d is eligible for admission\n",i+1);
    }
   }
 }
 return 0;
}
```

OUTPUT:

```
Enter the total no of students:5
Student 1 - student id:001
Student 1 - student age:23
Student 1 - student marks:70
Student 2 - student id:002
Student 2 - student age:20
Student 2 - student marks:56
Student 3 - student id:003
Student 3 - student age:19
Student 3 - student marks:95
Student 4 - student id:004
Student 4 - student age:26
Student 4 - student marks:98
Student 5 - student id:005
Student 5 - student age:21
Student 5 - student marks:75
The students who are eligible are:
student 1 is eligible for admission
student 4 is eligible for admission
student 5 is eligible for admission
```

LAB Program 2

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>
#define SIZE 5
int top = -1;
int stack[SIZE];
void push(int ele) {
  if (isFull()) {
    printf("stack overflow!\n");
  } else {
    top++;
    stack[top] = ele;
  }
}
int pop() {
  if (isEmpty()) {
    return 0;
  } else {
    return stack[top--];
  }
}
int isEmpty() {
  if (top == -1)
    return 1;
  else
    return 0;
}
int isFull() {
```

```
  if (top == SIZE - 1)
    return 1;
  else
    return 0;
}
void display() {
  if (isEmpty())
    printf("Stack underflow!\n");
  else {
    printf("The elements are:\n");
    for (int i = 0; i <= top; i++) {
      printf("%d\n", stack[i]);
    }
  }
}
int main() {
  int c, d, p;
  while (1) {
    printf("Enter Choice\n**********\n \n1- Push\n2- Pop\n3- Display\n");
    scanf("%d", & c);
    switch (c) {
    case 1:
      printf("Enter an element:\n");
      scanf("%d", & d);
      push(d);
      break;
    case 2:
      p = pop();
      if (p == 0)
        printf("Stack underflow!\n");
      else
        printf("Element deleted succesfully!\n");
      break;
    case 3:
      display();
      break;
```

```c
    case 4:
      exit(0);
    default:
      printf("Invalid input!\n");
    }
   return 0;
}
```

OUTPUT:

```
> ./main
Enter Choice
**********

1- Push
2- Pop
3- Display
3
Stack underflow!
Enter Choice
**********

1- Push
2- Pop
3- Display
1
Enter an element:
4
Enter Choice
**********

1- Push
2- Pop
3- Display
1
Enter an element:
5
Enter Choice
**********

1- Push
2- Pop
3- Display
1
Enter an element:
7
Enter Choice
**********

1- Push
2- Pop
3- Display
1
Enter an element:
8
Enter Choice
**********

1- Push
2- Pop
3- Display
1
Enter an element:
9
```

```
Enter Choice
**********

1- Push
2- Pop
3- Display
1
Enter an element:
6
stack overflow!
Enter Choice
**********

1- Push
2- Pop
3- Display
1
Enter an element:
10
stack overflow!
Enter Choice
**********

1- Push
2- Pop
3- Display
2
Element deleted succesfully!
```

LAB Program 3

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)

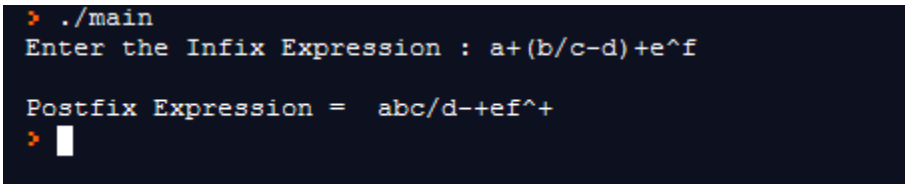
CODE:

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 30
char stack[SIZE];
int top=-1;

push(char elem)
{
    stack[++top]=elem;
}

char pop()
{
    return(stack[top--]);
}

int pr(char symbol)
{

  if(symbol == '^')
  {
    return(3);
  }
  else if(symbol == '*' || symbol == '/')
  {
    return(2);
  }
```

