SHRI SHANKARACHARYA GROUP OF INSTITUTIONS

FACULTY OF ENGINEERING AND TECHNOLOGY

CERTIFICATE

THIS IS TO CERTIFY THAT THIS PRACTICAL RECORD CONTAINS THE BONAFIDE PRACTICAL WORK FOR THE SUBJECT

"DATA STRUCTURES LAB MANUAL USING'C' "

SHUBHAM BHOJWANI

DURING THE ACADEMIC SESSION 2018-2019
OF 4th SEMESTER SECTION "C"

ROLL NO. 12
DATE :24/04/2019

SIGNATURE OF HOD SIGNATURE OF LECTURER

Sr no.	EXPERIMENT NAME	EXP. DATE	SUBMISSION DATE	SIGNATURE
1	Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).	22/01/19	29/01/19	
2	Write a program to implement stack and perform push, pop operation.	29/01/19	5/02/19	
3	Write a program to convert Infix expression to postfix expression using stack	29/01/19	5/02/19	
4	Write a program to perform the following operations in linear queue – addition, deletion, and traversing.	5/02/19	12/02/19	
5	Write a program to perform the following operations in circular queue – addition, deletion, and traversing	5/02/19	12/02/19	
6	Write a program to perform the following operations in singly linked list – creation, insertion, and deletion.	12/02/19	19/02/19	
7	Write a program to perform the following operations in singly linked list – creation, insertion, and deletion.	19/02/19	26/02/19	
8	Write a program to perform the following operations in doubly linked list – creation, insertion, and deletion	26/02/19	5/03/19	
9	Write a program to implement polynomial in linked list and perform the following a. Arithmetic. b. Evaluation.	5/03/19	12/03/19	
10	Write programs to implement linked stack and linked queue.	12/03/19	26/03/19	
11	Write programs to perform Insertion sort, Selection sort, and Bubble sort.	26/03/19	2/04/19	

12	Write a program to perform Quick sort	2/04/19	9/04/19	
13	Write a program to perform Merge sort.	2/04/19	9/04/19	
14	Write a program to perform Heap sort.	9/04/19	16/04/19	
15	Write a program to create a binary search tree and perform – insertion, deletion, and traversal.	9/04/19	16/04/19	
16	Write a program for traversal of graph (B.F.S., D.F.S.).	16/04/19	16/04/19	

DATA STRUCTURES LAB MANUAL USING‘C’

LIST OF EXPERIMENTS

1) Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).
2) Write a program to implement stack and perform push, pop operation.
3) Write a program to convert Infix expression to postfix expression using stack.
4) Write a program to perform the following operations in linear queue – addition, deletion, and traversing.
5) Write a program to perform the following operations in circular queue – addition, deletion, and traversing.
6) Write a program to perform the following operations in double ended queue – addition, deletion, and traversing.
7) Write a program to perform the following operations in singly linked list – creation, insertion, and deletion.
8) Write a program to perform the following operations in doubly linked list – creation, insertion, and deletion.
9) Write a program to implement polynomial in linked list and perform the following
    c. Arithmetic.
    d. Evaluation.
10) Write programs to implement linked stack and linked queue.
11) Write programs to perform Insertion sort, Selection sort, and Bubble sort.
12) Write a program to perform Quick sort.
13) Write a program to perform Merge sort.
14) Write a program to perform Heap sort.
15) Write a program to create a binary search tree and perform – insertion, deletion, and traversal.
16) Write a program for traversal of graph (B.F.S., D.F.S.).

EXPERIMENT No.1 (a)

Aim:- Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).

Theory:

1. Locate the position where the element in to be inserted (position may be user-specified in case of an unsorted list or may be decided by search for a sorted list).
2. Reorganize the list and create an 'empty' slot.
3. Insert the element.

Example: (Sorted list)

Data:	345	358	490	501	513	555	561	701	724	797
Location:	0	1	2	3	4	5	6	7	8	9

Insert 505 onto the above list:

1. Locate the appropriate position by performing a binary search. 505 should be stored in location 4.
2. Create an 'empty' slot

Data:	345	358	490	501	513	555	561	701	724	797	
Location:	0	1	2	3	4	5	6	7	8	9	10

3. Insert 505

Data:	345	358	490	501	505	513	555	561	701	724	797
Location:	0	1	2	3	4	5	6	7	8	9	10

Source Code:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 20
/********** Function Declaration begins **********/
int insert(int[],int,int,int);
void traverse(int[],int);
/********** Function Declaration ends **********/
void main()
{
        int i=0,A[SIZE],n,pos,item;
        clrscr();
        printf("\n\n\t\t Program to insert element in 1-Dimensional array: ");
        printf("\n\n\t\tHow many number you want to store in the array: ");
        scanf("%d",&n);
        while(i<n)
        {
                printf("\n Enter value A[%d]: ",i);
```

```
                scanf("%d",&A[i]);
                i++;
        }
        traverse(A,n);
        printf("\nEnter the index to insert new number: ");
        scanf("%d",&pos);
        printf("\nEnter the number: ");
        scanf("%d",&item);
        n = insert(A,n,pos,item);
        traverse(A,n);
        getch();
}
/********** Traversing array elements **********/
/********** Function Definition begins **********/
void traverse(int A[], int n)
{
        int i=0;
        printf("\n\n\t\t elements of array are:\n");
        while(i<n)
        {
                printf("A[%d]: ",i);
                printf("%d\n",A[i]);
                i++;
        }
        printf("\n");
}
/********** Function Definition ends **********/

/********** inserting array element **********/
/********** Function Definition begins **********/
int insert(int A[], int n, int pos, int item)
{
        int i;
        for(i=n;i>=pos;i—)
                A[i+1] = A[i];
        A[pos] = item;
        n= n+1;
        return n;
}
/********** Function Definition ends **********/
```

Output:

```
Program to insert an element from 1-Dimensional array:
How many number you want to store in the array:6

Enter value A[0]: 11
Enter value A[1]: 22
Enter value A[2]: 33
Enter value A[3]: 44
Enter value A[4]: 55
Enter value A[5]: 66
                elements of array are:
A[0]: 11
A[1]: 22
A[2]: 33
A[3]: 44
A[4]: 55
A[5]: 66

Enter the index to insert new number: 3
Enter the number: 88
                elements of array are:
A[0]: 11
A[1]: 22
A[2]: 33
A[3]: 88
A[4] :44
A[5]: 55
A[6]: 66
```

EXPERIMENT No.1 (b)

Aim:- Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).

Theory:

1. Locate the element in the list (this involves searching).
2. Delete the element.
3. Reorganize the list and index.

Example:

Data:	345	358	490	501	513	555	561	701	724	797
Location:	0	1	2	3	4	5	6	7	8	9

Delete 358 from the above list:

1. Locate 358: If we use 'linear search', we'll compare 358 with each element of the list starting from the location 0.
2. Delete 358: Remove it from the list (space=10).

Data:	345	490	501	513	555	561	701	724	797
Location:	0	1	2	3	4	5	6	7	8	9

3. Reorganize the list: Move the remaining elements. (Space=9)

Data:	345	490	501	513	555	561	701	724	797	? (797)
Location:	0	1	2	3	4	5	6	7	8	9

Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define SIZE 20
/********** Function Declaration begins **********/
int deletion(int[],int,int);
void traverse(int[],int);
/********** Function Declaration ends **********/

void main()
{
        int i=0,A[SIZE],n,pos;
        clrscr();
        printf("\n\n\t\t Program to delete an element from 1-Dimensional array: ");
        printf("\n\n\t\t How many number you want to store in the array: ");
        scanf("%d",&n);
        while(i<n)
        {
```

```
                printf("\nEnter value A[%d]: ",i);
                scanf("%d",&A[i]);
                i++;
        }
        traverse(A,n);
        printf("\nEnter the index for deleting the number: ");
        scanf("%d",&pos);
        n = deletion(A,n,pos);
        traverse(A,n);
        getch();
}

/********** Traversing array elements **********/
/********** Function Definition begins **********/
void traverse(int A[], int n)
{
        int i=0;

        while(i<n)
        {
                printf("\n A[%d]:",i);
                printf("%d\n",A[i]);
                i++;
        }
        printf("\n");
}
/********** Function Definition ends **********/
/********** Deleting array element **********/
/********** Function Definition begins **********/
int deletion(int A[], int n, int pos)
{
        int item;

        item = A[pos];
        printf("Deleted item from the index %d is :%d\n",pos,item);
        while(pos<=n)
        {
                A[pos] = A[pos+1];
                pos++;
        }
        n= n-1;
        return n;
}
/********** Function Definition ends **********/
```

Output:

```
        Program to delete an element from 1-Dimensional array:
        How many number you want to store in the array: 6
Enter value A[0]: 11
Enter value A[1]: 22
Enter value A[2]: 33
Enter value A[3]: 44
Enter value A[4]: 55
Enter value A[5]: 66
 A[0]: 11
 A[1]: 22
 A[2]: 33
 A[3]:44
 A[4]:55
 A[5]:66
Enter the index for deleting the number: 3
Deleted item from the index 3 is: 44
 A[0]:11
 A[1]:22
 A[2]:33
 A[3]:55
 A[4]:66
```

EXPERIMENT No.1(c) (Linear search)

Aim:- Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).

Theory:

In this algorithm in the set of 'N' data item is given—D_1, D_2 .... D_n having k_1, k_2 .... k_N, 'N' distinct respective keys. If the desired record is located that contains the key 'k_i' then the search is successful otherwise unsuccessful. We assume that $N \leftarrow 1$.

Step 1 Initialization
Set i ← 1.
Step 2 Loop, Comparison
while (i < = N)
{
if (k = k_i) then
{
message : ''successful search''
display (k) go to step 4
}
else
Set i ←i + 1
}
End of loop.
Step 3 If no match
If (k ← k_i) then
message : "unsuccessful search".
Step 4 Finish
Exit.

Source Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100],n,i,item,loc=-1;
clrscr();
```

```
printf("\nEnter the number of element:");
scanf("%d",&n);
printf("Enter the number:\n");
for(i=0;i<=n-1;i++)
  {
   scanf("%d",&a[i]);
  }
printf("Enter the no. to be search\n");
scanf("%d",&item);
for(i=0;i<=n-1;i++)
  {
   if(item==a[i])
     {
      loc=i;
      break;
    }
  }
if(loc>=0)

  printf("\n%dis found in position%d",item,loc+1);
  else
  printf("\nItem does not exits");
  getch();
   }
```

Output:

```
How many elements:
5
Enter element of the array:
2 5 8 1 3
Enter the element to be searched:
8
Search is Successful
Position of the item searched , 3.


How many elements:
7
Enter element of the array:
2 5 8 1 3 12 45
Enter the element to be searched:
4
Search is Unsuccessful
```

EXPERIMENT No.1(d) (Binary search)

Aim:- Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).

Theory:

Procedure Bsearch (*K*, *N*):

The above procedure searches the desired data item having key '*K*' from the ordered set of data item. The set consists of '*N*' data items having '*N*' distinct keys such that,

$k_1 < k_2 < k_3 < .... < k_N$. This procedure searches for a given argument *K*,

Step 1 Initialization.
Set l ← 1, u ← N.
Step 2 Middle key, loop.
while (u > = l)
{
Set m=1.
if (K = k_m) then
{
Message : "successful search".
display (K).
}
else if (K > k_m)
Set l ←m + 1.
else
Set u ← m – 1.
}
End of loop.
Step 3 Return at the point of call.
Return.

Source code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a[100],i,loc,mid,beg,end,n,flag=0,item;
 clrscr();
 printf("How many elements");
 scanf("%d",&n);
```

```
 printf("Enter the element of the array\n");
 for(i=0;i<=n-1;i++)
{
 scanf("%d",&a[i]);
}
 printf("Enter the element to be searching\n");
 scanf("%d",&item);
 loc=0;
 beg=0;
 end=n-1;
 while((beg<=end)&&(item!=a[mid]))
 {
    mid=((beg+end)/2);
      if(item==a[mid])
            {
              printf("search is successfull\n");
              loc=mid;
              printf("position of the item%d\n",loc+1);
              flag=flag+1;
            }
     if(item<a[mid])
          end=mid-1;
 else
          beg=mid+1;
}
if(flag==0)
{
printf("search is not successfull\n");
}
getch();
}
```

Output:

How many elements:
5
Enter element of the array:
2 5 8 13 25
Enter the element to be searched:
8
Search is Successful
Position of the item searched , 3.
How many elements:
7
Enter element of the array:

1 2 3 4 5 6 7
Enter the element to be searched:
8
Search is Unsuccessful

EXPERIMENT No.2

Aim:- Write a program to implement stack and perform push, pop operation.

Theory:

Procedure Create (S) :

The function creates the stack 'S'. The SIZE variable denotes the maximum limit of an array it is assumed that its size is such that it can accomodate *n* number of elements. The variable 'top' holds the topmost index of array. Initially top stores the value –1, which shows stack is empty.

Step 1 Initialize variable top with value as – 1.
Set top ← –1.
Step 2 Return at the point of call.
Return.

