DSA Practical 1

Objective:- To create a dynamic array of any size. Ask the size from the user. Ask all the elements from the user to put it in the selected size array. After that apply all three operations on data structure: Traversal, Insertion and deletion.

Use of Malloc:- Here malloc function is used to dynamically assign memory for taking array size from user. Also the header file for malloc function is “<stdlib>”. Operations used in programming

Operations used in program:-

1) Traversal :- In traversing operation of an array, each element is accessed once and only once.
2) Insertion :- Adding one element element / item / member to an existing data structure at any point is called insertion operation.
3) Deletion :- Removing one element / item / member from an existing data structure from any position is deletion operation.

Program:-

```
#include<stdio.h>
#include<stdlib.h>
void putNumber(int *A,int n){
   for(int j=0;j<n;j++)    scanf("%d",&A[j]);
}
void traversal(int A[],int n){
   for(int j=0;j<n;j++)    printf("%d ",A[j]);
   printf("\n");
}
int insert(int A[],int n,int ene,int elem,int pos){
   int j;
   if(ene<n-1){
       for(j=ene;j>=pos;j--)
           A[j+1]=A[j];
           A[pos]=elem;
           ene=ene+1;
       }
       else
       printf("There is no place to add element\n");
       return ene;
   }
   int deletion(int A[],int n,int ene,int elem){
       int j,pos;
       for(j=0;j<ene;j++){
           if(A[j]==elem){
               pos=j;
```

```
            break;
        }           else pos=-1;        }
    if(pos==-1){
        printf("Element not found\n");
    }
    else{
        for(j=pos;j<ene;j++)
        A[j]=A[j+1];
        ene=ene-1;
    }       return ene;    }
void main(){
    int *a,n,i,ene,elem,pos;
    printf("Enter the size of array\n");        scanf("%d",&n);
    a=(int*)malloc(n*sizeof(int));
    printf("Enter the number of elements\n");       scanf("%d",&ene);
    printf("Enter the %d element in array\n",ene);
    putNumber(a,ene);
    printf("Array after traversal is \n");        traversal(a,ene);
    elem=12,pos=3;
    ene=insert(a,n,ene,elem,pos-1);
    printf("Array after insertion is \n");
    for(i=0;i<ene;i++)
    {printf("%d ",a[i]);}printf("\n");        elem=5;
    ene=deletion(a,n,ene,elem);
    printf("Array after deleting element \n");
    for(i=0;i<ene;i++)       { printf("%d ",a[i]);}
        printf("\n");
   }
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ gcc 1.c -o 1
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./1
Enter the size of array
8
Enter the number of elements
6
Enter the 6 element in array
1 2 3 4 5 6
Array after traversal is
1 2 3 4 5 6
Array after insertion is
1 2 12 3 4 5 6
Array after deleting element
1 2 12 3 4 6
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

DSA Practical 2

Objective:- Implement Binary search on One Dimensional Array. Assume that the array is sorted in ascending order.

Operations used in program:-

sorting:- Arranging the item/elements/member of an existing data structure in a particular order, either in ascending or in descending order is called sorting operation.

Binary searching:- In binary searching, start from comparing the mid value if found search is successful or it will repeat the process till the lowest value will be equal to the high value.

Predefined functions in program:-

malloc function:- The library function malloc is used to allocate a block of memory on the heap. The program accesses this block of memory via a pointer that malloc returns. Here malloc is used to dynamically assign memory files for taking array size from the user. Also the header file for malloc function is “stdlib.h”.

Program:-

```
#include<stdio.h>
#include<stdlib.h>
void enternumber(int *a,int n){
   for(int i=0;i<n;i++)        scanf("%d",&a[i]);
}
int binarysearch(int *a,int n,int k){
   int low=0,high=n-1,mid;
   while(low<=high){
       mid=(low+high)/2;
       if(k==a[mid])
       return 1;
       else
       low=mid+1;
   }
   return 0;
}
void sorting(int a[],int s){
   int i,j,t,flag=1;
   for(i=0;i<s-1;i++)
```