```
  else if(symbol == '+' || symbol == '-')
  {
    return(1);
  }
  else
  {
    return(0);
  }
}
int main()
{
char infix[50],postfix[50],ch,elem;
  int i=0,k=0;

    printf("Enter the Infix Expression : ");
    scanf("%s",infix);
    push('#');
    while( (ch=infix[i++]) != '\0')
 {
     if( ch == '(') push(ch);
       else
   if(isalnum(ch)) postfix[k++]=ch;
   else
     if( ch == ')')
  {
     while( stack[top] != '(')
     postfix[k++]=pop();
      elem=pop();
  }
       else
       {
      while( pr(stack[top]) >= pr(ch) )
       postfix[k++]=pop();
       push(ch);
      }
    }
```

```
    while( stack[top] != '#')
        postfix[k++]=pop();
    postfix[k]='\0';
    printf("\nPostfix Expression =  %s\n",postfix);
}
```

OUTPUT:



LAB Program 4

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 MAX 5
void insert();
void delete();
void display();

int queue_array[MAX];
int rear = - 1;
int front = - 1;

int main()
{
int option;
 while (1)
  {
    printf("Menu\n");
    printf("***********\n");
    printf("1- Insert \n");
    printf("2- Delete\n");
    printf("3- Display\n");
    printf("4- Exit \n");
    printf("***********\n");
    printf("Enter your option: ");
    scanf("%d", &option);
```

```
        switch (option)
       {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(1);
            default:
            printf("Invalid Option \n");
         }
    }
}
void insert()
{
 int add_item;
 if (rear == MAX - 1)
 printf("Queue Overflow \n");
 else
  {
   if (front == - 1)
   front = 0;
    printf("Enter the element to insert: ");
    scanf("%d", &add_item);
    rear = rear + 1;
    queue_array[rear] = add_item;
    }
}
void delete()
{
  if (front == - 1 || front > rear)
```

```
    {
     printf("Queue Underflow \n");
     return ;
    }
    else
    {
    printf("deleted element from queue is : %d\n", queue_array[front]);
    front = front + 1;
    }
}
void display()
{
  int i;
  if (front == - 1)
   printf("Queue is empty \n");
    else
    {
  printf("Elements in the Queue are:\n");
  for (i = front; i <= rear; i++)
  printf("%d ", queue_array[i]);
  printf("\n");
    }
}
```

OUTPUT:

```
> ./main
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 3
Queue is empty
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Enter the element to insert: 5
```

```
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Enter the element to insert: 6
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Enter the element to insert: 7
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Enter the element to insert: 8
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Enter the element to insert: 9
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Queue Overflow
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 3
Elements in the Queue are:
5 6 7 8 9
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 2
deleted element from queue is : 5
```

LAB Program 5

WAP to simulate the working of a circular 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
void enQueue(int);
void deQueue();
void display();

int cQueue[SIZE], front = -1, rear = -1;

int main()
{
   int choice, value;
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Insert\n2. Delete\n3. Display\n4. Exit");
      printf("\n******************\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      printf("\n------------------\n");
      switch(choice){
   case 1: printf("\nEnter the value to be insert:  ");
     scanf("%d",&value);
```

```
      enQueue(value);
      break;
   case 2: deQueue();
      break;
   case 3: display();
      break;
   case 4: printf("\n Program Exit Successful\n");
   exit(0);
   default: printf("\nPlease select the correct choice!!!\n");
      }
   }
}
void enQueue(int value)
{
   if((front == 0 && rear == SIZE - 1) || (front == rear+1))
      printf("\nCircular Queue Overflow, insertion not possible!\n");
   else{
      if(rear == SIZE-1 && front != 0)
   rear = -1;
      cQueue[++rear] = value;
      printf("\nInsertion Success!!\n");
      if(front == -1)
   front = 0;
   }
}
void deQueue()
{
   if(front == -1 && rear == -1)
      printf("\nCircular Queue Underflow, deletion not possible!\n");
   else{
      printf("\nDeleted element : %d\n",cQueue[front++]);
      if(front == SIZE)
   front = 0;
      if(front-1 == rear)
   front = rear = -1;
   }
```