Function IsEmpty (S) :

This Function checks the empty condition in a stack S. The Function returns true if it is empty otherwise false.

```
Step 1    Checking, Is Empty ;
          if (top = –1) then
             return true
             else
          return false
```

The above function can also be written by using ternary operator as follows:

```
Boolean IsEmpty (stack * S)
{
    return ( (S → top = = –1 ) ? TRUE: FALSE) ) ;
}
```

Function IsFull (S) :

The above Function checks whether there exists a stack overflow or not. It returns true if stack overflow occurs otherwise it returns false.

```
Step 1  Checking Is Full?
          if (top  >=  SIZE –1 ) then
          return true
          else
          return false.
```

The above function can also be written by using ternary operator as follows:

```
Boolean IsFull (stack * S)
  {
     return ( ( S → top > = SIZE —1 ) ? TRUE : FALSE ;
  }
```

Procedure push (S, n) :

The above procedure inserts an element stored in variable 'n' to the top of the stack 'S'. Variable 'top' holds the index of the topmost element. Stack overflow condition can be checked by making a call to function IsFull.

```
Step 1  Is overflow?
            if (! IsFull (S) then              R Call to IsFull
            Set top ← top +1
            Set S [top] ← n.
            else
            message : "STACK OVERFLOW"
Step 2   Return of the point of call .
```

The above function can also be written in the following way:

```
void push (stack *S, int n )
{
    if (! IsFull (S))
    {
        S → item [++ S → top] = n ;
    }
    else
    printf (''\n STACK OVERFLOW'' ) ;
}
```

Function Pop (S) :

This Procedure deletes an element from the stack 'S' variable 'top' holds the topmost elements index. Stack underflow (empty) condition can be checked by making a call to Function IsEmpty.

```
Step 1      1s Empty?
            if (! 1sEmpty (S)) then
            Set temp ← S [top]
            Set top  ← top –1
            else
            message : 'STACK UNDERFLOW'
Step 2      Return at the point of call.
            return(temp)
```

Source code:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 100

typedef struct s_tag
{
                int top;
                int item[SIZE];
}stack;

/********** Function Declaration begins **********/
void create(stack *);
void display(stack *);
void push(stack *, int);
void pop(stack *, int);
/********** Function Declaration ends **********/

void main()
{
        int data,ch;
        stack S;
        clrscr();
        create(&S);
        printf("\n\t\t Program shows working of stack : ");
do
{
        printf("\n\n\t\t Menu");
        printf("\n\t\t 1: Push");
        printf("\n\t\t 2: Pop ");
        printf("\n\t\t 3: Exit ");
        printf("\n\t\t Enter choice :");
        scanf("%d",&ch);
        switch(ch)
            {
                case 1:
                                if (S.top >= SIZE)
                                {
                                        printf("\n Stack is full\n");
                                        continue;
                                }
                                else
                                {

                                 printf("\n Enter number to be pushed in the stack: ");
```

```
                                scanf("%d",&data);
                                push(&S,data);
                                S.top—;
                                printf("\n Elements in a stack are :");
                                display(&S);
                                S.top++;
                                continue;
                               }
                case 2:
                                        pop(&S,data);
                                        if (S.top<=0)
                                        {
                                                printf("\n stack is empty\n");
                                                continue;
                                        }
                                        else
                                        {
                                                S.top—;
                                                printf("\n Elements in a stack are : ");
                                                display(&S);
                                                S.top++;
                                                continue;
                                        }
                case 3: printf("\n finish"); return;
            }
}while(ch!=3);
getch();
}

/********** Creating an empty stack **********/
/********** Function Definition begins **********/
void create(stack *S)
{
        S->top=0;
}
/********** Function Definition ends **********/

/********** Pushing an element in stack **********/
/********** Function Definition begins **********/
void push(stack *S, int data)
{

                if (S->top >= SIZE)
                {
                        printf("Stack is full\n");
                }
```

```
                else
                {
                        S->item[S->top] = data;
                        S->top = S->top +1;
                }

}
/********** Function Definition ends **********/

/********** Popping an element from stack **********/
/********** Function Definition begins **********/
void pop(stack *S, int data)
{
                if (S->top <=0)
                {
                        printf("\n Stack is empty\n");
                }
                else
                {
                        S->top = S->top -1;
                        data = S->item[S->top];
                        printf("\n element %d popped\n",data);
                }
}
/********** Function Definition ends **********/

/********** Displaying elements of stack **********/
/********** Function Definition begins **********/
void display(stack *S)
{
        int x;

        for(x=S->top;x>=0;—x)
        {
                printf("%d\t",S->item[x]);
        }
        printf("\n\n");
}
/********** Function Definition ends **********/
```

Output:

```
Program shows working of stack:
Menu
   1: Push
   2: Pop
   3: Exit
  Enter choice: 1
Enter number to be pushed in the stack: 11
Elements in a stack are: 11
          Menu
  1: Push
  2: Pop
  3: Exit
  Enter choice: 1

Enter number to be pushed in the stack: 22
Elements in a stack are: 22    11
          Menu
  1: Push
  2: Pop
  3: Exit
  Enter choice: 1

Enter number to be pushed in the stack: 33
Elements in a stack are: 33    22      11
          Menu
  1: Push
  2: Pop
  3: Exit
  Enter choice: 1

Enter number to be pushed in the stack: 44
Elements in a stack are: 44    33      22      11
          Menu
          1: Push
          2: Pop
          3: Exit
 Enter choice: 1

Enter number to be pushed in the stack: 55
Elements in a stack are: 55    44      33      22      11
          Menu
  1: Push
  2: Pop
  3: Exit
```

```
   Enter choice: 2
Element 55 popped
Elements in a stack are: 44    33      22      11
        Menu
        1: Push
        2: Pop
        3: Exit
        Enter choice: 2
Element 44 popped
Elements in a stack are: 33      22      11
        Menu
        1: Push
        2: Pop
        3: Exit
        Enter choice: 2
Element 33 popped
Elements in a stack are: 22      11
        Menu
        1: Push
        2: Pop
        3: Exit
        Enter choice: 2
Element 22 popped
Elements in a stack are: 22      11
        Menu
        1: Push
        2: Pop
        3: Exit
        Enter choice: 2
Element 11 popped
Stack is empty
        Menu
        1: Push
        2: Pop
        3: Exit
        Enter choice: 3
```

EXPERIMENT No.3

Aim:- Write a program to convert infix expression into postfix expression using stack.

Theory:

Suppose Q is an arithmetic exp written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "(" onto stack and add ")" to the end of Q.
2. Scan Q from left to right and repeat steps 3 to 6 for each element of Q until the stack is empty.
3. If an operand is encountered add it to p.
4. If a left parenthesis is encountered push it onto stack.
5. If an operator is encountered then:
    a. Repeatedly pop from stack.
    b. add operator to stack.
6. If right parenthesis is encountered then:
    a. Repeatedly pop from stack.
    b. Remove the left parenthesis.
7. Exit.

Source code:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
#include<conio.h>
#define SIZE 20

typedef struct stack_t
{
        int top;
        char item[SIZE];
} stack;

/********** Function Declaration begins **********/
void create (stack *S);
void push(stack *, char ch[]);
void pop(stack *);
void infix_to_postfix();
/********** Function Declaration ends **********/
int m,l;
```

```
char A[40],c;

void main()
{
        clrscr();
        printf("\n\ t  Program to covert infix expression into postfix expression; ");
        printf("\n\t Enter your expression & to quit enter fullstop(.) :");
        while((c=getc(stdin))!='\n')
        {
                A[m]=c;
                m++;
        }
        l=m;
        infix_to_postfix ();
        getch();
}

/********** Creating an empty stack **********/
/********** Function Definition begins **********/
void create(stack *S)
{

        S->top = 0 ;
}
/********** Function Definition ends **********/
/********** Pushing an element in stack **********/
/********** Function Definition begins **********/
void push(stack *S, char A[])
{
                if(S->top >= SIZE)
                {
                        printf("\nStack is full");
                }
                else
                {

                        S->item[S->top] = A[m];
                   S->top = S->top+1;
                }
}
/********** Function Definition ends **********/
/********** Popping an element from stack **********/
/********** Function Definition begins **********/
void pop(stack *S)
{
```

```
                if (S->top < 0)
                {
                        printf("\n Stack is empty");
                }
                else
                {
                   if(S->top >=0)
                        {
                                S->top = S->top-1;
                                if(S->item[S->top]!='(')
                                        printf("%c",S->item[S->top]);
                        }
                }

}
/********** Function Definition ends **********/
/********** Infix to Postfix conversion **********/
/********** Function Definition begins **********/
void infix_to_postfix()
{
        stack S;
        create(&S);
        m=0;
        while(m<l)
        {
                switch(A[m])
                {
                case '+' :
                case '-' :
                        while(S.item[S.top-1] =='-' || S.item[S.top-1] =='+' ||S.item[S.top-
1] =='*' ||S.item[S.top-1] =='/' ||S.item[S.top-1] =='^' && S.item[S.top-1] !='(')
                        pop(&S);
                        push(&S,A);
                        ++m;
                        break;
                case '/' :
                case '*' :
                        while(S.item[S.top-1] =='*' ||S.item[S.top-1] =='/' ||S.item[S.top-
1] =='^' && S.item[S.top-1] !='(')
                        pop(&S);
                        push(&S,A);
                        ++m;
                        break;
                case '^' :
                        push(&S,A);
                        ++m;
```

```
                break;
        case '(' :
                push(&S,A);
                ++m;
                break;
        case ')' :
                while(S.item[S.top-1]!='(')
                        pop(&S);
                        pop(&S);
                        ++m;
                        break;
        case '.' :
                while (S.top >= 0)
                        pop(&S);
                exit(0);
        default : if(isalpha(A[m]))
                          {
                                printf("%c",A[m]);
                                ++m;
                                break;
                          }
                        else
                        {
                                printf("\n some error");
                        exit(0);
                        }
        }
    }
}
/********** Function Definition ends **********/
```

Output:

```
Program to covert infix expression into postfix expression;
Enter your expression & to quit enter fullstop(.) :A+B/C-D.
    ABC/+D-
```

EXPERIMENT No.4

Aim: - Write a program to perform following operations in linear queue-addition, deletion, and traversing.

Theory:

Procedure createQ(Q):

The above procedure creates an empty queue. Variable front and rear set to value – 1.

Step 1 [setting values to –1]
Set Q (front) ← –1.
Set Q (rear) ← – 1.
Step2 [return at the point of call]
Return.

Procedure Enqueue (Q , item):

This procedure inserts an element 'item' at the rear-end of the queue, '*Q*' only when it is not full. Variable 'rear' points to the element recently inserted. Queue overflow condition can be checked by making a call to Function 'IsFull'.

Step 1 [checking overflow condition]
call to IsFull.
if (IsFull (Q)) then
message: 'Queue overflow'
return .
else goto step 2
Step 2 [setting rear, insert item value]
Set Q (rear) ← Q(rear) + 1.
Set Q (item [a (rear))] ← item.
Step 3 [setting front value]
if (Q (front) = – 1) then
Set Q (front) → 0.
Return.

Function Dequeue (Q):

The above Function deletes an element from the queue '*Q*'. Queue empty condition is checked by making a call to 'IsEmpty'.

Step 1 [Is Empty, call to IsEmpty]

if (IsEmpty (Q)) then
message ‘Queue Empty’
return .
else goto step 2.

Step 2 [Deletion of an element]
Set temp ← Q (item [Q (front)]).

Step 3 [setting front and rear, if queue is empty]
if Q (front) = Q(rear)) then
Set Q (front) ← – 1.
Set Q (rear) ← – 1.
else
Set Q(front) → Q(front) + 1.

Step 4 [return value at the time of call]
return (temp).

Source code:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 20

typedef struct q_tag
{
        int front,rear;
        int item[SIZE];
}queue;

/********** Function Declaration begins **********/
void create(queue *);
void display(queue *);
void enqueue(queue *, int);
int dequeue(queue *, int);
/********** Function Declaration ends **********/

void main()
{
    int data,ch;
    queue Q;
    clrscr();
    create(&Q);
    printf("\n\t\t Program shows working of queue using array");
do
{
    printf("\n\t\t Menu");
    printf("\n\t\t 1: enqueue");
```

```
        printf("\n\t\t 2: dequeue ");
        printf("\n\t\t 3: exit. ");
        printf("\n\t\t Enter choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                                if (Q.rear >= SIZE)
                                {
                                        printf("\n Queue is full");
                                        continue;
                                }
                                else
                                {

                                    printf("\n Enter number to be added in a queue  ");
                                    scanf("%d",&data);
                                    enqueue(&Q,data);
                                    printf("\n Elements in a queue are: ");
                                   display(&Q);
                                   continue;
                                }
                case 2:
                                        dequeue(&Q,data);
                                        if (Q.front==0)
                                        {
                                                continue;
                                        }
                                        else
                                        {
                                                printf("\n Elements in a queue are : ");
                                                display(&Q);
                                                continue;
                                        }
                case 3: printf("\n finish"); return;
        }
}while(ch!=3);
getch();
}

/********** Creating an empty queue **********/
/********** Function Definition begins **********/
void create(queue *Q)
{
        Q->front=0;
        Q->rear =0;
```

```
}
/********** Function Definition ends **********/

/********** Inserting an element in queue **********/
/********** Function Definition begins **********/
void enqueue(queue *Q, int data)
{
                if (Q->rear >= SIZE)
                {
                        printf("\n Queue is full");
                }
                if (Q->front == 0)
                {
                        Q->front = 1;
                        Q->rear = 1;
                }
                else
                {
                        Q->rear = Q->rear +1;
                }
                Q->item[Q->rear] = data;

}
/********** Function Definition ends **********/

/********** Deleting an element from queue **********/
/********** Function Definition begins **********/
int dequeue(queue *Q, int data)
{
                if (Q->front == 0)
                {
                        printf("\n Underflow.");
                        return(0);
                }
                else
                {
                        data = Q->item[Q->front];
                        printf(" \n Element %d is deleted",data);
                }
                if (Q->front==Q->rear)
                {
                        Q->front =0;
                        Q->rear = 0;
                        printf("\n Empty Queue");
                }
                else
```