```
        if(flag==1){
            flag=0;
            for(int j=0;j<s-1;j++)
            if(a[j]>a[j+1]){
                t=a[j];
                a[j]=a[j+1];
                a[j+1]=t;
                flag=1;
            }
        }
        else break;
}
void main(){
   int *a,s,i,n,sn;
   printf("Enter the size of the array\n");   scanf("%d",&s);
   a=(int*)malloc(s*sizeof(int));
   printf("Enter the number in array\n");   scanf("%d",&n);
   printf("Enter the element in the array\n");
   enternumber(a,n);
   printf("\n Array before sorting \n");
   for(i=0;i<s;i++)       printf("%d ",a[i]);
   sorting(a,s);
   printf("\n Array after sorting \n");
   for(i=0;i<s;i++)       printf("%d ",a[i]);
   printf("\nEnter the number you want to search\n");  scanf("%d",&sn);
   if(binarysearch(a,s,sn))       printf("Search successful\n");
   else       printf("Search unsuccessful\n");
}
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ gcc 2.c -o 1
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./1
Enter the size of the array
6
Enter the number in array
6
Enter the element in the array
9 3 8 5 5 2

 Array before sorting
9 3 8 5 5 2
 Array after sorting
2 3 5 5 8 9
Enter the number you want to search
4
Search unsuccessful
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

DSA Practical 3

Objective:- Develop the programs using the STACK : Program to convert a decimal number to ternary number.Program to reverse a string, reverse a number.

Stack Operation used in this program :-

push:- Adding item at TOP end (in case of array ,TOP index) check stack full status.Update TOP before insertion (PUSH).

pop:- Deleting an element from TOP end (in case of array ,TOP index) check stack empty stack.(POP) or delete the item (element) then update TOP.

Program 1)

```
#include<stdio.h>
#include<stdlib.h>
void push(int item,int stack[],int size,int *top){
   if(*top==size)  printf("Stack is full\n");
   else *top=*top+1;
   stack[*top]=item;
}
void pop(int stack[],int *top,int size){
   int item,i;
   if(*top==-1)    printf("Stack is empty\n");
   else    item=stack[*top];
   *top=*top-1;
}
void main(){
   int n,num,x,stack[10],top=-1;
   printf("Enter a number to change to ternary ");   scanf("%d",&n);
   num=n;
   while(n!=0){
       x=n%3;       push(x,stack,10,&top);       n/=3;
   }   printf("Ternary of %d is \n",num);
   while(top!=-1){
       pop(stack,&top,10);
       printf("%d ",stack[top+1]);
   }   printf("\n");
}
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./1
Enter a number to change to ternary 19
Ternary of 19 is
2 0 1
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

Program 2)

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void push(char *stack,int size,int *top,char item){
   if(*top==(size-1))  printf("Stack is full\n");
   else *top=*top+1;
   stack[*top]=item;
}
void pop(char *stack,int size,int *top){
   char item;
   if(*top==-1)    printf("Stack is empty\n");
   else    item=stack[*top];
   *top=*top-1;
}
int main(){
   char stack[100],item,st[20];
   int *top,i,l;
   top=(int *)malloc(sizeof(int));
   *top=-1;
   printf("Enter a string\n");
   scanf("%s",&st);
   l=strlen(st);
   for(i=0;i<l;i++){
       item=st[i];
       push(stack,100,top,item);
   }
   for(i=*top;i!=-1;i--){
       printf("%c ",stack[*top]);
       pop(stack,100,top);
   }
   printf("\n");
}
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./1
Enter a string
ShyamTiwari
i r a w i T m a y h S
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

Program 3

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

```c
void push(int *stack,int size,int *top,int item){
   if(*top==size)  printf("Stack is full\n");
   else *top=*top+1;
   stack[*top]=item;
}
void pop(int *stack,int size,int *top){
   int item,i;
   if(*top==-1)    printf("Stack is empty\n");
   else    item=stack[*top];
   *top=*top-1;
}
void main(){
   int x=0,y=0,stack[10],*top,item,n,i;
   top=(int *)malloc(sizeof(int));
   *top=-1;
   printf("Enter a number");
   scanf("%d",&n);
   while(n!=0){
       item=n%10;
       push(stack,10,top,item);
       n=n/10;
   }
   for(i=*top;i!=-1;i--){
       x+=(stack[*top])*pow(10,y);
       pop(stack,10,top);
       y++;
   }
   printf("The reverse of number entered is %d\n",x);
}
```