```
}
void display()
{
   if(front == -1)
      printf("\nCircular Queue is Empty!\n");
   else{
      int i = front;
      printf("\nElements in Circular Queue are:\n");
      if(front <= rear){
   while(i <= rear)
      printf("%d\t",cQueue[i++]);
      }
      else{
   while(i <= SIZE - 1)
      printf("%d\t", cQueue[i++]);
   i = 0;
   while(i <= rear)
      printf("%d\t",cQueue[i++]);
      }
   }
}
```

Output:

```
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
******************
Enter your choice: 1

------------------

Enter the value to be insert:  1

Insertion Success!!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
******************
Enter your choice: 1

------------------

Enter the value to be insert:  5

Insertion Success!!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
******************
Enter your choice: 1

------------------

Enter the value to be insert:  7

Insertion Success!!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
******************
Enter your choice: 1

------------------

Enter the value to be insert:  21

Insertion Success!!
```

```
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
******************
Enter your choice: 1

------------------

Enter the value to be insert:  4

Insertion Success!!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
******************
Enter your choice: 1

------------------

Enter the value to be insert:  7

Circular Queue Overflow, insertion not possible!

****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
******************
Enter your choice: 3

------------------

Elements in Circular Queue are:
1   5   7   21  4
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
******************
Enter your choice: 4

------------------

 Program Exit Successful
>
```

LAB Program 6

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

typedef struct node{
  int data;
  struct node *link;
}node;

node *head=NULL;

void insert_end()
{
   node *temp;
   temp=(node *)malloc(sizeof(node));

   printf("Please enter the node element:\n");
   scanf("%d",&temp->data);
    temp->link=NULL;
   if(head==NULL)
  {
  head=temp;
  }
  else
  {
```

```
   node *p=head;
   while(p->link!=NULL)
  {
   p=p->link;
  }
  p->link=temp;
 }
}

void insert_begin()
{
 node *temp;
temp=(node *)malloc(sizeof(node));
 printf("Please enter the node element:\n");
 scanf("%d",&temp->data);
 temp->link=NULL;

 if(head==NULL)
 {
    head=temp;
 }
 else
 {
  temp->link=head;
  head=temp;
 }
}

int length()
{
  node *p;
  p=head;
  int i=0;

  while(p!=NULL)
  {
```

```
  i++;
  p=p->link;
 }
return i;
}

void insert_after(){

node *p,*temp;
int loc,i=1;
printf("Please enter the location:");
scanf("%d",&loc);

if(loc>length())
{
 printf("incorrect location!. The list has %d nodes",length());
}
else
{

p=head;
while(i<loc)
{
 p=p->link;
 i++;
}
temp=(node *)malloc(sizeof(node));
printf("Please enter the node element:\n");
scanf("%d",&temp->data);
temp->link=NULL;

temp->link=p->link;
p->link=temp;
}
}
void delete()
```

```
{
int loc;
node *temp;
printf("Please enter the locatin of node to be deleted:\n");
scanf("%d",&loc);

if (loc>length())
{
  printf("Invalid Node!\n");
}
else if (loc==1)
{
 temp=head;
head=temp->link;
temp->link=NULL;
free(temp);
}
else
{
node *p=head,*q;
int i=1;
while(i<loc-1)
{
  p=p->link;
  i++;
}
q=p->link;
p->link=q->link;
q->link=NULL;
free(q);
}
}

void display()
{
 node *temp=head;
```

```
 if(temp==NULL)
 {
  printf("No nodes in the list\n");
 }
 else
 {
  while(temp!=NULL)
  {
   printf("%d\n",temp->data);
   temp=temp->link;
  }
 }
}

int main()
{
 int op,len;
 while(1)
{ printf("**********MENU**********\n");
  printf("1. Insert in beginning\n");
  printf("2. Insert at end\n");
  printf("3. Insert after node\n");
  printf("4. Delete node\n");
  printf("5. Display\n");
  printf("6. Length of list\n");
  printf("7. Exit\n");
  printf("************************\n");
  printf("Please enter your choice:");
  scanf("%d",&op);
  switch (op)
  {
  case 1:insert_begin();
    break;
  case 2: insert_end();
    break;
  case 3: insert_after();
```

```
 break;
  case 4: delete();
    break;
  case 5: display();
    break;
  case 6: len=length();
        printf("The length is %d\n",len);
    break;
  case 7: exit(0);
     break;
  default: printf("Invalid choice!\n");
  }
 }
return 0;
}
```