```
                {
                        Q->front = Q->front +1;
                }
                return data;
}
/********** Function Definition ends **********/

/********** Displaying elements of queue **********/
/********** Function Definition begins **********/
void display(queue *Q)
{
        int x;
        for(x=Q->front;x<=Q->rear;x++)
        {
                printf("%d\t",Q->item[x]);
        }
        printf("\n\n");
}
/********** Function Definition ends **********/
```

Output:

```
        Program shows working of queue using array
        Menu
        1: enqueue
        2: dequeue
        3: exit.
        Enter choice : 1
Enter number to be added in a queue 11
Elements in a queue are :11
        Menu
        1: enqueue
        2: dequeue
        3: exit.
        Enter choice :1
Enter number to be added in a queue 22
Elements in a queue are: 11     22
        Menu
        1: enqueue
        2: dequeue
```

```
        3: exit.
        Enter choice :1
Enter number to be added in a queue 33
Elements in a queue are: 11     22      33
        Menu
        1: enqueue
        2: dequeue
        3: exit.
        Enter choice :1
Enter number to be added in a queue 44
Elements in a queue are:11     22      33      44
        Menu
        1: enqueue
        2: dequeue
        3: exit.
        Enter choice :1
Enter number to be added in a queue 55
Elements in a queue are:11     22      33      44      55
        Menu
        1: enqueue
        2: dequeue
        3: exit.
        Enter choice : 2
Element 11 is deleted
Elements in a queue are : 22    33      44      55
        Menu
        1: enqueue
        2: dequeue
        3: exit.
        Enter choice : 2
Element 22 is deleted
Elements in a queue are : 33    44      55
        Menu
        1: enqueue
        2: dequeue
        3: exit.
        Enter choice : 2
Element 33 is deleted
Elements in a queue are :  44    55
        Menu
        1: enqueue
        2: dequeue
        3: exit.
        Enter choice : 2
Element 44 is deleted
Elements in a queue are  :55
```

Menu
1: enqueue
2: dequeue
3: exit.
Enter choice : 2
Element 55 is deleted
Empty Queue
Menu
1: enqueue
2: dequeue
3: exit.
Enter choice : 3

EXPERIMENT No.5

Aim:- Write a program to perform following operations operation in circular queue- addition, deletion, and traversing.

Theory:

Procedure EnCqueue (Q, data):

This procedure inserts value data in circular queue.

```
Step 1    [If Empty]
          if (CQ (front = – 1) then
          {
          Set CQ (front) ← 0.
          Set CQ (rear) ← 0 .
          }
          elseif (CQ (rear) = (SIZE – 1) then
          Set CQ (rear) = 0.
          else
          Set CQ(rear) ← CQ(rear) + 1.
Step 2    [Inserts value at rear end]
          Set CQ (item [CQ (rear)]) ← data.
Step 3    return at the point of call
          Return.
```

Function DeCqueue (CQ):

This Function deletes an element from circular queue.

```
Step 1    [copying front index value to temporary variable]
          Set data ← CQ (item [Q(front)]
Step 2    [setting values.]
          if (CQ (front) = CQ(rear))
              {
                 Set CQ (front) = – 1.
                 Set CQ(rear) = – 1.
              }
          elseif (CQ(front) = SIZE – 1)
               Set CQ (front) = 0.
          else
               Set CQ(front ) ← CQ (front) + 1.
Step 3    [return value at the  time of call.]
             Return (data).
```

Procedure Qcreate (Q):

The above Procedure creates Q by setting pointer variables 'front' and 'rear' value to NULL.

Step 1 [setting value to NULL]
Set Q(front) ← NULL.
Set Q(rear) ← NULL.
Step 2 [return at the point of call]
Return.

Source code:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 20

typedef struct circularq_t
{
        int front,rear;
        int item[SIZE];
}circularQ;
/********** Function Declaration begins **********/
void create(circularQ *);
void display(circularQ *);
void encqueue(circularQ *, int);
void decqueue(circularQ *, int);
/********** Function Declaration ends **********/

void main()
{
    int data,ch;
    circularQ CQ;
    clrscr();
    create(&CQ);
    printf("\n\t\t Program shows working of circular queue");
do
{
    printf("\n\t\t Menu");
    printf("\n\t\t 1: encequeue");
    printf("\n\t\t 2: decqueue ");
    printf("\n\t\t 3: exit. ");
    printf("\n\t\t Enter choice : ");
    scanf("%d",&ch);
```

```
    switch(ch)
    {
    case 1:
            printf("\n Enter data:");
              scanf("%d",&data);
              encqueue(&CQ,data);
            printf("\n Elements in a circular queue are : ");
            display(&CQ);
            continue;
    case 2:
            decqueue(&CQ,data);
            if (CQ.front==0)
                continue;
            else
            {
                printf("\n Elements in a circular queue are : ");
          display(&CQ);
          continue;
             }
          case 3:  printf("\n finish"); return;
    }
}while(ch!=3);
getch();
}

/********** Creating an empty circular queue **********/
/********** Function Definition begins **********/
void create(circularQ *CQ)
{
    CQ->front=0;
    CQ->rear =0;
}
/********** Function Definition ends **********/

/********** Inserting elements in circular queue **********/
/********** Function Definition begins **********/
void encqueue(circularQ *CQ, int data)
{
    if (((CQ->rear == (SIZE-1)) && (CQ->front ==1 )) ||
    (CQ->front == (CQ->rear + 1)))
    {
          printf("\n Circular queue is full");
          return;
    }
    else
```

```
    {
        if (CQ->front == 0)
        {
            CQ->front = 1;
            CQ->rear = 1;
            CQ->item[CQ->rear] = data;
        }
        else
        if(CQ->rear == SIZE-1)
        {
            CQ->rear = 1;
            CQ->item[CQ->rear] = data;
        }
        else
        {
            CQ->rear = CQ->rear +1;
            CQ->item[CQ->rear] = data;
        }
    }
}
/********** Function Definition ends **********/

/********** Deleting element from circular queue **********/
/********** Function Definition begins **********/
void decqueue(circularQ *CQ, int data)
{
    if (CQ->front == 0)
    {
        printf("\n Circular queue underflow");
        return;
    }
    data = CQ->item[CQ->front];
    CQ->item[CQ->front] = 0;
    printf("\n Element %d is deleted :",data);
    if (CQ->front==CQ->rear)
    {
        CQ->front =0;
        CQ->rear = 0;
        printf("\n Circular queue is empty");
    }
    else
    if (CQ->front == (SIZE-1))
        CQ->front =1;
    else
```

```
        CQ->front = CQ->front +1;
}
/********** Function Definition ends **********/

/********** Displaying elements of circular queue **********/
/********** Function Definition begins **********/
void display(circularQ *CQ)
{
    int x;
    if ((CQ->rear > 1)&&(CQ->front == (CQ->rear+1)))
    {
        for(x=1;x<SIZE;x++)
        {
            printf("%d\t",CQ->item[x]);
        }
        printf("\n");
    }
    else
    if(CQ->front == (CQ->rear+1))
    {
        for(x=CQ->rear;x<=SIZE;x++)
        {
            printf("%d\t",CQ->item[x]);
        }
        printf("\n");
    }
    else
    if(CQ->front > (CQ->rear +1))
    {
        for(x=1;x<=CQ->rear;x++)
        {
            printf("%d\t",CQ->item[x]);
        }
        for(x=CQ->front;x<SIZE;x++)
        {
            printf("%d\t",CQ->item[x]);
        }
        printf("\n");
    }
    else
    {
        for(x=CQ->front;x<=CQ->rear;x++)
        {
            printf("%d\t",CQ->item[x]);
```

```
            }
            printf("\n");
        }
    }
    /********** Function Definition ends **********/
```

Output:

```
Program shows working of circular queue
Menu
1: encequeue
2: decqueue
3: exit.
Enter choice : 1

Enter data : 11
Elements in a circular queue are : 11
Menu
1: encequeue
2: decqueue
3: exit.
Enter choice :1

Enter data : 22
Elements in a circular queue are : 11   22
Menu
1: encequeue
2: decqueue
3: exit.
Enter choice :1

Enter data: 33
Elements in a circular queue are : 11   22      33
Menu
1: encequeue
2: decqueue
3: exit.
Enter choice : 2

Element 11 is deleted :
Elements in a circular queue are :22   33
Menu
```

1: encequeue
2: decqueue
3: exit.
Enter choice :2

Element 22 is deleted :
Elements in a circular queue are :33
Menu
1: encequeue
2: decqueue
3: exit.
Enter choice :2

Element 33 is deleted :
Circular queue is empty
Menu
1: encequeue
2: decqueue
3: exit.
Enter choice: 3

EXPERIMENT No.6

Aim: - Write a program to perform following operations in double ended queue addition , deletion , and traversing.

Theory:

Function Dequeue (Q):

The above function deletes node from the front of the list. A call to FreeNode Function is made in order to return the memory to the available list.

```
Step 1      Initialization.
            Set temp ← Q(front (item)).
            Set p ← Q(front).
Step 2  Checking for single element.
            if (Q (front) = Q (rear) then
                {
                    Set Q (front) ← NULL.
                    Set Q (rear) ← NULL.
                }
            else
                {
            Set Q (front) ←Q(front (link)).
Step 3  Calling FreeNode, and returning value of temp.
            call to FreeNode ( )
            return.
```

Source code:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 20

typedef struct dq_t
{
        int front,rear;
        int item[SIZE];
}deque;

/********** Function Declaration begins **********/
void create(deque *);
```

```
void display(deque *);
void insert_rear(deque *, int);
void insert_front(deque *, int);
int  delete_front(deque *, int);
int  delete_rear(deque *, int);
/********** Function Declaration ends **********/

void main()
{
    int x,data,ch;
    deque DQ;
    clrscr();
    create(&DQ);
    printf("\n\t\t Program shows working of double ended queue");
do
{
    printf("\n\t\t Menu");
    printf("\n\t\t 1: insert at rear end");
    printf("\n\t\t 2: insert at front end");
    printf("\n\t\t 3: delete from front end");
    printf("\n\t\t 4: delete from rear end");
    printf("\n\t\t 5: exit. ");
    printf("\n\t\t Enter choice : ");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
                if (DQ.rear >= SIZE)
                {
                    printf("\n Deque is full at rear end");
                    continue;
                }
                else
                {
                    printf("\n Enter element to be added at rear end : ");
                    scanf("%d",&data);
                    insert_rear(&DQ,data);
                    printf("\n Elements in a deque are : ");
                    display(&DQ);
                    continue;
                }
        case 2:
                if (DQ.front <=0)
                {
```

```
                        printf("\n Deque is full at front end");
                        continue;
                }
                else
                {
                        printf("\n Enter element to be added at front end : ");
                        scanf("%d",&data);
                        insert_front(&DQ,data);
                        printf("\n Elements in a deque are : ");
                        display(&DQ);
                        continue;
                }
        case 3:
                        x = delete_front(&DQ,data);
                        if (DQ.front==0)
                        {
                                continue;
                        }
                        else
                        {
                                printf("\n Elements in a deque are : ");
                                display(&DQ);
                                continue;
                        }
        case 4:
                       x = delete_rear(&DQ,data);
                        if (DQ.rear==0)
                        {
                                continue;
                        }
                        else
                        {
                                printf("\n Elements in a deque are : ");
                                display(&DQ);
                                continue;
                        }
        case 5: printf("\n finish"); return;
    }
}while(ch!=5);
getch();
}

/********** Creating an empty double ended queue **********/
/********** Function Definition begins **********/
```

```
void create(deque *DQ)
{
    DQ->front=0;
    DQ->rear =0;
}
/********** Function Definition ends **********/

/********** Inserting element at rear end **********/
/********** Function Definition begins **********/
void insert_rear(deque *DQ, int data)
{
        if ((DQ->front == 0) &&(DQ->rear == 0))
        {
            DQ->item[DQ->rear] = data;
            DQ->rear = DQ->rear +1;
        }
        else
        {
            DQ->item[DQ->rear] = data;
            DQ->rear = DQ->rear +1;
        }
}
/********** Function Definition ends **********/

/********** Deleting element from front end **********/
/********** Function Definition begins **********/
int delete_front(deque *DQ, int data)
{
        if ((DQ->front == 0) && (DQ->rear == 0))
        {
            printf("\n Underflow");
            return(0);
        }
        else
        {
            data = DQ->item[DQ->front];
            printf("\n Element %d is deleted from front :",data);
            DQ->front = DQ->front +1;
        }
        if (DQ->front==DQ->rear)
        {
            DQ->front =0;
            DQ->rear = 0;
            printf("\n Deque is empty (front end)");
```

```
        }
        return data;
}
/********** Function Definition ends **********/

/********** Inserting element at front end **********/
/********** Function Definition begins **********/
void insert_front(deque *DQ, int data)
{
        if(DQ->front > 0)
        {
                DQ->front = DQ->front-1;
                DQ->item[DQ->front] = data;
        }
}
/********** Function Definition ends **********/

/********** Deleting element from rear end **********/
/********** Function Definition begins **********/
int delete_rear(deque *DQ, int data)
{
        if (DQ->rear == 0)
        {
                printf("\n Underflow");
                return(0);
        }
        else
        {
                DQ->rear = DQ->rear -1;
                data = DQ->item[DQ->rear];
                printf("\n Element %d is deleted from rear: ",data);
        }
        if (DQ->front==DQ->rear)
        {
                DQ->front =0;
                DQ->rear = 0;
                printf("\n Deque is empty(rear end)");