```
Enter a number12345
The reverse of number entered is 54321
```

DSA Practical 4

Objective:- Implement simple Queue operation without using Global variable for array and index variable.

Queue operation using in this program:-

enqueue:- Restricted insertion operation on Queue is called an enqueue operation(inserting an item at the tail or rear end).

After repeating enqueue operation Queue may reach a state called ‘Queue Full’ or ’Queue overflow’.

dequeue:- Restricted deletion operation on Queue is called a dequeue operation (removing an item from the front end).

After repeating Qdelete operation Queue may reach to start called ‘Queue Empty’ or ‘Queue underflow’.

```
#include<stdio.h>
#include<stdlib.h>
#define max 10
void enqueue(int *queue,int size,int item,int *rear,int *front){
   if(*rear==max-1)
   printf("Queue is full..\n");
   else
       if(*rear==-1){
           *front=0;           *rear=0;
       }
       else           *rear=*rear+1;
       queue[*rear]=item;
}
void dequeue(int *queue,int size,int *rear,int *front){
   int item;
   if(*front==-1)
       printf("Queue empty\n");
       else{           item=queue[*front];
           if(*front==*rear){
               *front=-1;               *rear=-1;           }
           else               *front=*front+1;
       }    }
void main(){
   int queue[max],item,ch,rear=-1,front=-1;
```

```
    printf("Enter 1 to insert\n");    printf("Enter 2 to delete\n");
    printf("Enter 3 to exit\n");
    while(1){
        printf("Enter the choice\n");        scanf("%d",&ch);
        switch(ch){
            case 1:
                printf("enter the number\n");        scanf("%d",&item);
                enqueue(queue,max,item,&rear,&front);
                break;
            case 2:       dequeue(queue,max,&rear,&front);         break;
            case 3:
                exit(0);
                default:{   printf("Enter the valid number\n");    }
    }    }
}
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./
Enter 1 to insert
Enter 2 to delete
Enter 3 to exit
Enter the choice
1
enter the number
2
Enter the choice
1
enter the number
3
Enter the choice
2
Enter the choice
1
enter the number
4
Enter the choice
2
Enter the choice
2
Enter the choice
2
Queue empty
Enter the choice
3
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

DSA Practical 5

Objective:- Design recursive program to find Factorial, to find Greatest common divisor, to generate Fibonacci sequence, Ackermann function (A(m, n)=n+1, if m==0, A(m-1,1) if m>0 && n== 0, A(m-1,A(m,n-1)) if m>0 && n>0), Tower of Hanoi (3 tower (pegs) gam e to transfer the discs from source peg to Destination peg using intermediate peg with condition of smaller disc is always placed above the large disc.

Program 1)

```
#include<stdio.h>
long fact(int n){
   if(n>0)
       return (n*fact(n-1));
   else
       return (1);
}
void main(){
   int n;
   printf("Enter the number for doing factorial \n");
   scanf("%d",&n);
   printf("factorial of %d is %ld\n",n,fact(n));
}
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ gcc 7.c -o 1
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./1
Enter the number for doing factorial
5
factorial of 5 is 120
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

Program 2)

```
#include<stdio.h>
int gcd(int a,int b){
   if(a==b)
       return a;
   if(a%b==0)
        return b;
   if(b%a==0)
        return a;
   if(a>b)
       return (gcd(a%b,b));
   if(b>a) return (gcd(a,b%a));
```

```
}
void main(){
   int a,b;
   printf("Enter the value of a,b\n");
   scanf("%d %d",&a,&b);
   printf("\n GCD is %d\n",gcd(a,b));
}
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./1
Enter the value of a,b
6 9