Output:

```
> ./main
**********MENU**********
1. Insert in beginning
2. Insert at end
3. Insert after node
4. Delete node
5. Display
6. Length of list
7. Exit
************************
Please enter your choice:5
No nodes in the list
**********MENU**********
1. Insert in beginning
2. Insert at end
3. Insert after node
4. Delete node
5. Display
6. Length of list
7. Exit
************************
Please enter your choice:4
Please enter the locatin of node to be deleted:
1
Invalid Node!
**********MENU**********
1. Insert in beginning
2. Insert at end
3. Insert after node
4. Delete node
5. Display
6. Length of list
7. Exit
************************
Please enter your choice:6
The length is 0
```

```
**********MENU**********
1. Insert in beginning
2. Insert at end
3. Insert after node
4. Delete node
5. Display
6. Length of list
7. Exit
************************
Please enter your choice:1
Please enter the node element:
10
**********MENU**********
1. Insert in beginning
2. Insert at end
3. Insert after node
4. Delete node
5. Display
6. Length of list
7. Exit
************************
Please enter your choice:1
Please enter the node element:
20
**********MENU**********
1. Insert in beginning
2. Insert at end
3. Insert after node
4. Delete node
5. Display
6. Length of list
7. Exit
************************
Please enter your choice:2
Please enter the node element:
30
**********MENU**********
1. Insert in beginning
2. Insert at end
3. Insert after node
4. Delete node
5. Display
6. Length of list
7. Exit
************************
Please enter your choice:5
20
10
30
```

LAB Program 7

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 <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("Please enter elements to create list\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 u 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 ;
    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);
    printf("\n");
    i++;
    ptr=ptr->next;
     }
   }
   }
```

```c
int main()
{
   int choice,pos;
   do
   {
   printf("\n1- Insert Node \n2- Sort Node\n3- Reverse Node\n4- Concatenate 2 Lists \n5- Exit\n");
   printf("\nEnter your choice: ");
   scanf("%d",&choice);
   switch(choice)
   {
case 1:
     Insert();
     break;
 case 2:
     bubbleSort();
     display1();
     break;
 case 3:
     reverse();
     display1();
     break;
 case 4:
      Insert2();
      concat();
      display1();
     break;
 case 5:
     break;
    default:
     printf("Wrong choice!\n");
     break;
   }
 }
while(choice!=5);
 return 0;
}
```

Output:

```
> ./main

1- Insert Node
2- Sort Node
3- Reverse Node
4- Concatenate 2 Lists
5- Exit

Enter your choice: 1
Enter integer: 3
first node of linked list created

1- Insert Node
2- Sort Node
3- Reverse Node
4- Concatenate 2 Lists
5- Exit

Enter your choice: 1
Enter integer: 2
Node created!

1- Insert Node
2- Sort Node
3- Reverse Node
4- Concatenate 2 Lists
5- Exit

Enter your choice: 1
Enter integer: 1
Node created!

