VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

DATA STRUCTURE LAB RECORD

Submitted by

SHREYAS KS(1BM18CS104)

Under the Guidance of

Prof. POOJA S

Assistant Professor

BMSCE

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Sep-2020 to Jan-2021

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the LAB RECORD carried out by **SHREYAS KS (1BM18CS104)** who is the bonafide students of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visveswaraiah Technological University, Belgaum during the year 2020-2021. The lab report has been approved as it satisfies the academic requirements in respect of **DATA STRUCTURE LAB RECORD (19CS3PCDST)** work prescribed for the said degree.

Signature of the Guide
Prof. Prof. POOJA S
Assistant Professor
BMSCE, Bengaluru

Signature of the HOD
Dr. Umadevi V
Associate Prof.& Head, Dept. of CSE
BMSCE, Bengaluru

External Viva

Name of the Examiner | Signature with date

1._____ _____

2. _____ _____

INDEX

1: 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

Date __/__/__
Page
SPLASH

Lab program -1

1: Write a program to stimulate 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.

```
#include <stdio.h>
#include <stdlib.h>
#define Max 5
int top = -1
int s[10];
int item;
void push()
{
 if (top == Max -1)
 {
 printf("STACK Overflow elements
        cannot be added\n");
  return;
 }
top += 1;
s[top] = item;
}
int pop()
{
  if top== -1)
```

```
{
 return -1;
}
return s[top--];
}
void display()
{
 if (top == -1)
{
 printf ("Stack is empty no item
          can be printed\n");
 return;
}
else
{
 for (int i = top; i > 0; i--)
{
 printf ("%d\n", s[i]);
}
}
}
int main()
{
int item_deleted;
int choice;
for (;;)
{
printf ("\n ***** Stack Menue *****\n");
printf ("1: push\n 2: pop\n 3: display
         \n 4: Exit")
printf ("Enter the item to be
         inserted\n");
```

```
Scanf ("%d", &choice);
Switch (choice)
{
case 1:
printf("Enter the item to be
        inserted \n");
Scanf ("%d", &item);
push ();
break;

case 2:
item_deleted = pop();
if (item_deleted == -1)
printf ("Stack is empty \n");
else {
printf ("Deleted item is %d \n",
        item_deleted);
}
break;

case 3:
printf ("Elements of stack are \n");
display();
break;

default:
exit (0);
}
}
```

```
#include<stdio.h>
#include<stdlib.h>
#define Max 5
int top=-1;
int s[10];
int item;
void push()
{
if(top==Max-1)
{
printf("STACK OVERFLOW ELEMENT CANNOT BE ADDED\n");
return;
}
top+=1;
s[top]=item;
}
int pop()
{
if(top==-1)
{
```

```
return -1;
}
return s[top--];
}
void display()
{
if(top==-1)
{
printf("STACK IS EMPTY NO ITEM CAN BE PRINTED\n"); return;
}
else
{
for(int i=top;i>=0;i--)
{
printf("%d\n",s[i]);
}
}
}
int main()
{
```

```
int item_deleted;
int choice;
for(;;)
{
printf("\n1:push\n2:pop\n3:display\n4:exit\n");
printf("Enter your choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter the item to be inserted\n");
scanf("%d",&item);
push();
break;
case 2:
item_deleted=pop();
if(item_deleted==-1)
printf("Stack is empty\n");
else{
```

```
printf("Deleted item is %d\n",item_deleted);
}
break;
case 3:


printf("Elements of stack are \n");

display();

break;

default:

exit(0);

}

}
```

}

OUTPUT:

```
PS C:\Users\It's mine!\Desktop\C C++> cd "c:\Users\It's mine!\Desktop\C C++\.vscode\" ; if ($?) { gcc sss.c -o sss } ; if ($?) { .\sss }
***Stack Menue***
1:push
2:pop
3:display
4:exit
Enter any number between 1 to 4
1
Enter the item to be inserted
45
***Stack Menue***
1:push
2:pop
3:display
4:exit
Enter any number between 1 to 4
3
Elements of stack are
45
***Stack Menue***
1:push
2:pop
3:display
4:exit
Enter any number between 1 to 4
2
Deleted item is 45
***Stack Menue***
1:push
2:pop
3:display
4:exit
Enter any number between 1 to 4
3
Elements of stack are
STACK IS EMPTY NO ITEM CAN BE PRINTED
***Stack Menue***
1:push
2:pop
3:display
4:exit
```

2: 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)

Date / /
Page
SPLASH

Lab Program - 2

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

Soln.

```
#include <stdio.h>
#define N 100
int stack[N];
int top = -1;

void push(int item) {
    if (top == N-1)
        printf("Stack overflow!\n");
    else
        stack[++top] = item;
}
int pop() {
    if (top == -1)
        printf("Stack underflow!\n");
    else
        return stack[top--];
}
int priority(char op) {
    switch(op) {
    case '*': return 2;
              break;
    case '/': return 2;
              break;
```

```
        case '+' : return 1;
                   break;
        case '-' : return 1;
                   break;
        case '(' : return 0;
                   break;
    }
}
int main()
{
    char s[50];
    char t[50];
    int i;
    int choice = 1;

    do {
        i = 0;
        printf("Enter your infix expression:
               ");
        scanf("%s", s);
        for (int i=0; s[i] != '\0'; i++) {
          switch (s[i]) {
            case '(' : push('(');
                       break;
            case ')' : while (stack[top] != '(')
              {
                t[i++] = pop();
              }
              pop();
              break;
            case '*';
            case '/' :
```

```
        case '+':
        case '-': while (top != -1 &&
                  priority (stack[top]) >=
                  priority (s[i])) {
                    t[j++] = pop ();
                  }
                  push (s[i]);
                  break;
        default : t[j++] = s[i];
        }
    }
while (top != -1) {
        t[j++] = pop ();
    }
t[j] = '\0';
printf ("Postfix expression for \"%s\"
        is \"%s\".\n", s, t);
printf ("\n :: Menu :: \n1 try
        another infix expression.\n2.
        Exit \n Enter your choice : ");
scanf ("%d", &choice);
} while (choice != 2);
return 0;
}
```

```
#include
<stdio.h>
            #define N 100


            int stack[N];


            int top = -1;


            void push(int item){
                if(top==N-1)
                    printf("Stack overflow!\n");
                else
                    stack[++top] = item;
            }


            int pop(){
                if(top==-1)
                    printf("Stack underflow!\n");
                else
                    return stack[top--];

            }


            int priority(char op){
                switch(op){
                    case '*':   return 2;
                                break;
                    case '/':   return 2;
                                break;
                    case '+':   return 1;
                                break;
```

```
        case '-':   return 1;
                    break;
        case '(':   return 0;
                    break;
    }
}


int main()
{
    char s[50];
    char t[50];
    int l;
    int choice = 1;

    do{
        l = 0;
        printf("Enter your infix expression: ");
        scanf("%s", s);

        for(int i=0; s[i]!='\0'; i++){
            switch(s[i]){
                case '(':   push('(');
                            break;
                case ')':   while(stack[top]!='('){
                                t[l++] = pop();
                            }
                            pop();
                            break;
                case '*':
                case '/':
                case '+':
                case '-':   while(top!=-1 &&
priority(stack[top])>=priority(s[i])){
                                t[l++] = pop();
                            }
```

```
                        push(s[i]);
                        break;
                default: t[l++] = s[i];
            }
        }

        while(top!=-1){
            t[l++] = pop();
        }

        t[l] = '\0';

        printf("Postfix expression for \"%s\" is \"%s\".\n", s, t);

        printf("\n     :: Menu ::     \n1. Try another infix expression.\n2.
Exit\nEnter your choice: ");


        scanf("%d", &choice);
    }while(choice!=2);

    return 0;
}
```