        }
        return data;
}
/********** Function Definition ends **********/

/********** Displaying elements of DEQUE **********/
```

```
/********** Function Definition begins **********/
void display(deque *DQ)
{
    int x;
        for(x=DQ->front;x<DQ->rear;x++)
    {
        printf("%d\t",DQ->item[x]);
    }
    printf("\n\n");
}
/********** Function Definition ends **********/
```

Output:

```
    Program shows working of double ended queue
    Menu
    1: insert at rear end
    2: insert at front end
    3: delete from front end
    4: delete from rear end
    5: exit.
    Enter choice : 1

Enter element to be added at rear end :  11
Elements in a deque are : 11
    Menu
    1: insert at rear end
    2: insert at front end
    3: delete from front end
    4: delete from rear end
    5: exit.
    Enter choice :1

Enter element to be added at rear end : 22
Elements in a deque are : 11    22
    Menu
    1: insert at rear end
    2: insert at front end
    3: delete from front end
    4: delete from rear end
    5: exit.
    Enter choice : 1
```

```
Enter element to be added at rear end : 33
Elements in a deque are : 11    22      33
    Menu
    1: insert at rear end
    2: insert at front end
    3: delete from front end
    4: delete from rear end
    5: exit.
    Enter choice : 2

Deque is full at front end
    Menu
    1: insert at rear end
    2: insert at front end
    3: delete from front end
    4: delete from rear end
    5: exit.
    Enter choice :3

Element 11 is deleted from front :
Elements in a deque are : 22    33
    Menu
    1: insert at rear end
    2: insert at front end
    3: delete from front end
    4: delete from rear end
    5: exit.
    Enter choice : 2

Element 11 is deleted from front :
Elements in a deque are : 22    33
    Menu
    1: insert at rear end
    2: insert at front end
    3: delete from front end
    4: delete from rear end
    5: exit.
      Enter choice : 5
```

EXPERIMENT No.7

Aim: - Write a program to perform following in singly linked list- creation, insertion, and deletion .

Theory:

Function GetNode ( ):

This procedure provides 'new', a pointer to a free node from the available list. If no node is available, then it displays an error message and return NULL

Step 1 [checking NULL].

if (avail = NULL) then

message : "overflow".

return (NULL).

Step 2 [Adjusting pointers]

Set new ← avail.

Set new ← Next (avail).

Step 3 [return at the point of call]

return (new).

Procedure FreeNode (F):

This procedure returns node pointed by 'F' to the available list.

Step 1 [Setting the pointers]

Set Next (F) ← avail.

Set avail ← Next.

Step 2 [return at the point of call]

Return.

Procedure SLCreation (START):

The above Procedure creates an empty linked list pointer variable START is set to NULL.

Step 1 [Initialization]

Set start = NULL.

Step 2 [return at the point of coll]

Return.

Function SLEmpty (START):

The Function checks the empty condition for the linked list. If the list is empty then it returns 'true' otherwise 'false'.

Step 1 [Checking empty]

if (START = NULL) then

return true.

else

return false.

Procedure SLTraverssing (START):

The above Procedure traverses the whole linked list. Pointer variable 'START' stores the address of the first node and 'keep' it to traverse the list.

Step 1 [Is Empty ?]

If (SL Empty (START)) then

message : ''Empty list''.

Return.

else go to step 2

Step 2 [Traversing the list]

Set keep ← START.

loop

while (Next (keep) ←NULL)

display : data (keep).

Set keep ← Next (keep).

End loop.

display : data (keep).

Function SLInsertionpos (START, pos):

The above Function inserts a node at a given position in a single linked list. Pointer variable 'START' stores the address of the first node, and 'keep' tracks the NULL. Pointer variable 'new' points to the new node obtained after calling the GetNode function.

Step 1 [Call to GetNode]

Set new ←Call to GetNode( ).

Step 2 [Checking empty ?]

if (SLEmpty (START)) then

message : "Empty list".

first node.

Set START ← new.

return (START).
else goto step 3

Step 3 [Insertion at desired position]
if (pos = 1) then
goto step 4.
else
goto step 5.

Step 4 [Insertion at the beginning.]
Set Next (new) ← START.
Set START ← new.
return (START).

Step 5 [Insertion at desired position other than beginning.]
Set count ← 1.
Set keep ← START.
loop
while (count ← pos –1)
Set keep□ ← Next (keep).
Set count ← count + 1.
End of loop.

Step 6 [Setting of pointers]
Set Next (new) ← Next (keep).
Set Next (keep) ← new.

Step 7 [Return at the point of call.]
Return(start).

Function SLDeletionpos (START, pos):

The Function deletes node at a given position in a single linked list. Pointer variable 'START' stores the address of the first node and 'keep' tracks the NULL address. The 'FreeNode' operation frees the memory back to the available list. The Function returns the current 'START' after deletion.

Step 1 [Is Empty ?]
if (SLEmpty (START)) then
message : "Empty list".
return (START).
else go to step 2.

Step 2 [Deletion at the beginning]
if (pos = 1) then
goto step 3.

else
go to step 4.

Step 3 [Deletion at the beginning]
Set keep ← START.
Set START ← Next (START)
FreeNode (keep).
return (START).

Step 4 [Deletion at desired position.]
Set count ← 1.
Set keep ← start.
loop
while (count ← pos –1)
Set prev ←keep.
Set keep ← Next (keep).
Set count ← count + 1.
End loop.

Step 5 [Setting of pointers]
Set Next (prev) ← Next (keep).

Step 6 [Returning back the memory]

Source code:

```
#include <stdio.h>
#include <malloc.h>
#include<process.h>
typedef struct list_tag
{
    int data;
    struct list_tag *link;
}node;

/**************Function Declaration Begin**********/
node *SLcreation(node *);
node *SLinsertion(node *);
node *SLdeletion(node *);
void  SLdisplay(node *);
```

```
/**************Function Declaration End**********/
void main()
{
    node *START=NULL;
    int ch;
    do
    {
        printf("\n\t\t Program for singly linked list\n");
        printf("\n\t\t\t Menu:\n");
        printf("\n\t\t1.Create");
        printf("\n\t\t2.Insert");
        printf("\n\t\t3.Delete");
        printf("\n\t\t4.Display");
        printf("\n\t\t5.Exit");
        printf("\n\t\tEnter choice : ");
        scanf("%d",&ch);
        switch(ch)
            {
                case 1 :
                    START = SLcreation(START);
                    break;
                case 2:
                    START = SLinsertion(START);
                    break;
                case 3:
                    START = SLdeletion(START);
                    break;
                case 4:
                    printf("\n***** Linked list *****\n");
                    SLdisplay(START);
                    break;
                case 5:
                    exit(0);
                default :
                    printf("\nWrong choice:");
            }
        }
```

```
        while (ch!=5);
        printf("\n");
}

/********** Creating of linked list MENU **********/
/********** Function Definition begins **********/
node *SLcreation(node *START)
{
    node *temp,*prev;
    int item;
    char ch;
    prev = START = NULL;
    do
    {
        printf("\n\t\t Menu:");
        printf("\n\t\t1.Add node");
        printf("\n\t\t2. Display:");
        printf("\n\t\t3. Quit:");
        printf("\n\t\tEnter choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                printf("\nEnter data:");
                scanf("%d",&item);
                temp = (node*)malloc(sizeof(node));
                temp->data = item;
                temp->link = NULL;
                if (START == NULL)
                    START = temp;
                else
                    prev->link = temp;
                prev = temp;
                break;
            case 2:
                printf("\n***** Linked list *****\n");
                SLdisplay(START);
```

```
                case 3:
                        break;
                default:
                                printf("\nWrong choice:");
                }
        }while (ch != 3);
        return START;
}
/********** Function Definition ends **********/

/********** Insertion of node in linked list **********/
/********** Function Definition begins **********/
node* SLinsertion(node *START)
{
        node *new_node, *temp;
        int i,item,pos;

        printf("\nEnter data to be inserted : ");
        scanf("%d",&item);
        do
        {
                printf("\nEnter the position of insertion : ");
                scanf("%d",&pos);
        }
        while (pos < 1);

        new_node = (node*)malloc(sizeof(node));
        new_node->data = item;
        if ((pos == 1) || (START == NULL))
        {
                new_node->link = START;
                START = new_node;
        }
        else
        {
                temp = START;
                i = 2;
```

```
            while ((i <pos) && (temp->link != NULL))
            {
                  temp = temp->link;
                  ++i;
            }
            new_node->link = temp->link;
            temp->link = new_node;
      }
      return START;
}
/********** Function Definition ends **********/

/********** Deletion of node in linked list **********/
/********** Function Definition begins **********/
node *SLdeletion(node *START)
{
      node *temp, *prev;
      int item;

      printf("\nEnter data to be deleted : ");
      scanf("%d",&item);
      if (START == NULL)
            printf("\nCan't delete - list empty\n");
      else
      {
            prev = NULL;
            temp = START;
            while ((temp != NULL) && (temp->data != item))
            {
                  prev = temp;
                  temp = temp->link;
            }
            if (temp == NULL)
                  printf("Element not found\n");
            else
            {
                  if (prev == NULL)
```

```
                    START = START->link;
                else
                    prev->link = temp->link;
            printf("\n***** Linked list *****\n");
            }
    }
    return START;
}
/********** Function Definition ends **********/

/********** Displaying nodes of linked list **********/
/********** Function Definition begins **********/
void SLdisplay(node *START)
{
    printf("\nSTART->");
        while (START != NULL)
        {
            printf("%d->",START->data);
            START = START->link;
        }
        printf("->NULL\n\n");
}
/********** Function Definition ends **********/
```

Output:

```
Program for singly linked list
Menu:
1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice : 1

Menu:
1.Add node
```

2. Display:
3. Quit:
Enter choice:1

Enter data:11
Menu:
1.Add node
2. Display:
3. Quit:
Enter choice:1

Enter data:22
Menu:
1.Add node
2. Display:
3. Quit:
Enter choice:1

Enter data:33
Menu:
1.Add node
2. Display:
3. Quit:
Enter choice:2

***** Linked list *****

START->11->22->33->->NULL
Menu:
1.Add node
2. Display:
3. Quit:
Enter choice:3

Program for singly linked list
Menu:
1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice : 3

Enter data to be deleted : 22

***** Linked list *****

Program for singly linked list
Menu:

1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice :4

***** Linked list *****

START->11->33->->NULL

Program for singly linked list
Menu:
1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice :5

EXPERIMENT No.8

Aim:- Write a program to perform creation , insertion , deletion of doubly linked list.

Theory:

Procedure Dcreate (START, END)

This procedure creates an empty list. The pointer variable START and END are assigned a sentinel value to indicate the list is empty in the beginning.

Step1 Initialization.
Set START ← NULL
Set END ← NULL
Step 2 R return at the point of call.
return

Source code:

```
#include <stdio.h>
#include <malloc.h>
#include<process.h>

typedef struct DList_tag
{
    int data;
    struct DList_tag *rlink, *llink;
}node;

/**************Function Declaration Begin**********/
node *DLcreation(node **);
void DLinsertion(node **, node **, int, int);
void DLdeletion(node **, node**);
void DLdisplay(node *, node *);
/**************Function Declaration End**********/

void main()
{
    node *left=NULL,*right;
    int item,pos,ch;
    printf("\n\t\tProgram for doubly linked list\n");
```

```
    do
    {
        printf("\n\t\tMenu");
        printf("\n\t\t1.Create");
        printf("\n\t\t2.Insert");
        printf("\n\t\t3.Delete");
        printf("\n\t\t4.Display");
        printf("\n\t\t5.Exit");
        printf("\n\t\tEnter choice : ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:
                left = DLcreation(&right);
                break;
            case 2:
                printf("\nEnter data :");
                scanf("%d",&item);
                do
                {
                    printf("\nEnter position of insertion :");
                    scanf("%d",&pos);
                }while(pos < 1);
                DLinsertion(&left,&right,item,pos);
                break;
            case 3:
                DLdeletion(&left,&right);
                break;
            case 4:
                printf("\n\t**** Doubly linked list *****\n");
                DLdisplay(left,right);
                break;
            case 5:
                exit(0);
            default:
                printf("\n Wrong Choice");
        }
    }while(ch!=5);
    printf("\n");
}

/********** Creating of double linked list MENU **********/
```

```
/********** Function Definition begins **********/
node *DLcreation( node **right )
{
    node *left, *new_node;
    int item,ch;
    *right = left = NULL;
    do
    {
        printf("\n\t\tMenu");
        printf("\n\t\t1.Add node");
        printf("\n\t\t2.Quit");
        printf("\n\t\tEnter choice : ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:
                printf("\n Enter data:");
                scanf("%d",&item);
                new_node = (node *)malloc(sizeof(node));
                new_node->data = item;
                new_node->rlink = NULL;
                if(left == NULL)
                    {
                        new_node->llink = NULL;
                        left = new_node;
                    }
                else
                    {
                        new_node->llink = (*right);
                        (*right)->rlink = new_node;
                    }
                (*right) = new_node;
                if(left != NULL)
                    (*right) = new_node;
                break;
            case 2:
                break;
            default:
                printf("\n Wrong Choice");
        }
    }while(ch!=2);
    return left;
}
```