1- Insert Node
2- Sort Node
3- Reverse Node
4- Concatenate 2 Lists
5- Exit

Enter your choice: 2
 1 2 3
1- Insert Node
2- Sort Node
3- Reverse Node
4- Concatenate 2 Lists
5- Exit

Enter your choice: 3
 3 2 1
1- Insert Node
2- Sort Node
3- Reverse Node
4- Concatenate 2 Lists
5- Exit

Enter your choice: 4
Please enter elements to create list
Enter integer:
6
first node of linked list created!
Do u want to continue adding 0 or 1
0
 3 2 1 6
```

LAB Program 8

Implement Stack & Queues using Linked Representation

Stack Code

```
#include <stdio.h>
#include <stdlib.h>
void push();
void pop();
void display();
struct node
{
int val;
struct node *next;
};
struct node *head;

void main ()
{
   int choice=0;
   while(choice != 4)
   {
      printf("\n\nMenu\n***************");
      printf("\n1- Push\n2- Pop\n3- Display\n4- Exit\n***************\n");
      printf("\nEnter your choice: \n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         {
            push();
            break;
         }
```

```
        case 2:
        {
            pop();
            break;
        }
        case 3:
        {
            display();
            break;
        }
        case 4:
        {
            printf("Exiting!");
            break;
        }
        default:
        {
            printf("Please enter valid choice!");
        }
    };
}
}
void push ()
{
    int val;
    struct node *ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("not able to push the element");
    }
    else
    {
        printf("Enter the value:");
        scanf("%d",&val);
        if(head==NULL)
        {
```

```
            ptr->val = val;
            ptr -> next = NULL;
            head=ptr;
        }
        else
        {
            ptr->val = val;
            ptr->next = head;
            head=ptr;
        }
        printf("Item pushed!");
    }
}
  void pop()
{
    int item;
    struct node *ptr;
    if (head == NULL)
    {
        printf("Underflow!");
    }
    else
    {
        item = head->val;
        ptr = head;
        head = head->next;
        free(ptr);
        printf("Item popped!");
    }
}
void display()
{
    int i;
    struct node *ptr;
    ptr=head;
    if(ptr == NULL)
```

```
    {
        printf("Stack is empty!\n");
    }
    else
    {
        printf("Printing Stack elements!\n");
        while(ptr!=NULL)
        {
            printf("%d\n",ptr->val);
            ptr = ptr->next;
        }
    }
}
```

OUTPUT:

```
Menu
***************
1- Push
2- Pop
3- Display
4- Exit
***************

Enter your choice:
1
Enter the value:5
Item pushed!

Menu
***************
1- Push
2- Pop
3- Display
4- Exit
***************

Enter your choice:
6
Please enter valid choice!

Menu
***************
1- Push
2- Pop
3- Display
4- Exit
***************

Enter your choice:
1
Enter the value:8
Item pushed!
```

```
Menu
***************
1- Push
2- Pop
3- Display
4- Exit
***************

Enter your choice:
3
Printing Stack elements!
8
5


Menu
***************
1- Push
2- Pop
3- Display
4- Exit
***************

Enter your choice:
2
Item popped!

Menu
***************
1- Push
2- Pop
3- Display
4- Exit
***************

Enter your choice:
2
Item popped!

Menu
***************
1- Push
2- Pop
3- Display
4- Exit
***************

Enter your choice:
3
Stack is empty!