OUTPUT:

```
Enter your infix expression: vv
Postfix expression for "vv" is "vv".

    :: Menu ::
1. Try another infix expression.
2. Exit
Enter your choice: 1
Enter your infix expression: A+B
Postfix expression for "A+B" is "AB+".

    :: Menu ::
1. Try another infix expression.
2. Exit
Enter your choice: 1
Enter your infix expression: A+(B-c)
Postfix expression for "A+(B-c)" is "ABc-+".

    :: Menu ::
1. Try another infix expression.
2. Exit
Enter your choice: 1
Enter your infix expression: (A+(B-C)*D)
Postfix expression for "(A+(B-C)*D)" is "ABC-D*+".

    :: Menu ::
1. Try another infix expression.
2. Exit
Enter your choice: 2
```

3: 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

Date / /
Page
SPLASH

Queue

```
# include <stdio.h>
# include <stdlib.h>
# define Que_size 3.
int item, front=0, rear=-1, q[Que_size];
void insert rear()
{ if( rear = Que_SIZE -1)
  {
     printf("Queue Overflow\n");
     return;
  }
rear = rear+1;
q[rear] = item;
} int deletefront()
{ if ( front > rear)
{ front = 0;
rear = -1
return -1;
} return q[ front++];
} void display Q()
{ int i;
if ( front > rear)
{
    printf ("Queue is empty\n");
    return;
}
  printf ("Contents of queue\n");
  for( i = front; i < rear; i++)
  {
      printf ("%d\n", q[i]);
} }
```

```
int main()
{
    int choice;
    for(;;)
    {
        printf("1) Insert rear\n 2) Delete front
                \n 3) Display \n 4) Exit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("Enter the item
                        to be inserted\n");
            scanf("%d", &item);
            insert rear();
            break;
            case 2: item = deletefront();
            if(item == -1)
            printf("Queue is empty\n");
            else
            printf("Item deleted = %d\n",
                                item);
            break;
            case 3: displayQ();
            break;
            default: exit(0);
        }
    }
}
```

```
#include<stdio.h>
                    #include<stdlib.h>
                    #define QUE_SIZE 3
                    int item,front=0,rear=-1,q[10];
                    void insertrear()
                    {if(rear==QUE_SIZE-1)
                    {
                           printf("Queue Overflow\n");
                           return;
                    }
                    rear=rear+1;
                    q[rear]=item;
                    }int deletefront()
                    {if (front>rear)
                    {front=0;
                    rear=-1;
                    return -1;
                    }return q[front++];
                    }void displayQ()
                    {int i;
                    if (front>rear)
                    {
                           printf("Queue is empty\n");
                           return;
                    }
                    printf("contents of queue\n");
                    for(i=front;i<=rear;i++)
                    {
                           printf("%d\n",q[i]);
                    }}
                    int main()
                    {
                           int choice;
                           for(;;)
                           {
```

```
                printf("1)Insert rear\n2)Delete
front\n3)Display\n4)Exit\n");
                printf("Enter your choice: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:printf("Enter the item to be inserted\n");
                        scanf("%d",&item);
                        insertrear ();
                        break;
                        case 2:item=deletefront();
                        if(item==-1)
                        printf("Queue is empty\n");
                        else
                        printf("Item deleted=%d\n",item);
                        break;
                        case 3:displayQ();
                        break;
                        default:exit (0);

                }

        }
        }
```

OUTPUT:

```
PS C:\Users\It's mine!\Desktop\C C++> cd "c:\Users\It's mine!\Desktop\C C++\.vscode\" ; if ($?) { gcc sss.c -o sss } ; if ($?)
 { .\sss }
1)Insert rear
2)Delete front
3)Display
4)Exit
Enter your choice: 1
Enter the item to be inserted
68
1)Insert rear
2)Delete front
3)Display
4)Exit
Enter your choice: 1
Enter the item to be inserted
67
1)Insert rear
2)Delete front
3)Display
4)Exit
Enter your choice: 2
Item deleted=68
1)Insert rear
2)Delete front
3)Display
4)Exit
Enter your choice: 3
contents of queue
67
1)Insert rear
2)Delete front
3)Display
4)Exit
Enter your choice: 4
PS C:\Users\It's mine!\Desktop\C C++\.vscode>
```

4: 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

Date / /
Page
SPLASH

Circular queue.