 GCD is 3
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

Program 3)

```
#include<stdio.h>
int fibonacci(int n){
   if(n==1||n==2)
       return 1;
   return (fibonacci(n-1)+fibonacci(n-2));
}
void main(){
   int n,i;
   printf("Enter the term\n");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
       printf("%d ",fibonacci(i));
   printf("\n%d term is %d \n",n,fibonacci(n));
}
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ gcc 9.c -o 1
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./1
Enter the term
8
1 1 2 3 5 8 13 21
8 term is 21
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

Program 4)

```
#include<stdio.h>
int Ackerman(int m,int n){
   if(m==0)
    return n+1;
   else
       if(m>0&&n==0)
```

```
          return Ackerman(m-1,1);
        else
            if(m>0&&n>0)
                return Ackerman(m-1,Ackerman(m,n-1));
}
int main(){
   int x,a,b;
   printf("Enter two number\n");
   scanf("%d %d",&a,&b);
   x=Ackerman(a,b);   printf("%d\n",x);
}
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ gcc 10.c -o 1
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./1
Enter two number
2 3
9
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

Program 5)

```
#include<stdio.h>
void TOH(int n,char beg,char aux,char end){
   if(n>=1){
       TOH(n-1,beg,end,aux);
       printf("move disk %d from %c to %c \n",n,beg,end);
       TOH(n-1,aux,beg,end);
   }
}
void main(){
   int n;char a,b,c;
   printf("Enter the number of disk \n");
   scanf("%d",&n);
   TOH(n,'a','b','c');
}
```

```
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$ ./1
Enter the number of disk
3
move disk 1 from a to c
move disk 2 from a to b
move disk 1 from c to b
move disk 3 from a to c
move disk 1 from b to a
move disk 2 from b to c
move disk 1 from a to c
shyam@shyam-HP-Laptop-15-da0xxx:~/Desktop/dsa$
```

Dsa Practical 6

Objective:- Write functions and programs to implement Singly Linked List and Traversal, searching, Insertion and Deletion operations. Implement linked Stack and Linked Queue.

Operations used in this Program:-

1.) Traversal:- In traversing operation of an array, each element is accessed once and only for one.

2.) Insertion:- Adding one element/item/member to an existing data structure any point is called as insertion operation.

3.) Deletion:- Removing one element/item/member from an existing data structure from any position is deletion operation.

Program 1

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data; struct node *next;
};
struct node *head, *temp = NULL;
void create(){
   struct node *p = (struct node *)malloc(sizeof(struct node));
   int data;printf("Enter the element\n");
   scanf("%d", &data); p->data = data;
   if (head == NULL) {
       head = temp = p;  temp->next = NULL;
   }  else  {
       temp->next = p;  temp = p; temp->next = NULL;
   }}
void display(){
   struct node *p = head;
   if (head == NULL) {
       printf("List is empty\n");
   }
   printf("Nodes of singly linked list: \n");
   while (p != NULL)    {
```

```
        printf("%d ", p->data); p = p->next;
    }    printf("\n");  }
void delete (){
    struct node *p = head; int pos;
    printf("enter the position to be deleted\n");
    scanf("%d", &pos);
    if (pos != 1) {
        while (pos != 2)  {
            p = p->next;    pos--;
        }       if (p->next->next == NULL)  {
            temp = p;  temp->next = NULL;
        }       else {
            p->next = p->next->next;
        }   }   else {
        p = head; p = p->next; head = p;
    }  }
void insert(){
    struct node *p = (struct node *)malloc(sizeof(struct node));
    int pos, element;
    printf("Enter the position in which element is to be inserted\n");
    scanf("%d", &pos);
    printf("Enter the element to be inserted\n");
    scanf("%d", &element);
    p->next = NULL;
    p->data = element;
    if (pos == 1)    {
        p->next = head;    head = p;
    }   else    {
        temp = head;
        while (pos > 2) {
            temp = temp->next;     pos--;
        }     p->next = temp->next; temp->next = p;   }}
void main(){
    temp = (struct node *)malloc(sizeof(struct node));
    int option = 0, options;
    while (option == 0)  {
        printf("Enter 1 to create a linked list\n");
        printf("Enter 2 to display a linked list\n");
        printf("Enter 3 to delete a value from linked list\n");
        printf("Enter 4 to insert a value in linked list\n");
        printf("Enter 5 to quit\n");
        scanf("%d", &options);
        switch (options)  {
```

```
        case 1:  {  create();  break; }
        case 2: { display();   break;  }
        case 3:  {  delete ();     break;     }
        case 4:  {   insert();    break;     }
        case 5:  {    option=1;      break;       }
        default:{ printf("Invalid input\n");  }    }  } }
```

Operations used in this Program:-

1.) PUSH:- Adding item at Top end in Linked stack Update Top before insertion (PUSH)

2.) POP:- Deleting an element from Top end check Linked stack empty stack POP or delete the item (element) then Update Top.