Menu
***************
1- Push
2- Pop
3- Display
4- Exit
***************

Enter your choice:
4
Exiting!exit status 8
```

Queue Code:

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

typedef struct node {
int val;
struct node* next;
}node;

node* front = NULL;
node* rear = NULL;

void enqueue(int input) {

   node* ptr = (node*) malloc((sizeof(node)));
   ptr->next = NULL;
   ptr -> val = input;
   if(front==NULL&&rear==NULL)
   {
      front = rear = ptr;
   }
   else{
      rear->next = ptr;
      rear = ptr;
   }
   printf("\n\nEnqueued!\n\n");
}
void dequeue() {

   if(front == NULL && rear == NULL){
      printf("\n\nQueue is empty!\n\n");
      return;
   }
   if(front -> next == NULL)
   {
```

```
        free(front);
        front = rear = NULL;
        printf("\n\nDequeued!\n\n");
        return;
    }
    printf("\n\nDequeued element is %d",front->val);
    front=front->next;
}
void display() {

    if(front==NULL&&rear==NULL){
        printf("\n\nQueue is empty\n\n");
        return;
    }
    printf("Queue contains: ");
    node* ptr = front;
    do
    {
      printf("%d ", ptr->val);
        ptr = ptr->next;
    } while(ptr!=NULL);
    printf("\n\n");
}
int main() {
    int choice, input;
    while(1) {
        printf("\n1- Enqueue\n");
        printf("2- Dequeue\n");
        printf("3- Display\n");
        printf("4- Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        if(choice == -1)
            break;
        switch(choice) {
        case 1:
```

```
            printf("Enter value to enqueue: ");
            scanf("%d", &input);
            enqueue(input);
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        default:
            printf("\n\nWrong Input!\n\n");
        }
    }
    return 0;
}
```

Output:

```
1- Enqueue
2- Dequeue
3- Display
4- Exit
Enter your choice: 1
Enter value to enqueue: 6


Enqueued!


1- Enqueue
2- Dequeue
3- Display
4- Exit
Enter your choice: 1
Enter value to enqueue: 4


Enqueued!


1- Enqueue
2- Dequeue
3- Display
4- Exit
Enter your choice: 1
Enter value to enqueue: 3


Enqueued!
```

```
1- Enqueue
2- Dequeue
3- Display
4- Exit
Enter your choice: 1
Enter value to enqueue: 8


Enqueued!


1- Enqueue
2- Dequeue
3- Display
4- Exit
Enter your choice: 1
Enter value to enqueue: 7


Enqueued!


1- Enqueue
2- Dequeue
3- Display
4- Exit
Enter your choice: 3
Queue contains: 6 4 3 8 7


1- Enqueue
2- Dequeue
3- Display
4- Exit
Enter your choice: 2


Dequeued element is 6
1- Enqueue
2- Dequeue
3- Display
4- Exit
Enter your choice: 2


Dequeued element is 4
1- Enqueue
2- Dequeue
3- Display
4- Exit
Enter your choice: 3
Queue contains: 3 8 7
```