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#define que_size 3
int item, front = 0, rear = -1, q[que_size],
count = 0;
void insertrear()
{
    if (count == que_size)
    {
        printf("Queue overflow");
        return;
    }
    rear = (rear + 1) % que_size;
    q[rear] = item;
    count++;
}
int deletefront()
{
    if (count == 0) return -1;
    item = q[front];
    front = (front + 1) % que_size;
    count = count - 1;
    return item;
}
void displayq()
{
    int i, f;
    if (count == 0)
    {
```

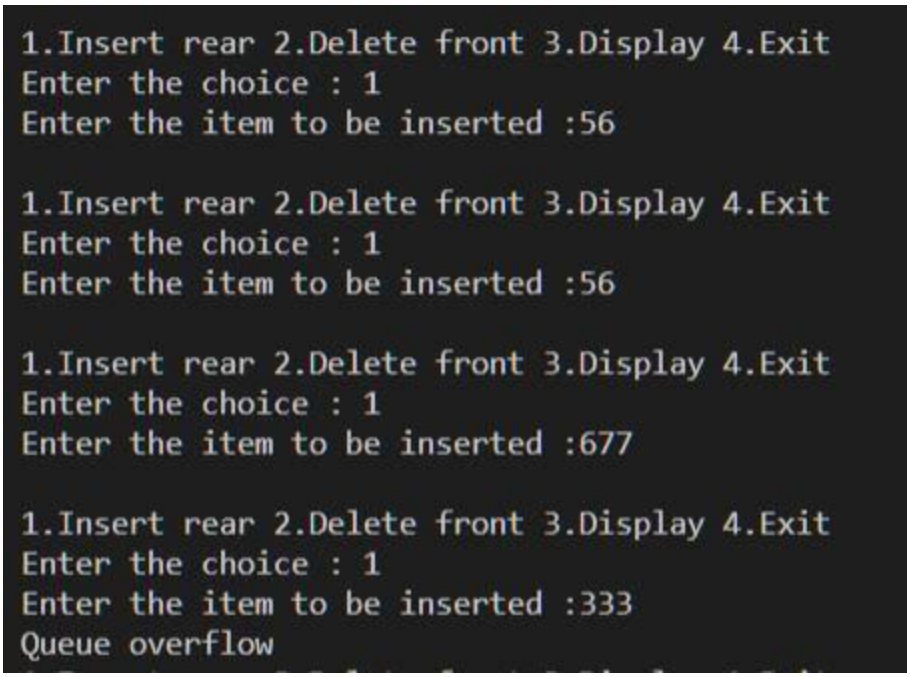
```
        printf("Queue is empty");
        return;
    }
    f = front;
    printf("Contents of queue \n");
    for( i=0; i<=count; i++)
    {
        printf("%d \n", q[f]);
        f = (f+1) % que_size;
    }
}
void main()
{
    int choice;
    for(;;)
    {
        printf("\n1.Insert rear \n2.Delete from
                \n3.display \n4.Exit \n");
        printf("Enter the choice: ");
        scanf("%d", &choice);
        switch(choice)
        {
            case1: printf("Enter the item to
                        be inserted: ");
                scanf("%d", &item);
                insert rear();
                break;
            case2: item=deletefront();
                if(item==-1)
                printf("Queue is empty\n");
                else
                printf("Item deleted is :%d\n
```

```
        break;
    case 3: displayq();
            break;
    default: exit(0);
    }
}
}
```

```
#include<stdio.h>
                    #include<stdlib.h>
                    #include<process.h>
                    #define que_size 3
                    int item,front=0,rear=-1,q[que_size],count=0;
                    void insertrear()
                    {
                            if(count==que_size)
                            {
                                    printf("Queue overflow");
                                    return;
                            }
                            rear=(rear+1)%que_size;
                            q[rear]=item;
                            count++;
                    }
                    int deletefront()
                    {
                            if(count==0) return -1;
                            item = q[front];
                            front=(front+1)%que_size;
                            count=count-1;
                            return item;
                    }
                    void displayq()
                    {
                            int i,f;
                            if(count==0)
                            {
                                    printf("Queue is empty");
                                    return;
                            }
                            f=front;
                            printf("Contents of queue \n");
                            for(i=0;i<=count;i++)
                            {
```