```
#include<stdio.h>
#include<stdlib.h>
struct node{    int data; struct node *next;    };
struct node *temp,*head=NULL;
void push(){
   struct node *p=(struct node *)malloc(sizeof(struct node));
   int element;   printf("Enter an element to be inserted\n");
   scanf("%d",&element);   p->next=NULL;   p->data=element;
   if(head==NULL){  temp=p;  head=p;   temp->next=NULL;   }
   else{       temp->next=p;  temp=p; temp->next=NULL;   }}
void pop(){   struct node *p=head;
   if(temp==head){ temp=NULL;   head=NULL;    }
   else{ while(p->next->next!=NULL){ p=p->next;  }
       temp=p;  temp->next=NULL;     } }
void display(){   struct node *p=head;   while(p!=NULL){
       printf(" << %d ",p->data);  p=p->next;   }   printf("\n");}
int main(){   int option,options=0;   while(options==0){
```

```
        printf("\nEnter 1 to push an element in stack\n");
        printf("Enter 2 to pop an element in stack\n");
        printf("Enter 3 to display elements of stack\n");
        printf("Enter 4 to exit the program\n");
        scanf("%d",&option);
        switch(option){
            case 1:{  push();  break;   }
            case 2:{   pop();    break; }
            case 3:{    display();    break; }
            case 4:{    options++;     break;}
            default:{    printf("Invalid input\n"); }  }  }  }
```

Operation used in this Program:-

1.) Enqueue:- Restricted insertion operation on Queue is called an enqueue operation (inserting an item at the tail or rear end).

2.) Dequeue:- Restricted deletion operation on Queue is called as dequeue operation (removing an item from the front end).After repeating dequeue operation Queue may reach to start called ‘Queue Empty’ or ‘ queue underflow’.

Program 3)

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;
   struct node *next;
};
struct node *temp,*head=NULL;
void enqueue(){
   struct node *p=(struct node *)malloc(sizeof(struct node));
```

```
    int element;
    printf("Enter an element to be inserted\n");
    scanf("%d",&element);
    p->next=NULL;
    p->data=element;
    if(head==NULL){
        temp=p;
        head=p;
        temp->next=NULL;
    }
    else{
        temp->next=p;
        temp=p;
        temp->next=NULL;
    }
}
void dequeue(){
    struct node *p=head;
    if(temp==head){
        temp=NULL;
        head=NULL;
    }
    else{
            head=p->next;
        }
}
void display(){
```

```
    struct node *p=head;
    while(p!=NULL){
        printf(" << %d ",p->data);
        p=p->next;
    }   printf("\n");}
int main(){
    int option,options=0;
    while(options==0){
        printf("\nEnter 1 to enqueue an element in stack\n");
        printf("Enter 2 to dequeue an element in stack\n");
        printf("Enter 3 to display elements of stack\n");
        printf("Enter 4 to exit the program\n");
        scanf("%d",&option);
        switch(option){
            case 1:{
                enqueue();          break;
            }
            case 2:{
                dequeue();              break;
            }
            case 3:{
                display();              break;
            }
            case 4:{
                options++;              break;
            }
            default:{               printf("Invalid input\n");
```

```
}        }    }}
```

Dsa Practical 7

Objective:- Write a C Program to implement Doubly Linked List and Circular Linked List.

Operations use d in this Program:-

1.) Traversal:- In traversing operation of an Doubly Linked List, each element is accessed once and only for one.

2.) Insertion:- Adding one element/item/member to an existing data structure any point is called as insertion operation.