LAB Program 9

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;
   int n;
   struct node *next;
}*h,*temp,*temp1,*temp2,*temp4;
void insert_beg();
void insert_atpos();
void display_beg();
void delete_atpos();
int count = 0;
int main()
{  int ch;
   h = NULL;
   temp = temp1 = NULL;
   printf("*************************************\n");
   printf("\n 1 - Insert\n");
   printf("\n 2 - Delete at specific position\n");
   printf("\n 3 - Display\n");
   printf("\n 4 - Exit\n");
   printf("*************************************\n");
   while (1)
   {
```

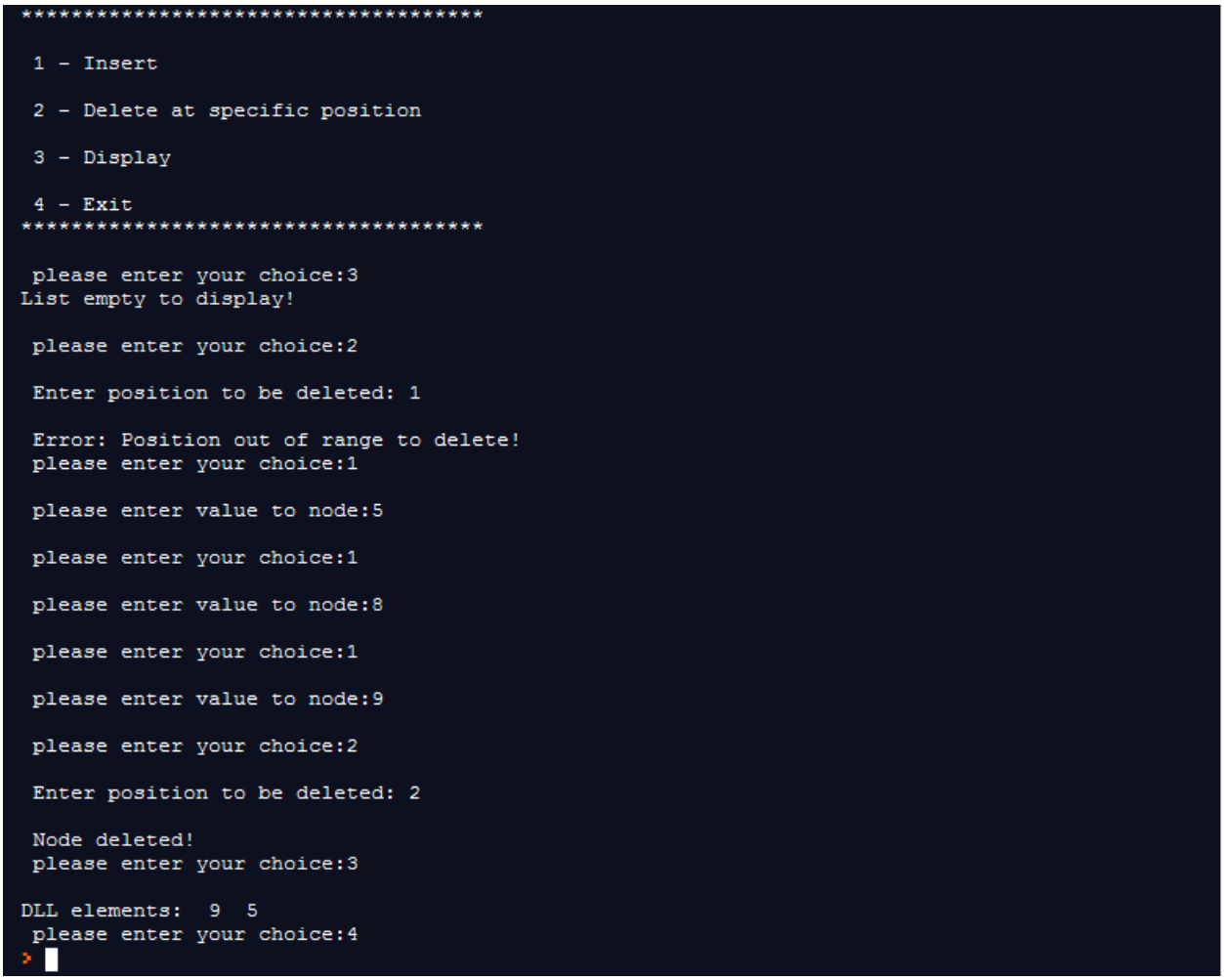
```
        printf("\n please enter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
           insert_beg();
           break;
        case 2:
           delete_atpos();
           break;
        case 3:
           display_beg();
           break;
        case 4:
           exit(0);
        default:
           printf("\nInvalid choice!\n");
        }
    }
}
/* TO create an empty node */
void create()
{
   int data;
   temp =(struct node *)malloc(1*sizeof(struct node));
   temp->prev = NULL;
   temp->next = NULL;
   printf("\n please enter value to node:");
   scanf("%d", &data);
   temp->n = data;
   count++;
}
/*  TO insert at beginning */
void insert_beg()
{
   if (h == NULL)
```