```
                printf("%d\n",q[f]);
                f=(f+1)%que_size;
        }
}
void main()
{
        int choice;
        for(;;)
        {
                printf("\n1.Insert rear 2.Delete front 3.Display 4.Exit
\n");
                printf("Enter the choice : ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:printf("Enter the item to be inserted :");
                               scanf("%d",&item);
                               insertrear();
                               break;
                        case 2:item=deletefront();
                                  if(item==-1)
                                  printf("Queue is empty\n");
                                  else
                                  printf("Item deleted is %d \n",item);
                                  break;
                    case 3:displayq();
                                  break;
                    default:exit(0);
                }
        }
}
```

OUTPUT:



5: 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) Display the contents of the linked list.

6: WAP to Implement Singly Linked List with following operations

a) Create a linked list. b) Deletion of first element, specified element and last element in

the list. c) Display the contents of the linked list.

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

Date __/__/__
Page _____
SPLASH

Singly linked list

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

Struct node
 {
   int info;
    struct node * link;
 };
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 insert-front (NODE first, int item)
{
  NODE temp;
  temp = getnode();
  temp -> info = item;
  temp -> link = NULL;
```

```
if (first == NULL)
return temp;
temp -> link = first;
first = temp;
return first;
}
NODE If (NODE second, int item)
{
NODE temp;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (second = NULL)
return temp;
temp -> link = second;
second = temp;
return second;
}
NODE temp;
if (first == NULL)
{
printf("list is empty cannot delete\n");
return first;
}
temp = first;
temp = temp -> link;
printf("item deleted at front-end
        is %d \n", first -> info);
free (first)
return temp;
}
NODE insert_rear (NODE first, int item)
```

```
{
NODE temp, cur;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (first == NULL)
  return temp;
cur = first;
while (cur -> link != NULL)
     cur = cur -> link;
     cur -> link = temp;
     return first;
}
NODE IR (NODE second, int item)
 {
NODE temp, cur;
temp = get node();
temp -> info = item;
temp -> link = NULL;
if (second = NULL)
   return temp;
  cur = second;
while (cur -> link != NULL)
     cur = cur -> link;
     cur -> link = temp;
     return second;
}
NODE delete_rear (NODE first)
 {
 NODE cur, prev;
 if (first == NULL)
 {
```

```
printf("list is empty cannot delete\n");
return first;
}
if (first -> link == NULL)
{
printf("item deleted is %d\n",
                        first->info);
free (first);
return NULL;
}
prev = NULL;
cur = first;
while (cur -> link != NULL)
{
prev = cur;
cur = cur -> link;
}
printf("item deleted at rear-end
        is %d", cur -> info);
free (cur);
prev -> link = NULL;
return first;
}
NODE insert_pos(int item, int pos,
                        NODE first)
{
NODE temp;
NODE prev, cur;
int count;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
```

```
if (first == NULL && pos == 1)
  return temp;
if (first == NULL)
{
printf ("invalid pos \n");
  return first;
}
if (pos == 1)
{
temp -> link = first;
return temp;
}
count = 1;
prev = NULL;
cur = first;
while (cur != NULL && count != pos)
{
 prev = cur;
  cur = cur -> link;
  count ++;
}
if (count == pos)
{
prev -> link = temp;
temp -> link = cur;
return first;
}
printf ("Invalid position \n");
return first;
}
NODE delete_pos(int pos, NODE first)
{
```

```
NODE cur;
NODE prev;
int count;
if ( first == NULL || pos <= 0)
{
printf ("Invalid position\n");
return NULL;
}
if (pos == 1)
{
cur = first;
first = first -> link;
freenode (cur);
return first;
}
prev = NULL;
cur = first;
count = 1;
while (cur != NULL)
{
if ( count == pos)
break;
prev = cur;
cur = cur -> link;
count++;
}
if (count != pos)
{
  printf ("Invalid position\n");
  return first;
}
if ( count != pos)
```

```
{
printf("Invalid position specified\n");
  return first;
}
prev -> link = cur -> link;
  freenode(cur);
return first;
}
NODE reverse(NODE first)
    {
        NODE cur, temp;
        cur = NULL;
        while(first != NULL)
          {
            temp = first;
            first = first -> link;
            temp -> link = cur;
            cur = temp;
        }
    return cur;
}
NODE asc(NODE first)
{
  NODE prev = first;
  NODE cur = NULL;
            int temp;
if(first == NULL) {
          return 0;
        }
else {
    while(prev != NULL) {
      cur = prev -> link;
```

```
while(cur1 = NULL) {
if (prev -> info > cur -> info) {
    temp = prev -> info;
    prev -> info = cur -> info;
    cur -> info = temp;
    }
    cur = cur -> link;
    }
    prev = prev -> link;
    }
    return first;
}
NODE des (NODE first)
{
  NODE prev = first;
  NODE cur = NULL;
        int temp;
if ( first == NULL) {
        return 0;
    }
    else {
    while (prev1 = NULL) {
    cur = prev -> link;
    while (cur1 = NULL) {
    if (prev -> info < cur->info) {
    temp = prev -> info;
    prev -> info = cur -> info;
    cur -> info = temp;
    }
    cur = cur -> link;
    }
```

```
        prev = prev -> link;
      }
    }
    return first;
  }
NODE concat(NODE first, NODE second)
{
  NODE cur;
  if (first == NULL)
  return second;
  if (second == NULL)
  return first;
  cur = first;
  while (cur -> link != NULL)
  {
    cur = cur -> link;
  }
  cur -> link = second;
  return first;
}

void display (NODE first)
{
NODE temp;
if (first == NULL)
printf ("list empty cannot display
                        item \n");
for (temp = first ; temp != NULL; temp =
                        temp -> link)
{
  printf ("%d \n", temp -> info);
}
```

```
};
int main()
{
int item, choice, pos, element, option,
        choice2, item1, num;
NODE first = NULL;
NODE second = NULL;

for (;;)
{
 printf("\n1: Insert - front
         \n2: Delete front
         \n3: Insert front
         \n4: Delete front
         \n5: random position
         \n6: reverse
         \n7: sort
         \n8: concat
         \n9. display - list
         \n10. Exit \n");
printf("Enter the choice\n");
scanf("%d", &choice\n");
scanf switch(choice)
   {
    case1: printf("Enter the item at
                    front-end\n");
           scanf("%d", &item);
           first= insert_front(first,
                               item);
           break;
    case2: first = delete_front(first);
           break;
```

```
case 3: printf("Enter the item at rear-
                                ent \n");
        scanf("%d", &item);
        first = insert_rear(first, item);
        break;
case 4: first = delete_rear(first);
        break;
case 5:
printf("press 1 to insert or 2 to delete at
        any desired position \n");
scanf("%d", &element);
if(element == 1){
        printf("Enter the position to
                insert \n");
        scanf("%d", &pos);
        printf("Enter the item to insert
                \n");
        scanf("%d", &item);
        first = insert_pos(item, pos, first);
}
if(element == 2){
        printf("Enter the position to delete
                \n");
        scanf("%d", &pos);
        first = delete_pos(pos, first);
}
        break;
case 6:
        first = reverse(first);
        break;
case 7:
        printf("press 1 for ascending sort
```

```
and 2 for descending sort :\n");
scanf("%d", &option);
if(option == 1)
first = asc(first);
if(option == 2)
first = des(first);
break;
case 8:
        printf("Create a second list \n");
        printf("Enter the number of
                elements in second
                list \n");
        scanf("%d", &num);
        for(int i=1; i<=num; i++){
printf("\n press 1 to insert
                front and 2 to insert
                rear \n");
        scanf("%d", &choice2);

        if(choice2==1){
          printf("Enter the item at
                front-end \n");
          scanf("%d", &item1);
        Second = IF(Second, item1);
        }
if(choice2==2){
printf("Enter the item at rear end\n");
    scanf("%d", &item1);
    Second = IR(Second, item1);
}
}
```

```
        first = concate (first, second);
        break;
    case 9: display (first);
            break;
    default : exit(0);
            break;
    }
}
return 0;
}
```

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


                    struct node
                    {
                      int info;
                      struct node *link;
                    };
                    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 insert_front(NODE first,int item)
                    {
                    NODE temp;
                    temp=getnode();
                    temp->info=item;
                    temp->link=NULL;
                    if(first==NULL)
                    return temp;
                    temp->link=first;
                    first=temp;
                    return first;
```

```
}
NODE IF(NODE second,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(second==NULL)
return temp;
temp->link=second;
second=temp;
return second;
}
NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
```

```
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}
NODE IR(NODE second,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(second==NULL)
 return temp;
cur=second;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return second;
}
NODE delete_rear(NODE first)
{
NODE cur,prev;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
if(first->link==NULL)
{
printf("item deleted is %d\n",first->info);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL)
```

```
{
prev=cur;
cur=cur->link;
}
printf("iten deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;
}
NODE insert_pos(int item,int pos,NODE first)
{
NODE temp;
NODE prev,cur;
int count;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL && pos==1)
return temp;
if(first==NULL)
{
 printf("invalid pos\n");
 return first;
}
if(pos==1)
{
temp->link=first;
return temp;
}
count=1;
prev=NULL;
cur=first;
while(cur!=NULL && count!=pos)
{
 prev=cur;
 cur=cur->link;
```