```
/********** Function Definition ends **********/

/********** Insertion of node in double linked list **********/
/********** Function Definition begins **********/
void DLinsertion(node **start, node **right,int item, int pos)
{
    node *new_node, *temp;
    int i;
    if((pos == 1) ||((*start) == NULL))
    {
        new_node = (node *)malloc(sizeof(node));
        new_node->data = item;
        new_node->rlink = *start;
        new_node->llink = NULL;
        if((*start) != NULL)
            (*start)->llink = new_node;
        else
            (*right) = new_node;
        *start = new_node;
    }
    else
    {
        temp = *start;
        i = 2;
        while((i < pos) && (temp->rlink != NULL))
        {
            temp = temp->rlink;
            ++i;
        }
        new_node = (node *)malloc(sizeof( node));
        new_node->data = item;
        new_node->rlink = temp->rlink;
        if(temp->rlink != NULL)
            temp->rlink->llink = new_node;
            new_node->llink = temp;
            temp->rlink = new_node;
    }
    if(new_node->rlink == NULL)
        *right = new_node;
}
/********** Function Definition ends **********/

/********** Deletion of node in linked list **********/
/********** Function Definition begins **********/
```

```
void DLdeletion( node **start, node **right)
{
    node *temp, *prec;
    int item;
    printf("\nElement to be deleted :");
    scanf("%d",&item);
    if(*start != NULL)
    {
        if((*start)->data == item)
        {
            if((*start)->rlink == NULL)
                *start = *right = NULL;
            else
            {
                *start = (*start)->rlink;
                (*start)->llink = NULL;
            }
        }
        else
        {
            temp = *start;
            prec = NULL;
            while((temp->rlink != NULL) && (temp->data != item))
            {
                prec = temp;
                temp = temp->rlink;
            }
            if(temp->data != item)
                printf("\n Data in the list not found\n");
            else
            {
                if(temp == *right)
                    *right = prec;
                else
                    temp->rlink->llink = temp->llink;
                prec->rlink = temp->rlink;
            }
        }
    }
    else
        printf("\n!!! Empty list !!!\n");
    return;
}
/********** Function Definition ends **********/
```

```
/********** Displaying nodes of double linked list **********/
/********** Function Definition begins **********/
void DLdisplay(node *start, node *right)
{
    printf("\n***** Traverse in Forward direction *****\n left->");
    while(start != NULL)
    {
        printf("%d-> ",start->data);
        start = start->rlink;
    }
    printf("right");
    printf("\n***** Traverse in Backward direction *****\n right->");
    while(right != NULL)
    {
        printf("%d-> ",right->data);
        right = right->llink;
    }
    printf("left");
}
/********** Function Definition ends **********/
```

Output:

```
Program for doubly linked list
Menu
1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice : 1
Menu
1.Add node
2.Quit
Enter choice : 1

Enter data:11
Menu
1.Add node
2.Quit
 Enter choice : 1
```

```
Enter data:22
    Menu
    1.Add node
    2.Quit
    Enter choice : 1
Enter data:33
    Menu
    1.Add node
    2.Quit
    Enter choice : 1
Enter data:44
    Menu
    1.Add node
    2.Quit
    Enter choice : 1
Enter data:55
    Menu
    1.Add node
    2.Quit
    Enter choice : 2

    Menu
    1.Create
    2.Insert
    3.Delete
    4.Display
    5.Exit
    Enter choice : 2

    Menu
    1.Create
    2.Insert
    3.Delete
    4.Display
    5.Exit
    Enter choice : 2
Enter data :99
Enter position of insertion :3
    Menu
    1.Create
    2.Insert
    3.Delete
    4.Display
```

```
      5.Exit
      Enter choice :4

**** Doubly linked list *****

***** Traverse in Forward direction *****
 left->11-> 22-> 99-> 33-> 44-> 55-> right
***** Traverse in Backward direction *****
 right->55-> 44-> 33-> 99-> 22-> 11-> left

      Menu
      1.Create
      2.Insert
      3.Delete
      4.Display
      5.Exit
      Enter choice : 3

Element to be deleted :33
      Menu
      1.Create
      2.Insert
      3.Delete
      4.Display
      5.Exit
      Enter choice : 4

   **** Doubly linked list *****

***** Traverse in Forward direction *****
 left->11-> 22-> 99-> 44-> 55-> right
***** Traverse in Backward direction *****
 right->55-> 44-> 99-> 22-> 11-> left
      Menu
      1.Create
      2.Insert
      3.Delete
      4.Display
      5.Exit
   Enter choice :5
```

EXPERIMENT No. 9

Aim:-Write a program to implement polynomial in link list and perform

(a) **Arithmetic.**

(b) **Evaluation.**

Theory:-

Linked lists are widely used to represent and manipulate polynomials. Polynomials are the expressions containing number of terms with non zero coefficients and exponents. Consider the following polynomial.

$p(X)=a^n x^e_n+a_{n-1}x^e_{n-1}+\ldots+a_1 x_1^e+a$

where a_i a are nonzero coefficients.

e_i a are exponents such that

In the linked representation of polynomials, each term is considered as a node. And such a node contains three fields.

1.Coefficent field 2.Exponent field 3.Link field.

The coefficient field holds the value of the coefficient of a term and the exponent field contains the exponent value of that term and the exponent field contains the exponent value of that term. And the link field contains the addresses of the next term in the polynomial.

The logical representation of the above node is given below:

```
struct polynode
{
int coeff;
int expo;
struct polynode *ptr;
};
typedef struct polynode PNODE;
```

Two polynomials can be added. And the steps involved in adding two polynomials are given below:

1.Read the number of terms in the first polynomial P.

2.Read the coefficients and exponents of the first polynomial.

3.Read the number of terms in the second polynomial Q.

4.Read the coefficients and exponents in the second polynomial.

5.Set the temporary pointers p and q to traverse the two polynomials respectively.

6.Compare the exponents of two polynomials starting from the first nodes.

(a) If both exponents are equal then add the coefficients and store it in the resultant linked list.

(b) If the exponent of the current term in the first polynomial P is less than the exponent of the current term of the second polynomial is added to the resultant linked list. And move the pointer q to point to the next node in the second polynomial Q.

(c) If the exponent of the current term in the first polynomial P is greater than the exponent of the current term in the second polynomial Q then the current term of the first polynomial is added to the resultant linked list. And move the pointer p to the next node.

(d) Append the remaining nodes of either of the polynomials to the resultant linked list.

Source Code:-

```
#include<stdio.h>
#include<conio.h>
#include<limits.h>
int select();
struct rec
{
float coef;
int exp;
struct rec *next;
};
struct rec *rear;
struct rec *create(struct rec *list);
void *add(struct rec *first,struct rec *second);
struct rec *insert(double coef,int exp,struct rec *rear);
void *display(struct rec *list);
int nodes;
void main()
{
struct rec *first=NULL,*second=NULL;
int choice;
do
{
choice=select();
switch(choice)
{
case 1: first=create(first);continue;
case 2: second=create(second);continue;
case 3: add(first,second);continue;
case 4: puts("END");exit(0);
}
}while(choice!=4);
}
int select()
{
int selection;
```

```
do
{
puts("Enter 1: create the first list");
puts("Enter 2: create the second list");
puts("Enter 3: add the two list");
puts("Enter 4: END");
puts("Entr your choice");
scanf("%d",&selection);
}while((selection<1)||(selection>4));
return (selection);
}


struct rec *create(struct rec *x)
{
float coef;
int exp;
int endexp=INT_MAX;
struct rec *element;
puts("Enter coefs &exp:exp in descending order:""to quit enter 0 for exp");
x=(struct rec *)malloc(sizeof(struct rec));
x->next=NULL;
rear=x;
for(;;)
{
puts("Enter coefficient");
element=(struct rec*)malloc(sizeof(struct rec));
scanf("%f",&coef);
element->coef=coef;
if(element->coef==0.0)break;
puts("Enter exponent");
scanf("%d",&exp);
element->exp=exp;
if((element->exp<=0)||(element->exp>=endexp))
{
puts("Invalid exponent");
break;
}
element->next=NULL;
rear->next=element;
rear=element;
}
x=x->next;
return(x);
}
void *add(struct rec *first,struct rec *second)
```

```
{
float total;
struct rec *end,*rear,*result;
result=(struct rec *)malloc(sizeof(struct rec));
rear=end;
while((first!=NULL)&&(second!=NULL))
{
if(first->exp==second->exp)
{
if((total=first->exp+second->exp)!=0.0)
rear=insert(total,first->exp,rear);
first=first->next;
second=second->next;
}
Else


if(first->exp>second->exp)
{
rear=insert(first->coef,first->exp,rear);
first=first->next;
}else
{
rear=insert(second->coef,second->exp,rear);
second=second->next;
}
}
for(;first;first=first->next)
rear=insert(first->coef,first->exp,rear);
for(;second;second=second->next)
rear=insert(second->coef,second->exp,rear);
rear->next=NULL;
display(end->next);
free(end);
}
void *display(struct rec *head)
{
while(head!=NULL)
{
printf("%2lf",head->coef);
printf("%2d",head->exp);
head=head->next;
}
printf("\n");
}
struct rec *insert(double coef,int exp,struct rec *rear)
```

```
{
rear->next=(struct rec *)malloc(sizeof(struct rec));
rear=rear->next;
rear->coef=coef;
rear->exp=exp;
return(rear);
}
```

Output:

```
Enter 1 : Create the first list
Enter 2 : Create the second list
Enter 3 : Add the two list
Enter 4 : END
Enter  your  choice

1
Enter coefs & exp : exp in descending order : to quit enter 0 for exp
Enter coefficient
5
Enter exponent
4
Enter coefficient
7
Enter exponent
9
Enter coefficient
1
Enter exponent
3
Enter coefficient
0
Enter 1 : Create the first list
Enter 2 : Create the second list
Enter 3 : Add the two list
Enter 4 : END
Enter  your  choice

2
Enter coefs & exp : exp in descending order : to quit enter 0 for exp
Enter coefficient
9
Enter exponent
```

```
3
Enter coefficient
2
Enter exponent
2
Enter coefficient
11
Enter exponent
1
Enter coefficient
5
Enter exponent
0
Invalid exponent
Enter 1 : Create the first list
Enter 2 : Create the second list
Enter 3 : Add the two list
Enter 4 : END
Enter  your  choice
3
5.000000   47.000000   96.000000   32.000000   211.000000   1
Enter 1 : Create the first list
Enter 2 : Create the second list
Enter 3 : Add the two list
Enter 4 : END
Enter  your  choice
4
```

EXPERIMENT No. 10(a)

Aim:- Write programs to implement linked stack and linked queue.

Theory:-

Pushing:-

1. Input the data element to be pushed.
2. Create a NewNode.
3. NewNode → DATA=DATA.
4. NewNode → Next=TOP.
5. TOP=NewNode.
6. Exit.

Popping:-

1. If(TOP is equal to NULL)
   (a) Display "The Stack is empty".
2. Else
   (a) TEMP=TOP.
   (b) Display "The popped element is TOP→DATA".
   (c) TOP=TOP→Next.
   (d) TEMP→Next=NULL.
   (e) Free the TEMP node.
3. EXIT.

Source Code:-

```
#include <stdio.h>
#include <malloc.h>
#include<process.h>
typedef struct link_tag
{
     int data;
     struct link_tag *link;
}node;

/********** Function Declaration begins **********/
node *push(node *);
node *pop(node *);
```

```
void display(node *);
/********** Function Declaration ends **********/

void main()
{
    node *start=NULL;
    int ch;


    printf("\n\t\t Program of stack using linked list");

    do
    {
        printf("\n\t\tMenu");
        printf("\n\t\t1.Push");
        printf("\n\t\t2.Pop");
        printf("\n\t\t3.Display");
        printf("\n\t\t4.Exit");
        printf("\n\t\tEnter choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                start = push(start);
                break;
            case 2:
                start = pop(start);
                break;
            case 3:
                printf("\n\t**** Stack *****\n");
                display(start);
                break;
            case 4:
                exit(0);
            default:
                printf("\nwrong choice : ");
        }
    }
    while (ch!=4);
    printf("\n");
}

/********** Pushing an element in stack **********/
/********** Function Definition begins **********/
```

```
node *push(node *temp)
{
    node *new_node;
    int item;

    printf("Enter an data to be pushed : ");
    scanf("%d",&item);

    new_node = ( node *)malloc(sizeof( node));
    new_node->data = item;
    new_node->link = temp;
    temp = new_node;
    return(temp);
}
/********** Function Definition ends **********/


/********** Popping an element from stack **********/
/********** Function Definition begins **********/
node *pop(node *p)
{
    node *temp;


    if(p == NULL)
        printf("\n***** Empty *****\n");
    else
    {
        printf("Popped data = %d\n",p->data);
        temp = p->link;
        free(p);
        p = temp;
        if (p == NULL)
            printf("\n***** Empty *****\n");
    }
    return(p);
}
/********** Function Definition ends **********/


/********** Displaying elements of Multistack1 **********/
/********** Function Definition begins **********/
void display(node *seek)
{
```

```
    printf("\nTop");
    while (seek != NULL)
    {
        printf("-> %d",seek->data);
        seek = seek->link;
    }
    printf("->NULL\n");
    return;
}
/********** Function Definition ends **********/
```

Output:

```
Program of stack using linked list
    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
  Enter choice : 1

Enter an data to be pushed : 11
    Menu
  1.Push
  2.Pop
  3.Display
  4.Exit
  Enter choice : 1

Enter an data to be pushed : 22
    Menu
  1.Push
  2.Pop
  3.Display
  4.Exit
  Enter choice : 1

Enter an data to be pushed : 33
  Menu
  1.Push
    2.Pop
    3.Display
    4.Exit
```