```
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {  create();
        temp->next = h;
        h->prev = temp;
        h = temp;
    }
}
/* To delete an element */
void delete_atpos()
{
    int i = 1, pos;
    printf("\n Enter position to be deleted: ");
    scanf("%d", &pos);
    temp2 = h;
    if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Error: Position out of range to delete!");
        return;
    }
    if (h == NULL)
    {
        printf("\n Error: Empty list no elements to delete!");
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
```

```
        if (i == 1)
        {
            if (temp2->next == NULL)
            {
                printf("Node deleted from list!");
                free(temp2);
                temp2 = h = NULL;
                return;
            }
        }
        if (temp2->next == NULL)
        {
            temp2->prev->next = NULL;
            free(temp2);
            printf("Node deleted from list!");
            return;
        }
        temp2->next->prev = temp2->prev;
        if (i != 1)
            temp2->prev->next = temp2->next;
        if (i == 1)
            h = temp2->next;
        printf("\n Node deleted!");
        free(temp2);
    }
    count--;
}
/* display from beginning */
void display_beg()
{
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("List empty to display!\n");
        return;
```

```
    }
    printf("\nDLL elements: ");

    while (temp2->next != NULL)
    {
        printf(" %d ", temp2->n);
        temp2 = temp2->next;
    }
    printf(" %d ", temp2->n);
}
```

Output:



LAB Program 10

Write a program
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>
typedef struct node {
  int data;
  struct node * left;
  struct node * right;
}
node;
node * create() {
  node * temp;
  temp = (node * ) malloc(sizeof(node));
  printf("Enter the new data\n");
  scanf("%d", & temp -> data);
  temp -> right = temp -> left = NULL;
  return temp;
}void insert(node * root, node * temp) {
  if (temp -> data < root -> data) {
    if (root -> left != NULL)
      insert(root -> left, temp);
    else
      root -> left = temp;
  } else {
    if (root -> right != NULL)
      insert(root -> right, temp);
    else
      root -> right = temp;
  }
```

```
}
void preorder(node * root) {
  if (root != NULL) {
    printf("%d\t", root -> data);
    preorder(root -> left);
    preorder(root -> right);
  }
}
void inorder(node * root) {
  if (root != NULL) {
    inorder(root -> left);
    printf("%d\t", root -> data);
    inorder(root -> right);
  }
}
void postorder(node * root) {
  if (root != NULL) {
    postorder(root -> left);
    postorder(root -> right);
    printf("%d\t", root -> data);
  }
}
void main() {
  int ch;
  node * temp, * root = NULL;
  while (1) {
    printf("Menu\n***************\n1- Insert \n2- Preorder Display\n3-
Inorder Display\n4- Postorder Display\n5- Exit\n***************\n");
    scanf("%d", & ch);
    switch (ch) {
    case 1:
      temp = create();
      if (root == NULL)
        root = temp;
      else
        insert(root, temp);
```

```
      printf("Element inserted successfully!\n");
      break;
    case 2:
      if (root == NULL) {
        printf("Tree is empty!\n");
      } else {
        printf("Preorder Display:\n");
        preorder(root);
        printf("\n");
      }
      break;
    case 3:
      if (root == NULL) {
        printf("tree is empty!\n");
      } else {
        printf("Inorder Display:\n");
        inorder(root);
        printf("\n");
      }
      break;
    case 4:
      if (root == NULL) {
        printf("Tree is empty!\n");
      } else {
        printf("Postorder Display::\n");
        postorder(root);
        printf("\n");
      }
      break;
    case 5:
      exit(0);
    default:
      printf("invalid choice!\n");
    }
  }
}
```

OUTPUT:

```
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Enter the element to insert: 6
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Enter the element to insert: 7
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Enter the element to insert: 8
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Enter the element to insert: 9
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 1
Queue Overflow
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 3
Elements in the Queue are:
5 6 7 8 9
Menu
***********
1- Insert
2- Delete
3- Display
4- Exit
***********
Enter your option: 2
deleted element from queue is : 5
```