```
 count++;
}
if(count==pos)
{
prev->link=temp;
temp->link=cur;
return first;
}
printf("Invalid Position \n");
return first;
}
NODE delete_pos(int pos, NODE first)
{
NODE cur;
NODE prev;
int count;
if(first==NULL || pos<=0)
{
printf("invalid position \n");
return NULL;
}
if (pos==1)
{
cur=first;
 first=first->link;
freenode(cur);
return first;
}
prev=NULL;
cur=first;
count=1;
while(cur!=NULL)
{
if(count==pos)
break;  //if found
prev=cur;
```

```
cur=cur->link;
count++;
}
if(count!=pos)
{
  printf("invalid position\n");
   return first;
}
if(count!=pos)
{
  printf("invalid position specified\n");
  return first;
}

prev->link=cur->link;
 freenode(cur);
return first;
}
NODE reverse(NODE first)
  {
    NODE cur,temp;
    cur=NULL;
    while(first!=NULL)
    {
        temp=first;
        first=first->link;
        temp->link=cur;
        cur=temp;
    }
    return cur;
}
NODE asc(NODE first)
{
    NODE prev=first;
    NODE cur=NULL;
                int temp;
```

```
if(first== NULL) {
          return 0;
          }
else {
         while(prev!= NULL) {

          cur = prev->link;

          while(cur!= NULL) {

        if(prev->info > cur->info) {
          temp = prev->info;
          prev->info = cur->info;
          cur->info = temp;
           }
           cur = cur->link;
          }
          prev= prev->link;
                    }
               }
               return first;
           }
NODE des(NODE first)
{
    NODE prev=first;
    NODE cur=NULL;
                int temp;

if(first==NULL) {
       return 0;
        }
       else {
         while(prev!= NULL) {

         cur = prev->link;
```

```
          while(cur!= NULL) {

        if(prev->info < cur->info) {
          temp = prev->info;
          prev->info = cur->info;
          cur->info = temp;
              }
              cur = cur->link;
              }
              prev= prev->link;
               }
               }
               return first;
           }
NODE concate(NODE first,NODE second)
{
    NODE cur;
    if(first==NULL)
    return second;
    if(second==NULL)
    return first;
    cur=first;
    while(cur->link!=NULL)
    {
        cur=cur->link;

    }
    cur->link=second;
    return first;
}


void display(NODE first)
{
 NODE temp;
```

```
 if(first==NULL)
 printf("list empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
int main()
{
int item,choice,pos,element,option,choice2,item1,num;
NODE first=NULL;
NODE second=NULL;


for(;;)
{
printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n
4:Delete_rear\n 5:random_position\n 6:reverse\n 7:sort\n 8.concate\n
9:display_list\n 10:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("enter the item at front-end\n");
         scanf("%d",&item);
         first=insert_front(first,item);
         break;
  case 2:first=delete_front(first);
         break;
  case 3:printf("enter the item at rear-end\n");
         scanf("%d",&item);
         first=insert_rear(first,item);
         break;
  case 4:first=delete_rear(first);
         break;
 case 5:
```

```
printf("press 1 to insert or 2 to delete at any desired position \n");
scanf("%d",&element);
if(element==1){
        printf("enter the position to insert \n");
               scanf("%d",&pos);
               printf("enter the item to insert \n");
               scanf("%d",&item);
               first=insert_pos(item,pos,first);
}
if(element==2){
        printf("enter the position to delete \n");
               scanf("%d",&pos);
               first=delete_pos(pos,first);
}
               break;
 case 6:
       first=reverse(first);
       break;
 case 7:
       printf("press 1 for ascending sort and 2 for descending sort:\n");
       scanf("%d",&option);
       if(option==1)
       first=asc(first);
       if(option==2)
       first=des(first);
       break;
 case 8:
       printf("create a second list\n");
       printf("enter the number of elements in second list\n");

       scanf("%d",&num);
        for(int i=1;i<=num;i++){
       printf("\n press 1 to insert front and 2 to insert rear \n");
       scanf("%d",&choice2);

           if(choice2==1){
```

```
                printf("enter the item at front-end\n");
                    scanf("%d",&item1);
                second=IF(second,item1);
                 }

                if(choice2==2){
                printf("enter the item at rear-end\n");
                   scanf("%d",&item1);
                second=IR(second,item1);
                }
              }

             first=concate(first,second);
             break;

  case 9:display(first);
         break;
 default:exit(0);
         break;
 }
}
return 0;
}
```

OUTPUT:

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
34

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
45

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
56

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
67
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
85

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
85
67
56
45
34

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
7
press 1 for ascending sort and 2 for descending sort:
1

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
34
45
56
67
85
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
5
press 1 to insert or 2 to delete at any desired position
2
enter the position to delete
5

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
6

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
67
56
45
34

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
8
create a second list
enter the number of elements in second list
4

 press 1 to insert front and 2 to insert rear
1
enter the item at front-end
2

 press 1 to insert front and 2 to insert rear
1
enter the item at front-end
3
```

```
 press 1 to insert front and 2 to insert rear
1
enter the item at front-end
5

 press 1 to insert front and 2 to insert rear
2
enter the item at rear-end
4

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
67
56
45
34
5
3
2
4

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
10
PS C:\Users\It's mine!\Desktop\C C++\.vscode>
```

8: WAP to implement Stack & Queues using Linked Representation

Date __/__/__
Page _____
SPLASH

Stack and Queues using Linked representation.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
Struct node
{
  int item;
  Struct node *next;
};
typedef struct node * NODE; Node;
Node getNode()
 {
   Node x;
   x = (Node) malloc (sizeof (Struct node));
   return x;
}
Node insert_front (Node first, int data)
{
  Node new_node;
  new_node = getNode();
  new_node -> item = data;
  new_node -> next = NULL;
  if (first = NULL)
  {
    return new_node;
  }
  new_node -> next = first;
  first = new_node;
  return first;
```

```
}
Node insert_end(Node first, int data)
{
 Node last;
 Node new_node;
 new_node = getNode();
 new_node -> item = data;
 new_node -> next = NULL;
 if ( first = NULL)
 {
  return new node;
 }
 last = first;
 while (last -> next != NULL)
 {
   last = last -> next;
 }
 last -> next = new node;
 return first;
}
Node delete_front (Node first)
{
   Node temp;
   if ( first == NULL)
   {
    printf ("List is empty cannot be
              deleted \n");
    return first;
   }
   temp = first;
   temp = temp -> next;
   free (first);
```