3.) Deletion:- Removing one element/item/member from an existing data structure from any position is deletion operation.

Program 1)

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data;   struct node *prev;   struct node *next;};
struct node *head, *temp = NULL;
void create(){
   struct node *p = (struct node *)malloc(sizeof(struct node));
   int data;
   printf("Enter the element\n");   scanf("%d", &data);   p->data = data;
   if (head == NULL)   {
       head = temp = p;       head->prev = NULL;       temp->next = NULL;
   }   else   {       temp->next = p;
       p->prev = temp;       temp = p;       temp->next = NULL;   }}
void display(){   struct node *p = head;
   if (head == NULL)   {       printf("List is empty\n");   }
   printf("Nodes of doubly linked list: \n");
```

```
    while (p != NULL)   {       printf("%d ", p->data);
        p = p->next;   }   printf("\n");   p = temp;
    while (p != NULL)   {
        printf("%d ", p->data);       p = p->prev;
    }   printf("\n");}
void search(){
    struct node *p = head;   int element, count = 0;
    printf("Enter the value\n ");   scanf("%d", &element);
    while (p != NULL)   {
        if (p->data == element)       {
            printf("Element is present\n");
            count++;       }       p = p->next;   }
    if (count == 0)   {       printf("Element is not present\n");   }}
void delete (){
    struct node *p = head;   int pos;
    printf("enter the position to be deleted\n");   scanf("%d", &pos);
    if (pos != 1)   {
        while (pos != 2)       {
            p = p->next;   pos--;
        }       if (p->next->next == NULL)       {
            temp = p;    temp->prev = p->prev;     temp->next = NULL;
        }       else       {
            p = p->next; p->prev->next = p->next; p->next->prev = p->prev;
        }
    }   else   {
        p = head;    p = p->next;    p->prev = NULL;   head = p;   }}
void insert(){
    struct node *p = (struct node *)malloc(sizeof(struct node));
    struct node *temporary = (struct node *)malloc(sizeof(struct node));
    int pos, element;
    printf("Enter the position in which element is to be inserted\n");
    scanf("%d", &pos);
    printf("Enter the element to be inserted\n");   scanf("%d", &element);
```

```
   p->data = element;   p->next = NULL;   p->prev = NULL;
   if (pos == 1)   {
       p->next = head;       head->prev = p;       head = p;   }
   else   {       temporary = head;
       while (pos > 2)       {
           temporary = temporary->next;           pos--;       }
       p->next = temporary->next;       (temporary->next)->prev = p;
       temporary->next = p;       p->prev = temporary;
   }}
void main(){
   temp = (struct node *)malloc(sizeof(struct node));
   int option = 0, options;
   while (option == 0) {
       printf("Enter 1 to create a linked list\n");
       printf("Enter 2 to display a linked list\n");
       printf("Enter 3 to search a value from linked list\n");
       printf("Enter 4 to delete a value from linked list\n");
       printf("Enter 5 to insert a value in linked list\n");
       scanf("%d", &options);
       switch (options){
       case 1:{ create(); break;}
       case 2: { display(); break;  }
       case 3:{ search(); break; }
       case 4:{ delete (); break; }
       case 5: {  insert();  break;       }       }    }      }
```

Program 2)

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
      int data;
      struct node *next;
};
```

```
struct node *head, *temp = NULL;
void create(){
      struct node *p = (struct node *)malloc(sizeof(struct node));
      int data;
      printf("Enter the element\n"); scanf("%d", &data);      p->data =
data;
      if (head == NULL)  {
      head = temp = p;         temp->next = NULL;
      }     else  {
      temp->next = p;          temp = p;
      temp->next = NULL; }     temp->next = head;
}void display(){
      struct node *p = head;
      if (head == NULL)  {     printf("List is empty\n");}
      printf("Nodes of circular linked list: \n");
      do    {
      printf("%d ", p->data);        p = p->next;
      } while (p != head);     printf("\n");}
void search(){
      struct node *p = head;   int element, count = 0;
      printf("Enter the value\n ");  scanf("%d", &element);
      do    {     if (p->data == element)
      {       printf("Element is present\n");          count++;     }
      p = p->next;} while (p != head);
      if (count == 0)    {     printf("Element is not present\n");  }}
void delete (){
      struct node *p = head;   int pos;
      printf("enter the position to be deleted\n");    scanf("%d", &pos);
      if (pos != 1)      {
      while (pos != 2)         {           p = p->next;            pos--;
      }
      if (p->next->next == head)     {
            temp = p;          temp->next = head;
```

```
	}	else	{	p->next = p->next->next;	}	}
	else	{	p = head;	p = p->next;
	head = p;	temp->next = head; }}
void insert(){
	struct node *p = (struct node *)malloc(sizeof(struct node));
	int pos, element;
	printf("Enter the position in which element is to be inserted\n");
	scanf("%d", &pos);
	printf("Enter the element to be inserted\n");
	scanf("%d", &element);	p->next = NULL;	p->data = element;
	if (pos == 1)	{
	p->next = head;	head = p;	temp->next = head;
	}	else	{
	temp = head;	while (pos > 2)
	{	temp = temp->next;	pos--;
	}	p->next = temp->next;	temp->next = p;	}}
void main(){
	temp = (struct node *)malloc(sizeof(struct node));
	int option = 0, options;
	while (option == 0)	{
	printf("Enter 1 to create a linked list\n");
	printf("Enter 2 to display a linked list\n");
	printf("Enter 3 to search a value from linked list\n");
	printf("Enter 4 to delete a value from linked list\n");
	printf("Enter 5 to insert a value in linked list\n");
	scanf("%d", &options);
	switch (options)	{
	case 1:	{	create();	break;	}
	case 2:	{	display();	break;	}
	case 3:	{	search();	break;	}
	case 4:	{	delete ();	break;	}
	case 5:	{	insert();	break;
		}
```

```
    }
                }
    }
```