```
    Enter choice : 3
**** Stack *****

Top-> 33-> 22-> 11->NULL
    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
    Enter choice : 2
Popped data = 33
    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
    Enter choice : 2
Popped data = 22
    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
    Enter choice : 3
**** Stack *****

Top-> 11->NULL
    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
    Enter choice :2
Popped data = 11
***** Empty *****
    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
Enter choice : 4
```

EXPERIMENT No. 10(b)

Aim:- Write programs to implement linked stack and linked queue.

Theory:-

Pushing:-

1. Input the data element to be pushed.
2. Create a NewNode.
3. NewNode → DATA=DATA.
4. NewNode → Next=NULL.
5. If (REAR not equal to NULL)
   (a) REAR → Next=NewNode.
6. REAR=NewNode.
7. Exit.

Popping:-

1. If (FRONT is equal to NULL)
   (a) Display "The Queue is empty".
2. Else
   (a) Display "The popped element is FRONT→DATA".
   (b) If (FRONT is not equal to REAR)
      (i) FRONT=FRONT→Next.
   (c) Else
      FRONT=NULL.
3. EXIT.

Source Code:-

```
#include <stdio.h>
#include <malloc.h>
#include<process.h>

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

/********** Function Declaration begins **********/
void enqueue(node **, node **, int);
void dequeue(node **);
void display(node *);
```

```
/********** Function Declaration ends **********/

void main()
{
        node *front = NULL, *rear = NULL;
        int ch,item;

        printf("\n\t\t Program of queue using linked list");

        do
        {
                printf("\n\t\tMenu");
                printf("\n\t\t1.enqueue");
                printf("\n\t\t2.dequeue");
                printf("\n\t\t3.display");
                printf("\n\t\t4.exit");
                printf("\n\t\tEnter choice : ");
                scanf("%d",&ch);

                switch(ch)
                {
                        case 1:
                                printf("Enter an data to be enqueued : ");
                                scanf("%d",&item);
                                enqueue(&front,&rear,item);
                                break;
                        case 2:
                                dequeue(&front);
                                break;
                        case 3:
                                printf("\n\t**** Queue *****\n");
                                display(front);
                                break;
                        case 4:
                                exit(0);
                        default:
                                printf("\n wrong choice:");
                }
        }
        while (ch!=4);
        printf("\n");
}

/********** Inserting elements in queue **********/
/********** Function Definition begins **********/
void enqueue( node **front,node **rear,int item)
```

```
{
        node *new_node;

        new_node = (node *)malloc(sizeof( node));
        new_node->data = item;
        new_node->link = NULL;

        if ((*front) == NULL)
        {
                (*front) = new_node;
                (*rear) = new_node;
        }
        else
        {
                (*rear)->link = new_node;
                (*rear) = new_node;
        }
        return;
}
/********** Function Definition ends **********/

/********** Deleting element from queue **********/
/********** Function Definition begins **********/
void dequeue(node **front)
{
        node *temp;

        if((*front) != NULL)
        {
                temp = *front;
                (*front) = (*front)->link;
                free(temp);
        }
        return;
}
/********** Function Definition ends **********/

/********** Displaying elements of  queue **********/
/********** Function Definition begins **********/
void display(node *record)
{
        printf("\nRoot");
        while (record != NULL)
        {
                printf("-> %d",record->data);
                record = (record->link);
```

```
        }
        printf("->NULL\n");
        return;
}
/********** Function Definition ends **********/
```

Output:

```
        Program of queue using linked list
        Menu
        1.enqueue
        2.dequeue
        3.display
        4.exit
        Enter choice : 1
Enter an data to be enqueued : 11

        Menu
        1.enqueue
        2.dequeue
        3.display
        4.exit
        Enter choice : 1
Enter an data to be enqueued : 22
        Menu
        1.enqueue
        2.dequeue
        3.display
        4.exit
        Enter choice : 1
Enter an data to be enqueued : 33
        Menu
        1.enqueue
        2.dequeue
        3.display
        4.exit
        Enter choice : 3
**** Queue *****
Root-> 11-> 22-> 33->NULL
        Menu
        1.enqueue
        2.dequeue
        3.display
        4.exit
        Enter choice : 2
        Menu
```

1.enqueue
2.dequeue
3.display
4.exit
Enter choice : 2
Menu
1.enqueue
2.dequeue
3.display
4.exit
Enter choice : 3
**** Queue *****
Root-> 33->NULL
Menu
1.enqueue
2.dequeue
3.display
4.exit
Enter choice : 2
Menu
1.enqueue
2.dequeue
3.display
4.exit
Enter choice :3
**** Queue *****
Root->NULL
Menu
1.enqueue
2.dequeue
3.display
4.exit
Enter choice : 4

EXPERIMENT No. 11(a)

Aim:- Write programs to perform Insertion sort, Selection sort and Bubble sort.

Theory:-

Procedure Selectionsort (A, n):

The above Procedure Subalgorithm sorts the given element of an array 'A' of 'n' number of elements in an ascending order. The smallest element that is located in particular pass is denoted by variable 'S'. The variable 'p' denotes the index of a pass and position of the first element which is to be examined during a particular pass.

Step 1 Loop, repeated (n – 1) times.
Repeat through step 3 for p ← 1, 2, 3 ..... n – 1.
Set S← p.

Step 2 Element with smallest value is obtained in every pass.
Repeat step 2 for i ← p + 1, p + 2 ..... n.
if (A[S] > A[i]) then
Set S ← i.
End of step 2 loop.

Step 3 Exchanging the values
if (S←p) then
Set A[p] ← A[S]
End of step 1 loop.

Step 4 Finish.
Return.

Source Code:-

```
#include <stdio.h>
#include <conio.h>
#define  SIZE  20
/**************Function Declaration Begin**********/
void get_data(int A[],int n);
void selection_sort(int A[],int n);
void show_data(int A[],int n);
/**************Function Declaration End**********/

void main()
{
int n,A[SIZE];
```

```
clrscr();
printf("\n\t\t Program for selection sort");
printf("\n\t\t How many numbers do you want to store in the array : ");
scanf("%d",&n);
get_data(A,n);
selection_sort(A,n);
show_data(A,n);
getch();
}
/********** selection sort technique **********/
/********** Function Definition begins **********/
void selection_sort(int A[],int n)
{
int i,k,pos,min,temp;
for(k=0;k<n;k++)
{
     min = A[k];
     pos=k;
     for(i=k+1;i<n;++i)
     {
          if(A[i]<min)
          {
               min = A[i];
               pos=i;
          }
     }
     temp=A[k];
     A[k]=A[pos];
     A[pos]=temp;
}
}
/********** Function Definition ends **********/
/********** inputting elements **********/
/********** Function Definition begins **********/
void get_data( int A[],int n)
{
int i;
printf("\nEnter %d elements in the array :\n",n);
for (i=0;i<n;i++)
     scanf("%d",&A[i]);
printf("\nArray before sorting : ");
for(i=0;i<n;++i)
     printf("%d  ",A[i]);
printf("\n");
```

```
}
/********** Function Definition ends **********/

/********** displaying elements **********/
/********** Function Definition begins **********/
void show_data(int A[],int n)
{
int i;
printf("\nArray after sorting : ");
for(i=0;i<n;++i)
{
    printf("%d  ",A[i]);
}
printf("\n");
}
/********** Function Definition ends **********/
```

Output:

```
Program for selection sort
How many numbers do you want to store in the array : 6
Enter 6 elements in the array :
    66
    44
    55
    33
    11
    22
Array before sorting : 66  44  55  33  11  22
Array after sorting : 11  22  33  44  55  66
```

EXPERIMENT No. 11(b)

Aim:- Write programs to perform Insertion sort, Selection sort and Bubble sort.

Theory:-

This algorithm sorts the element of an array 'A' (having n elements) in the ascending (increasing) order. The pass counter is denoted by variable 'P' and the variable 'E' is used to count the number of exchanges performed on any pass. The last unsorted element is referred by variable 'l'.

Step 1 Initialization.
Set l ← n, P ← 1.
Step 2 loop
Repeat step 3, 4 while (P←n – 1).
Set E ← 0 R Initializing exchange variable.
Step 3 Comparison, loop.
Repeat for i ← 1, l, ... l – 1.
if (A [i] > A[i + 1]) then
Set A [i] <--> A [i + 1] R Exchanging values.
Set E ← E + 1.
Step 4 Finish, or reduce the size.
if (E = 0) then
Exit.
else
Set l ← l – 1.

Source code:-

```
#include <stdio.h>
#include <conio.h>
#define   SIZE 20
/**************Function Declaration Begin**********/
void get_elements(int A[],int n);
void Bubble_sort(int A[],int n);
void show_elements(int A[],int n);
/**************Function Declaration End**********/
void main()
```

```
{
        int n,A[SIZE];
        clrscr();
        printf("\n\t\t Program for Bubble sort : ");
        printf("\n\n\t\t How many number you want to store : ");
        scanf("%d",&n);
        get_elements(A,n);
        Bubble_sort(A,n);
        show_elements(A,n);
        getch();
}
/********** bubble sort technique **********/
/********** Function Definition begins **********/
void Bubble_sort(int A[],int n)
{
int i,k,temp;
for(k=n-1;k>0;k—)
{
        for(i=0;i<k;++i)
                {
                  if(A[i]>A[i+1])
                  {
                    temp=A[i];
                    A[i]=A[i+1];
                    A[i+1]=temp;
                  }
                }
}
}
/********** Function Definition ends **********/

/********** inputting elements **********/
/********** Function Definition begins **********/
void get_elements( int A[],int n)
{
        int i;
        printf("\n Enter %d elements : \n",n);
        for (i=0;i<n;i++)
```

```
            scanf("%d",&A[i]);
            printf("\n Array before sorting : ");
            for(i=0;i<n;++i)
            printf("%d  ",A[i]);
            printf("\n");
}
/********** Function Definition ends **********/
/********** displaying elements **********/
/********** Function Definition begins **********/
void show_elements(int A[],int n)
{
            int i;
            printf("\n Array after sorting : ");
            for(i=0;i<n;++i)
            {
            printf("%d  ",A[i]);
            }
            printf("\n");
}
/********** Function Definition ends **********/
```

Output:

```
Program for Bubble sort :
How many number you want to store : 6
Enter 6 elements :
            66
            44
            55
            33
            11
            22
 Array before sorting : 66  44  55  33  11  22
 Array after sorting : 11  22  33  44  55  66
```

EXPERIMENT No. 11(c)

Aim:- Write programs to perform Insertion sort, Selection sort and Bubble sort.

Theory:-

Procedure Insertionsort (A, n):

The Procedure Subalgorithm sorts the elements of an array 'A' in an ascending order. The array consists of 'n' number of elements. The variable 'i' is used for index and 't' is a temporary variable.

Step 1	Loop, uptil length [A]. Repeat step 1, 2 for i ← 2, 3, 4, 5, --- n Set t ←A[i], p ← i – 1 Temporary variable is set to new value, pointer is adjusted.
Step 2	Loop, comparison Repeat while (p > 0 and t < A[p]) Set A [p +1] ←A[p], p←p – 1. End of step 2 loop.
Step 3	Inserting element in appropriate place Set A [p + 1] ← t. End of step 1 loop.
Step 4	Finished. Return.

Source Code:-

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10

/**************Function Declaration Begin**********/
void insertion_sort(int A[],int);
/**************Function Declaration End**********/

void main()
{
int A[SIZE],n,i;
clrscr();
printf("\n\t\t Program for Insertion Sort:");
printf("\nHow many elements do you want to sort:\t");
```

```
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter A[%d] ",i+1);
scanf("%d",&A[i]);
}
printf("\n\t\t Elements before sorting : ");
for(i=0;i<n;i++)
{
printf("\nEnter A[%d]:%d",i+1,A[i]);
}
insertion_sort(A,n);
getch();
}

/********** insertion sort technique **********/
/********** Function Definition begins **********/
void insertion_sort(int A[], int n)
{
int i,j,count=0,count1=0,temp;
for(i=1;i<n;i++)
{
  for(j=i;j>0;j—)
   {
     if(A[j]<A[j-1])
      {
         temp=A[j];
        A[j]=A[j-1];
       A[j-1]=temp;
      }
    count++;
   }
  count1++;
}
  printf("\n Count : = %d+%d=%d",count,count1,count+count1);
  printf("\n\t\t Elements after sorting : ");
  for(i=0;i<n;i++)
  {
     printf("\nEnter A[%d] : %d",i+1,A[i]);
  }
}
  /********** Function Definition ends **********/
```

Output:

```
Program for Insertion Sort :
How many elements do you want to sort :  6
Enter A[1]66
Enter A[2]44
Enter A[3]55
Enter A[4]33
Enter A[5]11
Enter A[6]22
Elements before sorting :
Enter A[1]:66
Enter A[2]:44
Enter A[3]:55
Enter A[4]:33
Enter A[5]:11
Enter A[6]:22
 Count : =15+5=20
Elements after sorting :
Enter A[1]:11
Enter A[2]:22
Enter A[3]:33
Enter A[4]:44
Enter A[5]:55
Enter A[6]:66
```

EXPERIMENT No. 12

Aim:- Write a program to perform Quick sort.

Theory:-

Quick_Sort(a,l,h):

a → represents the list of elements.
l → represents the position of the first element in the list(only at the starting point, it's value change during the execution of the function).
h → represents the position of the last element in the list(only at starting point the value of it's changes during the execution of the function).

Step 1: [Initially]
low=l.
high=h.
key=a[(l+h)/2][middle element of the element of the list].