```
return temp;
}
Node delete_end (Node first)
{
  Node prev, cur;
  if (first == NULL)
  {
  printf ("List is empty and cannot be
          deleted \n"); return first;
  }
  cur = first;
  while (cur -> next != NULL)
  {
    prev = cur;
    cur = cur -> next;
  }
  prev -> next = NULL;
  free (cur);
  return first;
}
void display (Node first)
{
  Node temp;
  if (first == NULL)
  {
    printf ("List is empty \n");
  }
  for (temp = first; temp != NULL;
       temp = temp -> next)
  {
    printf ("%d \n", temp -> item);
  }
}
```

```
void stack()
{

void stack()
{
 int choice, data;
 Node head = NULL;
printf("Stack is implemented insertion
        rear and delete rear\n");

do {
   printf("1. insert rear\n");
   printf("2. delete rear\n");
   printf("3. Display\n");
   printf("4. Exit\n");
   printf("Enter your choice\n");
   Scanf("%d", &choice);
   Switch(choice)
   {
case 1:
 printf("Enter the Data to be inserted
        \n");
Scanf("%d", &data);
head = insert_end(head, data);
break;
case 2:
 head = delete_end(head);
 break;
case 2:
head = delete_end(head);
break;
printf("Enter the data to be
                inserted\n");
```

```
scanf("%d", &data);
head = insert_end(head, data);
break;
case 2:
head = delete_end(head);
break;
case 3:
display(head);
break;
}

} while(choice != 4);
}

void q()
{
int choice, data;
Node head = NULL;
printf("Queue is implemented
        insert front and delete
        front \n");
do {
printf("1: Insert front \n");
printf("2. Delete front \n");
printf("3. Display \n");
printf("4: Exit \n");
printf("Enter your choice \n");
scanf("%d", &choice);
switch(choice)
{
case 1:
  printf("Enter the data to be
          inserted \n");
```

```
Scanf ("%d", &data);
head = insert_front (head, data);
break;
case 2:
head = delete_front (head);
break;
case 3:
display (head);
break;
}
} while (choice != 4);
}
int main()
{
int option;
printf ("1: Stack \n");
printf ("2: Queue \n");
printf ("3: Exit \n");
printf ("Enter your choice \n");
Scanf ("%d", &option);
Switch (option)
{
case 1:
stack();
break;
case 2:
q();
break;
case 3:
exit (0)
break;
}
}
```

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<string.h>
struct node
{
int item;
struct node *next;
};
typedef struct node *Node;
Node getNode()
{
Node x;
x=(Node)malloc(sizeof(struct node));
return x;
}
Node insert_front(Node first,int data)
{
Node new_node;
new_node=getNode();
```

```
new_node->item=data;
new_node->next=NULL;
if(first==NULL)
{
return new_node;
}
new_node->next=first;
first=new_node;

return first;
}
Node insert_end(Node first,int data)
{
Node last;
Node new_node;
new_node=getNode();
new_node->item=data;
new_node->next=NULL;
if(first==NULL)
{
```

```
return new_node;
}
last=first;
while(last->next!=NULL)
{
last=last->next;
}
last->next=new_node;
return first;
}
Node delete_front(Node first)
{
Node temp;

if(first==NULL)
{
printf("List Is Empty Cannot be deleted\n"); return first;
}
temp=first;
temp=temp->next;
```

```
free(first);
return temp;
}
Node delete_end(Node first)
{
Node prev,cur;
if(first==NULL)
{
printf("List Is Empty And Cannot be deleted\n"); return first;
}
cur=first;
while(cur->next!=NULL)
{
prev=cur;
cur=cur->next;
}
prev->next=NULL;
free(cur);
return first;
}
```

```
void display(Node first)
{
Node temp;
if(first==NULL)
{
printf("List Is Empty\n");
}
for(temp=first;temp!=NULL;temp=temp->next)
{
printf("%d\n",temp->item);
}
}
void stack()
{
int choice,data;
Node head=NULL;
printf("STACK IS IMPLEMENTED INSERT REARE AND DELETE
REAR\n");
do{
printf("1:INSERT REAR\n");
```

```
printf("2:DELETE REAR\n");
printf("3:DISPLAY\n");
printf("4:EXIT\n");
printf("Enter Your Choice\n");
scanf("%d",&choice);
switch(choice)
{


case 1:printf("Enter The Data To Be Inserted\n");
scanf("%d",&data);
head=insert_end(head,data);
break;
case 2:
head=delete_end(head);
break;
case 3:
display(head);
break;
}
```

```
}while(choice!=4);
}
void q()
{
int choice,data;
Node head=NULL;
printf("QUEUE IS IMPLEMENTED INSERT FRONT AND DELETE FRONT\n");
do{
printf("1:INSERT FRONT\n");
printf("2:DELETE FRONT\n");
printf("3:DISPLAY\n");
printf("4:EXIT\n");
printf("Enter Your Choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter The Data To Be Inserted\n");
scanf("%d",&data);
```

```
head=insert_front(head,data);
break;
case 2:
head=delete_front(head);
break;
case 3:
display(head);
break;
}

}while(choice!=4);
}
int main()
{
int option;
printf("1:STACK\n");
printf("2:QUEUE\n");
printf("3:EXIT\n");
printf("Enter Your Choice\n");
scanf("%d",&option);
```

```
switch(option)
{
case 1:
stack();
break;
case 2:
q();
break;
case 3:
exit(0);
break;
}

}
```

OUTPUT:

```
PS C:\Users\It's mine!\Desktop\C C++> cd "c:\Users\It's mine!\Desktop\C C++\.vscode\" ; if ($?) { gcc sss.c -o sss } ; if ($?) { .\sss }
1:STACK
2:QUEUE
3:EXIT
Enter Your Choice
1
STACK IS IMPLEMENTED INSERT REARE AND DELETE REAR
1:INSERT REAR
2:DELETE REAR
3:DISPLAY
4:EXIT
Enter Your Choice
1
Enter The Data To Be Inserted
56
1:INSERT REAR
2:DELETE REAR
3:DISPLAY
4:EXIT
Enter Your Choice
1
Enter The Data To Be Inserted
78
1:INSERT REAR
2:DELETE REAR
3:DISPLAY
4:EXIT
Enter Your Choice
1
Enter The Data To Be Inserted
78
1:INSERT REAR
2:DELETE REAR
3:DISPLAY
4:EXIT
Enter Your Choice
2
1:INSERT REAR
2:DELETE REAR
3:DISPLAY
4:EXIT
Enter Your Choice
3
56
78
1:INSERT REAR
2:DELETE REAR
3:DISPLAY
4:EXIT
Enter Your Choice
4
PS C:\Users\It's mine!\Desktop\C C++\.vscode>
```