DSA Practical 8

Objective:-

Write a C Program to implement “Binary Search Tree Operations-Insertion, Deletion, Searching, Sorting”.

Operation used in this program:-

1.) BST Insertion:- Inserting data one by one in Binary search Tree.

2.) BST Deletion:- Deleting data anywhere from the Binary search Tree.

3.) BST Searching:- Searching data from the Binary Search Tree if found print data is found or if not found print data not found.

4.) BST Sorting:- Sorting data in ascending order from Binary Search Tree.

```
#include <stdio.h>
#include <stdlib.h>
struct node{   int data;   struct node *left, *right;};
struct node *newNode(int item){
   struct node *temp = (struct node *)malloc(sizeof(struct node));
   temp->data = item;
   temp->left = temp->right = NULL;   return temp;}
struct node *insert(struct node *node, int data){
   if (node == NULL)       return newNode(data);
   if (data < node->data)
       node->left = insert(node->left, data);
   else if (data > node->data)
       node->right = insert(node->right, data);   return node;}
void inorder(struct node *root){
   if (root != NULL)   {
       inorder(root->left);       printf("<< %d ", root->data);
       inorder(root->right);   }   }
void preorder(struct node *root){
```

```
    if (root != NULL)    {
        printf("<< %d ", root->data);        preorder(root->left);
        preorder(root->right);    } }
void postOrder(struct node *root){
    if (root != NULL)    {postOrder(root->left);
postOrder(root->right);printf("<< %d ", root->data);    }        }
int search(struct node *root, int data){
    if (root != NULL)  {search(root->left, data);
    search(root->right, data);
        if (data == root->data){
            printf("\nElement present\n");       }    }      }
int main(){
    struct node *root = NULL;    int option = 0, options, data;
    while (option == 0)    {
        printf("\nEnter 1 to insert a value in tree\n");
        printf("Enter 2 for inorder traversal\n");
        printf("Enter 3 for preorder traversal\n");
        printf("Enter 4 for postorder traversal\n");
        printf("Enter 5 to search an element\n");
        printf("Enter 6 to exit an element\n");
        scanf("%d", &options);
        switch (options)        {        case 1:        {
            printf("Enter a value to be inserted\n");
            scanf("%d", &data);     root = insert(root, data);
            insert(root, data);           break;        }
        case 2:  {
            printf("Inorder traversal output is \n");
            inorder(root);    break;        }
        case 3: {     printf("\nPreorder traversal output is \n");
            preorder(root);          break;        }
        case 4:{     printf("\nPostorder traversal output is \n");
            postOrder(root);           break;        }
```

```
        case 5:{printf("Enter value to searched\n");scanf("%d", &data);
        search(root, data);break;}case 6:{option = 1;break;}
        default:{printf("Invalid input\n");}}}}
```