Step 2: Repeat through step 7 while (low<=high).
Step 3: Repeat step 4 while (a([low]<key)).
Step 4: low=low+1.
Step 5: Repeat step 6 while (a([high]<key)).
Step 6: high=high-1.
Step 7: if(low<=high)
(i) temp=a[low].
(ii) a[low]=a[high].
(iii) a[high]=temp.
(iv) low=low+1.
(v) high=high-1.
Step 8: if (i<high) Quick_Sort(a,l,high).
Step 9: if (low<h) Quick_Sort(a,low,h).
Step 10: Exit.

Source Code:-

```
#include <stdio.h>
#include <conio.h>
#define   SIZE 20

/**************Function Declaration Begin**********/
void Quick_sort(int A[],int n);
int Partition(int A[],int n);
void get_elements(int A[],int n);
```

```
void show_elements(int A[],int n);
/**************Function Declaration End**********/

void main()
{
     int n,A[SIZE];
     clrscr();
     printf("\n\t\t Program for Quick sort : ");
     printf("\n\n\t\t How many numbers you want to store in the array : ");
     scanf("%d",&n);
     get_elements(A,n);
     Quick_sort(A,n);
     show_elements(A,n);
     getch();
}

/********** inputting elements **********/
/********** Function Definition begins **********/
void get_elements( int A[],int n)
{
     int i;
     printf("\n Enter %d elements\n",n);
     for (i=0;i<n;i++)
          scanf("%d",&A[i]);
     printf("\n Array before sorting : ");
     for(i=0;i<n;++i)
          printf("%d  ",A[i]);
     printf("\n");
}
/********** Function Definition ends **********/

/********** displaying elements **********/
/********** Function Definition begins **********/
void show_elements(int A[],int n)
{
     int i;
     printf("\n Array after sorting : ");
     for(i=0;i<n;++i)
     {
          printf("%d  ",A[i]);
     }
     printf("\n");
}
/********** Function Definition ends **********/
```

```
/****************** Quick Sorting technique ****************/
/********** Function Definition begins **********/
void Quick_sort(int A[],int n)
{
	int pivot;
	if(n<=1)
	{
		return;
	}
	pivot=Partition(A,n);
	Quick_sort(A,pivot);
	Quick_sort(A+pivot+1,n-pivot-1);
}
/********** Function Definition ends **********/

/****************** partitioning technique ****************/
/********** Function Definition begins **********/
int Partition(int A[],int n)
{
	int pivot,l,r,s;
	pivot=A[0]; /* Fixing first element as pivot*/
	l=0; r=n-1;
	for( ; ; )
	{
		while(l<r && A[l]<=pivot)
		{
			l++;
		}
		while(l<r && A[r]>pivot)
		{
			r—;
		}
		if(l==r)
		{
			if(A[r]>pivot)
			{
				l=l-1;
			}
		break;
		}
		s=A[l];
		A[l]=A[r];
		A[r]=s;
```

```
    }
    s=A[0];
    A[0]=A[l];
    A[l]=s;
    return l;
}
/********** Function Definition ends **********/
```

Output:

```
Program for Quick sort :
How many numbers you want to store in the array : 6
Enter 6 elements
    66
    55
    44
    33
    22
    11
Array before sorting : 66  55  44  33  22  11
Array after sorting : 11  22  33  44  55  66
```

EXPERIMENT No. 13

Aim:- Write a program to perform Heap sort.

Theory:-

Procedure Heap-sort (H):

The above procedure sorts the element of a given array using heap sort technique. The procedure makes a call to Build-Heap, and Heapify.

Step 1 Building.
Call to Build-heap (H).
Step 2 Loop, exchanging root, and fixing the heap.
for i← length [H] down to 2
Set H [1] ← H [i].
Set heap-size [H] ←heap-size [H] – 1.
Call to Heapify (H, 1).
Step 3 Return at the point of call.

Procedure Heapify (H, i):

The above Procedure fixes the heap for index 'i' where 'i' refers to the index such that the tree rooted at H[i] is a heap. This Procedure recursively calls itself until the given heap is fixed.

Step 1 Left child, and right child, initialization.
Set l ← Lchild (i).
Set r ← Rchild (i).
Step 2 Place the maximum value, left child.
if (l ← heap-size [H] and H [l] > H[i]) then
Set max ← l.
else
Set max ← i.
Step 3 Place the maximum value, right child.
if (r <= heap-size [H] and H [r] > H [max]) then
Set max ← r.
Step 4 Checking with parent
if (max ≠ i) then
Set H [i] ← H [max].
Step 5 Fixing of heap.
Call to Heapify (H, max).

Source Code:-

```
#include <stdio.h>
#include <conio.h>
#define    SIZE  20

/**************Function Declaration Begin**********/
void get_elements(int A[],int n);
void movedown(int pos,int A[],int n);
void Heap_sort(int A[],int n);
void show_elements(int A[],int n);
/**************Function Declaration End**********/

void main()
{
     int n,A[SIZE];
     clrscr();
     printf("\n\t\t Program for Heap sort : ");
     printf("\n How many numbers you want to store in the array : ");
     scanf("%d",&n);
     get_elements(A,n);
     Heap_sort(A,n);
     show_elements(A,n);
     getch();
}

/********** heapify & adjusting element position **********/
/********** Function Definition begins **********/
void movedown(int pos,int A[],int n)
{
     int k,r,l,max,temp;
     for(k=pos;;)
     {
          l=2*k+1;
          r=l+1;
          if(l>=n)
          {
               return;
          }
          else
               if(r>=n)
               {
                    max=l;
```

```
                }
                else
                        if(A[l]>A[r])
                        {
                                max=l;
                        }
                        else
                        {
                                max=r;
                        }
            if(A[k]>A[max])
            {
                return;
            }
            temp=A[k];
            A[k]=A[max];
            A[max]=temp;
            k=max;
    }
}
/********** Function Definition ends **********/

/******************** heap sorting technique *****************/
/********** Function Definition begins **********/
void Heap_sort(int A[],int n)
{
    int i,temp;
    for(i=n/2;i>=0; —i)    /* Performimg Heapify */
    {
        movedown(i,A,n);
    }
    for(i=n-1;i>0;i—)
    {
        temp=A[0];
        A[0]=A[i];
        A[i]=temp;
        movedown(0,A,i);
    }
}
/********** Function Definition ends **********/

/********** inputting elements **********/
/********** Function Definition begins **********/
void get_elements( int A[],int n)
```

```
{
    int i;
    printf("\n Enter %d elemets : \n",n);
    for (i=0;i<n;i++)
        scanf("%d",&A[i]);
    printf("\n Array before sorting : ");
    for(i=0;i<n;++i)
        printf("%d  ",A[i]);
    printf("\n");
}
/********** Function Definition ends **********/

/********** displaying elements **********/
/********** Function Definition begins **********/
void show_elements(int A[],int n)
{
    int i;
    printf("\n Array after sorting : ");
    for(i=0;i<n;++i)
    {
        printf("%d  ",A[i]);
    }
    printf("\n");
}
/********** Function Definition ends **********/
```

Output:

```
Program for Heap sort :
How many numbers you want to store in the array : 6
Enter 6 elemets :
    66
    44
    55
    22
    33
    11
 Array before sorting : 66  44  55  22  33  11
 Array after sorting : 11  22  33  44  55  66
```

EXPERIMENT No. 14

Aim:- Write a program to perform Merge sort.

Theory:-

Procedure Mergesort (A, start, finish):

The above Procedure Subalgorithm recursively sorts given list of elements between position "start' and "finish" (inclusive) . Array 'A' contains 'n' number of elements. Variables 'length' and 'mid' refer to the number of elements in the current sublist and position of the middle element of the sublist, respectively.

Step 1 Computation, size of current sublist.
Set length ← finish – start + 1.

Step 2 Condition checking, if length is one
if (length ← 1) then
Return.

Step 3 Calculating, middle point
Set mid ← sort + [length/2]– 1.

Step 4 Solving I sublist, recursively.
Call Mergesort (A, start, mid).

Step 5 Solving II sublist, recursively.
Call Mergesort (A, mid + 1, finish).

Step 6 Merging two sublists, sorted.
Call Merge (A, start, mid + 1, finish).

Step 7 Finish.
Return.

Procedure Merge (A, first, second, third):

The above procedure subalgorithm merges the two lists and produces a new sorted list. Temporary array 'temp' is used for holding sorted values.

Step 1 Initialisation.
Set n ← 0, f ←first, S←second.

Step 2 Comparison, giving smallest element.
if (A[f] ← A[S]) then
Set n ← n + 1.
Set temp [n] ← A[f].
Set f ← f + 1.
else
Set n ← n + 1.

Set temp [n] ←A[S].
Set S←S + 1.

Step 3 Copying remaining elements
if (f ← second) then
Loop
Repeat while (S←third)
Set n ←n + 1.
Set temp [n] ←A[S].
Set S←S + 1.
End loop.
else
Loop
Repeat while (f < second)
Set n←n + 1.
Set temp [n] ← A[S].
Set f ← f + 1.
End of loop.

Step 4 Copying, element to original array
Loop
Repeat for f←1, 2, ---- n.
A [first – 1 + f] ← temp [f].
End of loop.

Step 5 Finish.
Return.

Source Code:-

```
#include<stdio.h>
#include<conio.h>
#define MAX 30

void main()
{
   int arr[MAX],temp[MAX],i,j,k,n,size,l1,l2,h1,h2;
   clrscr();
   printf("\n enter the number of elements : ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
     printf("\n enter element %d : ",i+1);
     scanf("%d",&arr[i]);
   }
printf("\nunsorted list is : ");
for(i=0;i<n;i++)
```

```
 {
   printf("%d ", arr[i]);

/*  l1 lower bound of first pair and so on */

   for( size=1;size<n;size=size*2)
      {
         l1 =0;
k=0;
while(l1+size<n)
{
  h1 = l1 + size -1;
  l2 = h1 + 1;
  h2 = l2 +size -1;
  if( h2>=n)      /* h2 exceeds the limit of array */
    h2 = n-1;
  i=l1;
  j=l2;
  while( i<=h1 && j<=h2)
   {
      if(arr[i] <= arr[j])
                  temp[k++] = arr[i++];
              else
                  temp[k++] = arr[j++];
            }
          while(i<=h1)
            temp[k++] = arr[i++];
           while( j<=h2)
            temp[k++] = arr[j++];   /* merging completed */
           l1 = h2 +1;
          /* take the next two pair of merging */
       } /*end of while*/
for(i=l1;k<n;i++)/* any pair left*/
temp[k++]=arr[i];
for(i=0;i<n;i++)/*any pair left*/
arr[i]=temp[i];
printf("\nsize=%d \n elements are : ", size);
for(i=0;i<n;i++)
printf("%d",arr[i]);
}
/*end of for loop*/
printf(("\nsorted list is :"\n");
for(i=0;i<n;i++)
printf("%d",arr[i]);
getch();
}
```

```
/*end of main()*/
```

Output:

```
Enter the number of elements : 5
Enter element 1 : 3d
Enter element 2 : 1
Enter element 3 : 12
Enter element 3 : 8
Enter element 3 : 9
Unsorted list is : 3 1 12 8 9
Size=5
Elements are : 3 1 12 8 9
Sorted list is :
1 3 8 9 12
```

EXPERIMENT No. 15

Aim:- Write a program to create a binary search tree and perform insertion, deletion, and traversal.

Theory:

A particular form of binary tree suitable for searching.

A binary search tree is a binary tree that is either empty or in which each node contains a key that satisfies the following conditions:

- All keys (if any) in the left subtree of the root precede the key in the root.
- The key in the root precedes all keys (if any) in its right subtree.
- The left and right subtrees of the root are again binary search trees.