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

Date __/__/__
Page _____
SPLASH

Lab-8

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
Struct node
{
   int item;
   struct node *next;
};
typedef struct node * Node;

Node getNode()
{
   Node x;
      x = (Node) malloc (sizeof( struct node));
      return x;
}
Node insert_front (Node first, int data)
{
   Node new_node;
   new_node = getNode();
   new_node -> item = data;
   new_node -> next = NULL;
   if (first == NULL)
   {
      return new_node;
   }
```

```
    new_node -> next = first;
    first = new_node;
    return first;
}
Node delete_end(Node first)
{
    Node prev, cur;
    if (first == NULL)
    {
        printf("List is Empty and
                cannot be deleted\n");
        return first;
    }
    cur = first;
    while (cur -> next != NULL)
    {
        prev = cur;
        cur = cur -> next;
    }
    prev -> next = NULL;
    free (cur);
    return first;
}
void display (Node first)
{
    int count=0;
    Node temp;
    if (first == NULL)
    {
        printf("\n List is Empty\n");
    }
    for (temp = first; temp != NULL; temp
                                = temp -> next)
```

```
{
    count++;
    printf("%d\n", temp->item);
}
printf("\n Number of Nodes
        in the list are
        %d\n", count);
}
void search(Node first, int data)
{
    int pos=0
    Node temp;
    int i;
    if(first == NULL)
    {
    printf("List is empty\n");
    return;
    }
  for(temp=first, i=0; temp!=NULL;
      temp=temp->next, i++)
  {
    if(temp->item == data)
    {
      pos=i+1;
      printf("\n Search Successfull\n");
      printf("Element is found at
              position %d\n", pos);
      break;
    }
    else
    {
      pos=0;
```

```
    }
}
if( pos==0)
{
    printf("\n Search Unsuccessfull \n");
}
}
void sort(Node first)
{
    int t;
    Node temp;
    if( first == NULL)
    {
        printf("List is Empty \n");
        return;
    }
    for(Node i = first; i != NULL; i=i
                                -> Next)
    {
        for( Node j = i -> next; j != Null; j=j
                                -> Next)
        {
            if((i -> item) > (j -> item))
            {
                t = i -> item;
                i -> item = j -> itm;
                j -> item = t;
            }
        }
    }
    printf("\n List in ascending orde
                r is \n");
```

```
}
int main()
{
  Node first = NULL;
  Node a = NULL;
  Node b = NULL;
  Node ans = NULL;
  int choice, val, pos, n;
  do
  {
    printf("**** Actions ****\n");
    printf("1. Insert front\n");
    printf("2. Delete end\n");
    printf("3: Sort the list\n");
    printf("4. Search for an element\n");
    printf("5. Display the count
                and list\n");
    printf("\n Which action do yo
                wanna perform ?\n");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1:
        printf("Enter the value to be
                inserted\n");
        scanf("%d", &val);
        first = insert_front(first, val);
        break;
      Case 2:
        first = delete_end(first);
        break;
```

```
case 3:
    sort (first)
    display (first);
    break;
case 4:
    printf ("Enter the number to
            searched \n");
    scanf ("%d", &n);
    Search (first, n);
    break;
case 5:
    printf ("\n The elements are. \n");
    display (first);
    break;
}
while (choice != 6);
}
```

OUTPUT:

```
PS C:\Users\It's mine!\Desktop\C C++> cd "c:\Users\It's mine!\Desktop
\C C++\.vscode\" ; if ($?) { gcc link2.c -o link2 } ; if ($?) { .\lin
k2 }
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
34
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
45
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
67
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
78
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
99
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
90
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
100
```

```
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
91
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
68
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
5

The Elements are:
68
91
100
90
99
78
67
45
34

Number of Nodes In The List Are 9
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
1
Enter the Value To Be Inserted
666
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit


Which action do you wanna perform?
4
Enter the Number To Searched
34

Search Successfull
Element Is Found At Position 10
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
2
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit
```

```
Which action do you wanna perform?
5

The Elements are:
666
68
91
100
90
99
78
67
45

Number of Nodes In The List Are 9
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
3

List In Ascending order is
45
67
68
78
90
91
99
100
666

Number of Nodes In The List Are 9
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
2
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
5

The Elements are:
45
67
68
78
90
91
99
100

Number of Nodes In The List Are 8
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit

Which action do you wanna perform?
4
Enter the Number To Searched
350

Search Unsuccesfull
****Actions****
1:Insert Front
2:Delete End
3:Sort the List
4:Search For an Element
5:Display the Count and list
6: Exit
```

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.

Date / /
Page
SPLASH

Binary tree

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

Struct node
{
  int info;
  Struct node * r link;
  Struct node * l link;
};
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 insert (NODE root, int item)
{
  NODE temp, cur, prev;
  temp = getnode ();
  temp -> r link = NULL;
```

```
temp -> llink = NULL;
temp -> info = item;
if (root == NULL)
return temp;
prev = NULL;
cur = root;
while (cur != NULL)
{
    prev = cur;
    cur = (item < cur -> info) ? cur
          -> llink : cur -> rlink;
}
if (item < prev -> info)
prev -> llink = temp;
else
prev -> rlink = temp;
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 -> info);
display (root -> llink, i+1);
}
NODE delete (NODE root, int item)
{
NODE cur, parent, q, suc;
```

```
if (root == NULL)
{
    printf ("Empty tree");
    return root;
}
parent = NULL;
cur root;
while (cur != NULL && item != cur
        -> info)
{
    parent = cur;
    cur = ( item < cur -> info ) ? 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 (cur == parent -> llink)
    parent -> llink = q;
else
   parent -> rlink = q;
   freenode (cur);
   return root;
}
void preorder (NODE root)
{
   if ( root != NULL)
   {
      printf ("%d\n", root -> info);
      preorder ( root -> llink );
      preorder ( root -> rlink);
   }
}
void postorder (NODE root)
{
  if (root != NULL)
  {
    postorder ( root -> llink)
    postorder (root -> rlink)
    printf ("%d\n", root -> info);
  }
}
void inorder (NODE root)
{
   if (root != NULL
```

```
        printf("%d\n", root->info);
        inorder(root->rlink);
    }
}
void main()
{
    int item, choice
    NODE root = NULL;
    for(;;)
    {
        printf("\n ---MENU---");
        printf("\n1. Insert 4. Post 2.
                . Display 5. In 3. Pre.
                6. Delete 7. Exit\n");
        printf("Enter your choice");
        switch(choice)
        {
        case1: printf("Enter the item\n");
               scanf("%d", &item);
               root = insert(root, item);
               break;
        case2: display(root, 0);
               break;
        case3: preorder(root);
               break;
        case4: postorder(root);
               break;
        case5: inorder(root);
               break;
        case6: printf("Enter the item\n");
               scanf("%d", &item);
               root = delete(root, item);
```