DSA Practical 9

Objective:-

Write a C Program to implement “Binary Search Tree Operations Insertion, Deletion, Searching, Sorting.

Operation used in this program:-

1.) BST Insertion:- Inserting data one by one in Binary search Tree.

2.) BST Deletion:- Deleting data anywhere from the Binary search Tree.

3.) BST Searching:- Searching data from the Binary Search Tree if found print data is found or if not found print data not found.

4.) BST Sorting:- Sorting data in ascending order from Binary Search Tree.

```
#include<stdio.h>
#include<stdlib.h>
struct Tnode{   int data;   struct Tnode *right,*left;};
struct Tnode *root=NULL;
void sorting(struct Tnode*root){
   if(root!=NULL){
        sorting(root->left);
        printf("%d->",root->data);
   }
}
struct Tnode *create(int da){
   struct Tnode *new;
   new=(struct Tnode*)malloc(sizeof(struct Tnode));
   new->data=da;
   new->left=NULL;
   new->right=NULL;
   return new;
```

```
};
struct Tnode *Insert(struct Tnode *root,int da){
   if(root==NULL)
   root=create(da);
   else
   if(da<=root->data){
       root->left=Insert(root->left,da);
   }
   else{
       root->right=Insert(root->right,da);
   }
   return root;
}
int FindMin(struct Tnode *root){
   int R,LR,RR;
   if(root==NULL)
   return 0
   R=root->data;
   LR=FindMin(root->left);
   RR=FindMin(root->right);
   if(LR<R)
   r=LR;
   if(RR<R)
   R=RR;
   return R;
}
struct Tnode*Delete(struct Tnode *root,int da){
   struct Tnode,*temp;
   if(root==NULL)
   return root;
   else
   if(da<root->data)
   root->left=Delete(root->left,da);
```

```
    else
    if(root->left==NULL&&root->right==null){
        free(root);
        root=NULL;
    }
    else
        if(root->right==NULL){
            temp=root;
            root=root->right;
            free(temp);
        }
    else
        if(root->left==NULL){
            temp=root;
            root=root->left;
            free(temp);
        }
        else{
            temp=FindMin(root->right);
            root->data=temp->data;
            root->right=Delete(root->right,temp->data);
        }
        return root;
};
struct Tnode *binarysearch(struct Tnode *root,int da){
    if(root==NULL){
        printf("%d is not found",da);
        return;
    }
    else
    if(root->data==da){
        printf("Element %d is found",da);
        return root;
```

```
    }
    else{
        if(root->data>da){           return binarysearch(root->left,da);
}
        else{    return binarysearch(root->right,da);      }
    }
};
void main(){
    int a,da;
    printf("Enter 1 for insertion \n");
    printf("Enter 2 for deletion"\n);
    printf("Enter 3 for searching in tree\n");
    printf("Enter 4 for sorting");
    printf("Enter 5 to exit\n");
    while(1){
        printf("Enter the choice\n");       scanf("%d",&a);
        switch(a){
            case 1:{
                root=NULL;               while(1){
                    printf("Enter data to add to tree and -1 to stop\n");
                    scanf("%d",&da);
                    if(da==-1) break;
                    root=Insert(root,da);
                }               break;           }
            case 2:{
                printf("Enter data to delete:");          scanf("%d",&da);
                Delete(root,da);               break;           }
            case 3:{
                printf("Enter element to search from tree\n");
                scanf("%d",&da);   binarysearch(root,da);     break;       }
            case 4:{  printf("Sorting\n");    sorting(root);      break;  }
            case 5:{     exit(0);           }
            default:{    printf("Enter the correct key\n");    break;} }   }}
```