Source Code:-

```
#include<stdio.h>
struct rec
{
        long num;
        struct rec *left;
        struct rec *right;
};
struct rec *tree=NULL;
struct rec *tree;
struct rec *delnum(long digit,struct rec *r);
struct rec *insert(struct rec *tree,long num);
struct rec *deletenode(long digit,struct rec *tree);
void search(struct rec *tree,long num);
void preorder(struct rec *tree);
void inorder(struct rec *tree);
void postorder(struct rec *tree);
void main()
{
int choice;
long digit;
int element;
        do
          {
                choice=select();
                switch(choice)
                 {
                  case 1: puts("Enter integer : To quit enter 0");
                          scanf("%ld",&digit);
```

```
                        while(digit!=0)
                                {
                                  tree=insert(tree,digit);
                                  scanf("%ld",&digit);
                                }continue;
                    case 2: puts("Enter the number to be search ");
                            scanf("%ld",&digit);
                            search(tree,digit);
                            continue;
                    case 3: puts("\npreorder traversing TREE");
                            preorder(tree);continue;
                    case 4: puts("\ninorder traversing TREEE");
                            inorder(tree);continue;
                    case 5: puts("\npostorder traversing TREE");
                            postorder(tree);continue;
                    case 6: puts("Enter element which do you wanbt delete from  the BST
");
                             scanf("%ld",&digit);
                            deletenode(digit,tree);continue;
                    case 7: puts("END");exit(0);
                  }
        }while(choice!=7);
}
int select()
{
int selection;
        do
          {
                puts("Enter 1: Insert a node in the BST");
                puts("Enter 2: Search a node in BST");
                puts("Enter 3: Display(preorder)the BST");
                puts("Enter 4: Display(inorder)the BST");
                puts("Enter 5: Display(postorder) the BST");
                puts("Enter 6: Delete the element");
                puts("Enter 7: END");
                puts("Enter your choice");
                scanf("%d",&selection);
                    if((selection<1)||(selection>7))
                        {
                          puts("wrong choice:Try again");
                          getch(); }
          }while((selection<1)||(selection>7));
          return (selection);
}
struct rec *insert(struct rec *tree,long digit)
{
```

```
        if(tree==NULL)
          {
                tree=(struct rec *)malloc(sizeof(struct rec));
                tree->left=tree->right=NULL;
                tree->num=digit;
          }
    else
          if(digit<tree->num)
                tree->left=insert(tree->left,digit);
    else
          if(digit>tree->num)
                tree->right=insert(tree->right,digit);
    else if(digit==tree->num)
            {
              puts("Duplicate node:program exited");exit(0);
            }
            return(tree);
}
struct rec *delnum(long digit,struct rec *r)
{
struct rec *q;
if(r->right!=NULL)delnum(digit,r->right);
else
q->num=r->num;
q=r;
r=r->left;
}
struct rec *deletenode(long digit,struct rec *tree)
{
struct rec *r,*q;
if(tree==NULL)
{
puts("Tree is empty.");
exit(0);
}
if(digit<tree->num)
deletenode(digit,tree->left);
else
if(digit>tree->num)deletenode(digit,tree->right);
q=tree;
if((q->right==NULL)&&(q->left==NULL))
q=NULL;
else
if(q->right==NULL)tree=q->left;else
if(q->left==NULL)tree=tree=q->right;else
delnum(digit,q->left);
```

```
free(q);
}
void search(struct rec *tree,long digit)
{
        if(tree==NULL)
          puts("The number does not exits\n");
  else
        if(digit==tree->num)
        printf("Number=%ld\n" ,digit);
  else
        if(digit<tree->num)
          search(tree->left,digit);
  else
         search(tree->right,digit);
}
void preorder(struct rec *tree)
{
        if(tree!=NULL)
          {
                printf("%12ld\n",tree->num);
                preorder(tree->left);
                preorder(tree->right);
          }
}
void inorder(struct rec *tree)
{
        if(tree!=NULL)
           {
                inorder(tree->left);
                printf("%12ld\n",tree->num);
                inorder(tree->right);
           }
}
void postorder(struct rec *tree)
{
        if(tree!=NULL)
          {
                postorder(tree->left);
                postorder(tree->right);
                printf("%12ld\n",tree->num);
          }

}
```

Output:

```
Enter 1: Insert a node in the BST
Enter 2: Search a node in the BST
Enter 3: Display (preorder) the BST
Enter 4: Display (inorder) the BST
Enter 5: Display (postorder) the BST
Enter 6: Delete the element
Enter 7: END

Enter your choice
1
Enter integer : to quit enter 0
82
77
90
346
35
0
Enter 1: Insert a node in the BST
Enter 2: Search a node in the BST
Enter 3: Display (preorder) the BST
Enter 4: Display (inorder) the BST
Enter 5: Display (postorder) the BST
Enter 6: Delete the element
Enter 7: END

Enter your choice
2
Enter the number to be search
35
Number = 35
Enter 1: Insert a node in the BST
Enter 2: Search a node in the BST
Enter 3: Display (preorder) the BST
Enter 4: Display (inorder) the BST
Enter 5: Display (postorder) the BST
Enter 6: Delete the element
Enter 7: END

Enter your choice
3
Preorder traversing TREE
82
77
346
```

```
90
35
Enter 1: Insert a node in the BST
Enter 2: Search a node in the BST
Enter 3: Display (preorder) the BST
Enter 4: Display (inorder) the BST
Enter 5: Display (postorder) the BST
Enter 6: Delete the element
Enter 7: END

Enter 1: Insert a node in the BST
Enter 2: Search a node in the BST
Enter 3: Display (preorder) the BST
Enter 4: Display (inorder) the BST
Enter 5: Display (postorder) the BST
Enter 6: Delete the element
Enter 7: END

Enter your choice
4
Inorder traversing TREE
346
77
82
90
35
Enter 1: Insert a node in the BST
Enter 2: Search a node in the BST
Enter 3: Display (preorder) the BST
Enter 4: Display (inorder) the BST
Enter 5: Display (postorder) the BST
Enter 6: Delete the element
Enter 7: END

Enter your choice
5
Postorder traversing TREE
346
77
35
90
82
Enter 1: Insert a node in the BST
Enter 2: Search a node in the BST
Enter 3: Display (preorder) the BST
Enter 4: Display (inorder) the BST
```

Enter 5: Display (postorder) the BST
Enter 6: Delete the element
Enter 7: END

Enter your choice
6
Enter which element do you want to delete from the BST
12
Enter 1: Insert a node in the BST
Enter 2: Search a node in the BST
Enter 3: Display (preorder) the BST
Enter 4: Display (inorder) the BST
Enter 5: Display (postorder) the BST
Enter 6: Delete the element
Enter 7: END

Enter your choice
7

EXPERIMENT No. 16(a)

Aim:- Write a program for traversal of graph (B.F.S., D.F.S.).

Theory:-

Procedure BFS (G, S):

The above Procedure computes the breadth-first-search for a given graph 'G'. Variable 'S' refers to the start of graph 'G'. Three arrays color [u] holds the color of u, pre [u] points to the predecessor of u, and dist [u] calculates the distance from S to u. The subalgorithms Enqueue and Dequeue are also used for inserting and deleting element from the queue.

```
Step 1   Initialization, loop.
         for u ←V1, V2, .....        R   For each u in V
         {
         Set color [u] ← white.
         Set dist [u] ← ∞.
         Set pre [u] ←NULL.
         }
Step 2   Intializing source S, placing 'S' in the queue.
         Set color [S] ← gray.
         Set dist [S] ← 0.
         Set Q ← {S} R putting S in the queue.
Step 3   Loop, while no more adjacent vertices
         while (Q ←NULL)
         {
         Set u ← Dequeue (Q) R u is the next to visit.
         for V ← Vk .... Vn        R for each V in adj [u]
         {
         if (color [V] = white) then { R if neighbour unreached
         Set color [V] ←gray  R mark it reached.
         Set dist [V] ← dist [u] + 1  R set its distance.
         Set pre [V] ← u   R set its predecessor.
         call to Enqueue (Q, V)  R put in the queue.
         }
         Set color [u] ← black R u is visited.
         }
Step 4   Return at the point of call.
         Return.
```

Source Code:-

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
#define FALSE 0
#define TRUE 1

typedef int adj_mat[SIZE][SIZE];
int front=1,rear=1;
int q[SIZE];

typedef struct graph_t{
                        int nodes;
                        int *visited;
                        adj_mat mat;
                    }graph;

/**************Function Declaration Begin**********/
void BFS(graph *);
void add_queue(int[],int);
int delete_queue();
/**************Function Declaration End**********/
void main()
{
    graph G;
    clrscr();
    printf("\n\t\t Program shows Breath First Search in a graph ");
    printf("\n\t\t Enter number of nodes in the graph : ");
    scanf("%d",&G.nodes);
    BFS(&G);
    getch();
}

/********** breadth first searching **********/
/********** Function Definition begins **********/
void BFS( graph *G )
{
    int k,i,j;
    for(k=1;k<=G->nodes;k++)
        G->visited[k] = FALSE;
    for(i=1;i<=G->nodes;i++)
    {
        for(j=1;j<=G->nodes;j++)
```

```
        {
            printf("\n Enter data of vertex %d for(%d,%d) : ",i,i,j);
            printf("\n Enter 1 for adjacent vertex and 0 otehrwise ");
            scanf("%d",&G->mat[i][j]);
        }
    }
    for(k=1;k<=G->nodes;k++)
    {
        if ( !G->visited[k] )
        {
                add_queue(q,k);
                do
                {
                    k= delete_queue(q);
                    G->visited[k] = TRUE;
                    for(j=1;j<=G->nodes;j++)
                    {
                        if(G->mat[k][j] == 0)
                            continue;
                            if (!G->visited[j])
                            {
                                G->visited[j] = TRUE;
                                add_queue(q, j);
                            }
                    }
                }while(front!=rear);
        }
    }
        printf("\n Adjacency matrix of a graph is :\n");
    for(i=1;i<=G->nodes;i++)
    {
        for(k=1;k<=G->nodes;k++)
        {
            printf("%d\t",G->mat[i][k]);
        }
        printf("\n");
    }
    i=0;
    printf("\n Traversal of a given graph is \n");
    while(i<G->nodes)
    {
        printf("%d\t",q[++i]);
    }
}
/********** Function Definition ends **********/
```

```
/********** inserting element in queue **********/
/********** Function Definition begins **********/
void enqueue(int q[], int k)
{
    q[rear] = k;
    rear++;
}
/********** Function Definition ends **********/

/********** deleting element from queue **********/
/********** Function Definition begins **********/
int dequeue(int q[])
{
    int data;
    data = q[front];
    front++;
    if(front==SIZE)
    {
        front=1;
        rear=1;
    }
    return(data);
}
/********** Function Definition ends **********/
```

Output:

```
Program shows the traversal of graph using breadth first search
Enter number of nodes in the graph : 3

Enter data of vertex 1 for (1,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (1,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 1

Enter data of vertex 1 for (1,3) :
Enter 1 for adjacent vertex and 0 for otherwise : 1

Enter data of vertex 1 for (2,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0
```

Enter data of vertex 1 for (2,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (2,3) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,4) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Adjacency matrix of the graph is
0 1 1
0 0 0
0 0 0
Traversal of a given graph is
1 2 3

EXPERIMENT No. 16(b)

Aim:- Write a program for traversal of graph (B.F.S., D.F.S.).

Theory:-

Procedure DFSvisit (u):

The above Procedure subalgorithm processes the given vertex. It makes a recursive call to itself. The arrays used in this procedure have been previously described.

Step 1 Start search at u, mark u visited.
Set color [u] ← gray.
Set time ← time + 1.
Set dis [u] ← time.
Loop,
for V ← Vk .... Vn R for each V in Adj [u]
{
if (color [V] = White) then R if neighbour marked unreached
Set pre [V] ← u R set predecessor pointer.
Call to DFS visit (V) R processed V.
}
End Loop
Step 2 U, is visited
Set color [u] ← black.
Set time ← time + 1.
Set pre [u] ← time.
Step 3 Return at the point of call
Return.

Procedur DFS (G):

The above Procedure computes the depth-first-search of the given 'G' graph 'G'. It takes the advantage of Procedure DFS visit ( ). All the auxillary arrays used in this procedure have been previously described.

Step 1 Loop, initialization.
for u ← V1, V2 .... Vn R for each u in V
{
Set color [u] ← white.
Set pre [u] ← NULL.
}
Step 2 Setting time
Set time ← 0.
Step 3 Loop, finding unreached vertex and start new search

for u ← V1, V2 .... V R for each u in V
{
if (color = white) R found unreached vertex
Call to DFS visit (u) R start a new search.
}
Step 4 Return at point of call
Return.

Source Code:-

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
#define FALSE 0
#define TRUE 1
typedef int adj_mat[SIZE][SIZE];
typedef struct graph_t{
                int nodes[SIZE];
                int n;
                int *visited;
                adj_mat mat;
                }graph;

/**************Function Declaration Begin**********/
void DFS(graph *);
void visit(graph *,int);
/**************Function Declaration End**********/

static int find=0;
void main()
{
        graph G;
        //clrscr();
        printf("\n\t\t Program shows the traversal of graph using Depth First Search
");
        printf("\n\t\t Enter number of nodes in the graph : ");
        scanf("%d",&G.n);
        DFS(&G);
getch();
}
/********** depth first searching **********/
/********** Function Definition begins **********/
void DFS( graph *G )
```

```
{
    int k,i,j;
    for(k=1;k<=G->n;k++)
        G->visited[k] = FALSE;
    for(i=1;i<=G->n;i++)
    {
        for(j=1;j<=G->n;j++)
        {
            printf("\n Enter data of vertex %d for(%d,%d) :\n",i,i,j);
            printf("\n Enter 1 for adjacent vertex and 0 for otherwise : ");
            scanf("%d",&G->mat[i][j]);
        }
    }
    for(k=1;k<=G->n;k++)
    {
            if ( !G->visited[k] )
                visit(G, k);
    }
    printf("\n Adjacency matrix of the grpah is \n");
    for(i=1;i<=G->n;i++)
    {
        for(k=1;k<=G->n;k++)
        {
            printf("%d\t",G->mat[i][k]);
        }
        printf("\n");
    }
    i=0;
    printf("\n Traversal of a given graph is \n");
    while(i<G->n)
    {
        printf("%d\t",G->nodes[++i]);
    }
}
/********** Function Definition ends **********/
/********** visiting graph **********/
/********** Function Definition begins **********/
void visit( graph *G, int k )
{
  int j;
  G->visited[k] = TRUE;
  G->nodes[++find] = k;
  for(j=1;j<=G->n;j++)
  {
```

```
            if(G->mat[k][j] == 1)
            {
                  if (!G->visited[j])
                        visit( G, j );
            }
  }
}
/********** Function Definition ends **********/
```

Output:

```
Program shows the traversal of graph using depth first search
Enter number of nodes in the graph : 3

Enter data of vertex 1 for (1,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (1,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 1

Enter data of vertex 1 for (1,3) :
Enter 1 for adjacent vertex and 0 for otherwise : 1

Enter data of vertex 1 for (2,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (2,2) :
Enter 1 for adjacent vertex and 0 for otherwise:0

Enter data of vertex 1 for (2,3):
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,4) :
Enter 1 for adjacent vertex and 0 for otherwise : 0
```

Adjacency matrix of the graph is
0 1 1
0 0 0
0 0 0
Traversal of a given graph is
1 2 3