Date __/__/__
Page _____
SPLASH

```
        break;
    default : exit(0);
              break;
    }
}
}
```

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>
struct node
{
int data;
struct node *left;
struct node *right;
};
typedef struct node *NODE;
NODE getnode(int data)
{
NODE x=(NODE)malloc(sizeof(struct node));
x->data=data;
x->right=NULL;
x->left=NULL;
return x;
}
```

```
NODE insert(NODE root,int info)
{
if(root==NULL)
{
root=getnode(info);
return root;
}
else if(info<=root->data)
{
root->left=insert(root->left,info);
}
else
{
root->right=insert(root->right,info);
}
return root;
}
void preorder(NODE root)
```

```
{
if(root==NULL)
return;
printf("%d\t",root->data);
preorder(root->left);
preorder(root->right);
}
void inorder(NODE root)
{
if(root==NULL)
return;
inorder(root->left);
printf("%d\t",root->data);
inorder(root->right);
}
void postorder(NODE root)
{
if(root==NULL)
return;
```

```
postorder(root->left);
postorder(root->right);
printf("%d\t",root->data);
}
NODE findmin(NODE root)
{
if(root==NULL)
{
return NULL;
}
else if(root->left==NULL)
{
return root;
}
return findmin(root->left);
}
NODE findmax(NODE root)
{
if(root==NULL)
```

```
return NULL;
else if(root->right==NULL)
return root;
return findmax(root->right);
}
NODE delete_node(NODE root,int info)
{
if(root==NULL)
{
return root;
}
else if(info<root->data)
{
root->left=delete_node(root->left,info);
}
else if(info>root->data)
{
root->right=delete_node(root->right,info);
}
```

```
else
if(root->left==NULL&&root->right==NULL)
{
free(root);
root=NULL;
return root;
}
else if(root->left==NULL)
{
NODE temp=root;
root=root->left;
free(temp);
return root;
}
else if(root->right==NULL)
{
NODE temp=root;
root=root->right;
```

```
free(temp);
return root;

}
else
{
NODE temp=findmin(root->right);
root->data=temp->data;
root->right=delete_node(root->right,temp->data);
return root;
}
}
}

void display(NODE root,int i)
{
if(root==NULL)
return;
display(root->right,i+1);
```

```
for(int j=1;j<=i;j++)
printf(" ");
printf("%d\n",root->data);
display(root->left,i+1);

}

int getleafcount(NODE root)
{
NODE current=root;
if(current==NULL)
return 0;
else if(current->right==NULL&&current->left==NULL)
return 1;
else
return (getleafcount(current->left)+getleafcount(current->right));
}
int height(NODE root)
{
```

```
if(root==NULL)
return -1;
int left_height=height(root->left);
int right_height=height(root->right);
if(left_height>right_height)
return left_height+1;
else

return right_height+1;
}
NODE search(NODE root,int key)
{
if(root==NULL)
return NULL;
if(root->data==key)
return root;
else if(key<root->data)
search(root->left,key);
else if(key>root->data)
```

```
search(root->right,key);
else
printf("Search UnSuccessfull\n");
}
NODE inorder_successor(NODE root,int data)
{
if(root==NULL)
{

printf("Tree Empty\n");
return root;
}
NODE current=search(root,data);
//NODE current=root;
if(current==NULL)
return NULL;
if(current->right!=NULL)
{
NODE temp=findmin(current->right);
```

```
}
else
{
NODE successor=NULL;
NODE ancestor=root;
while(current!=ancestor)
{
if(current->data<ancestor->data)
{
successor=ancestor;
ancestor=ancestor->left;
}
else
{
ancestor=ancestor->right;
}
}
return successor;
}
```

```
}
int main()
{
NODE root=NULL;
NODE value=NULL;
int data,option,ans;
do{
printf("1:Insert\n");
printf("2:Delete\n");
printf("3:Preorder\n");
printf("4:PostOrder\n");
printf("5:Inorder\n");
printf("6:Display\n");
printf("7:Height Of Tree\n");
printf("8:Find Maximum\n");
printf("9:Search\n");
printf("10:In Order Successor\n");
printf("11:Exit\n");
printf("Enter Your Choice\n");
```

```
scanf("%d",&option);
switch(option)
{
case 1:
printf("Enter The Data To Be Inserted\n");
scanf("%d",&data);
root=insert(root,data);
break;
case 2:
printf("Enter the Data To Be Deleted\n");
scanf("%d",&data);
root=delete_node(root,data);
break;
case 3:
preorder(root);
printf("\n");
break;
case 4:
postorder(root);
```

```
printf("\n");
break;
case 5:
inorder(root);
printf("\n");
break;
case 6:
display(root,1);
break;
case 7:
ans=height(root);
printf("Height Of Tree Is %d\n",ans);
break;
case 8:
value=findmax(root);
printf("Maximum Value is%d\n",value->data); break;
case 9:
printf("Enter the Key Value To Be Searched\n");
scanf("%d",&data);
```

```
value=search(root,data);
if(root!=NULL)
printf("Search SuccessFull\n");
else
printf("Search Unsuccesfull\n");
break;
case 10:
printf("Enter the key value whose successor you want to find\n");
scanf("%d",&data);
root=inorder_successor(root,data);
printf("Successor is %d\n",root->data);
}
}while(option!=11);
}
```

OUTPUT:


1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
1
Enter The Data To Be Inserted
15
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
1
Enter The Data To Be Inserted
10
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
1
Enter The Data To Be Inserted
25
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
1
Enter The Data To Be Inserted
35
1:Insert
2:Delete
3:Preorder


1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
0
35
25
15
10
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
3
15 10 25 35
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
4
10 35 25 15
1:Insert
2:Delete
3:Preorder
4:PostOrder
5:Inorder
6:Display
7:Height Of Tree
8:Find Maximum
9:Search
10:In Order Successor
11:Exit
Enter Your Choice
5
10 15 25 35
1:Insert
2:Delete
3:Preorder
4:PostOrder