Algorithm :

Step 1 : Start
Step 2 : Read str [50], rev str[50]
Step 3 : * strptr = str
Step 4 : * revptr = revstr
step 5 : len = -1.
step 6 : print the entered string str
step 7 : while (*strpt) then strptr++.
step 8 : len ++ & Repeat step 11-14 while (len>=0).
step 9 : strptr--
Step 10 : * revptr = strptr
Step 11 : revptr ++. and len *--
Step 12 : * revptr = '\0'
Step 13 : print the reverse of a string revstr.
Step 14 : Stop.

Prog. No.....

Date.....

Page No.....

AIM :

Program to reverse a string using pointer.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int main() {
    char str[50];
    char revstr[50];
    char *strptr=str;
    char *revptr=revstr;
    int len=-1
    printf("Enter the String : \n");
    scanf("%s", str);
    while (*strptr)
    {
      strptr++;
      len++;
    }
    while (len>=0) {
      strptr--;
      *revptr = *strptr;
      revptr++;
      -len
    }
```

output :

```
Enter the string :
Hello
reverse of a string is:
olleH
```

Prog. No.....

Date.....

Page No.....

```
    *revptr = '\0';
    printf("reverse of a string is: \n
                                %s", revstr);
    return 0;
}
```

Algorithm :

Step 1 : Start
Step 2 : Declare char txt[20], pat[20]
Step 3 : Read a, b, i, j
Step 4 : Enter the string
Step 5 : Enter the pattern to find.
Step 6 : Assign the length of string to variable
Step 7 : Assign the length of text to variable
Step 8 : Check for (i=0; i<=a-b; i++) then
for (j=0; j<a; j++)
Step 9 : if txt[+j] not equal to pat[j], then break the for loop, then go to Step 12
Step 10 : if value of j is equal to the value of a.
Step 11 : Then print pattern found at index j+2.
Step 12 : Stop.

output :

Enter the string : Hello world
Enter the pattern to find : w
pattern found at index 6

Prog. No.....
Date.....
Page No.....

AIM :

program to implement pattern matching algorithm

SOURCE CODE :

```
#include<stdio.h>
#include<string.h>
#include<conio.h>

void main() {
    char txt[20], pat[20];
    int a,b, i, j;
    printf("Enter the String:");
    gets(txt);
    printf("Enter the pattern to find");
    gets(pat);
    a= Strlen(pat);
    b= Strlen(txt);
    for (i=0; i<=b-a; i++) {
        for(j=0; j<a; j++)
            if (txt[i+i] != pat[j])
            break;
            if (j==a)
                printf("pattern found
                    at index %d", i+1);
    }
}
```

Algorithm:

Step 1 : Start
Step 2 : Input m, n, i, j, srchno
Step 3 : set count = 0, a[50][50]
Step 4 : Repeat steps 5-6 while (i<m)
Step 5 : Go to Step 6 while (i<n)
Step 6 : a[i][j] an array is created.
Step 7 : Enter the search number (srchno)
Step 8 : Repeat the Step 9-10 while (j<m)
Step 9 : Go to Step 10 while (j<n)
Step 10 : Check a[i][j] = srchno.
Step 11 : If true print the index position
Step 12 : If (count == 0) print "not found"
Step 13 : STOP.

Prog. No.....

Date.....

Page No.....

AIM :

Program to search an element in the 2-dimensional array.

SOURCE CODE :

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

void main() {
    int m,n, i, j, srchno, count=0, a[50][50];
    printf("Enter number of rows
                and column : \n");
    scanf("%d %d", &m, &n);
    printf("Enter %d %d elements: \n",
                                (m*n));
    for(j=0; j<n; j++) {
        scanf("%d", &a[i][j]);
    }
    }
    printf("Enter elements to get the
                position : \t");
    scanf("%d", &srchno);
    for(i=0; i<m; i++) {
        for(j=0; j<n; j++) {
            if(a[i][j] == srchno)
            {
```

Output:

```
Enter number of rows and column:
2 2
Enter 4 elements:
2 3
4 5
Enter elements to get the position:
3
(0 1)
```

Prog. No.....

Page No.....

Date.....

```
                printf ("(%d %d) \n", i,j);
                count ++;
            }
        }
    }
    if (count ==0)
        printf ("not found ");
}
```

Algorithm:

Step 1: Start
Step 2: Accept variables m,n to store size of 1st and 2nd array.
Step 3: input elements into ar[50], br[50]
Step 4: Repeat Step 5 while (i<m)
Step 5: cr[k] = ar[i]
Step 6: Repeat Step 7 while (j<n)
Step 7: cr[k+j] = br[j]
Step 8: print appended array using for loop until j(m+n) satisfies
Step 9: Stop.

Prog. No.....
Date.....
Page No.....

AIM:

program to append 2 arrays.

SOURCE CODE:

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

void main () {
    int ar[30], br[30], cr[30], i, j, m, n;
    printf ("Enter limit of 1st array:");
    scanf ("%d", &m);
    printf ("Enter limit of 2nd array:");
    scanf ("%d", &n);
    printf("Enter elements of 1st array:");
    scanf ("%d"
    for (i=0; i<m; i++) {
        scanf("%d", &ar[i]);
    }
    printf ("Enter elements of 2nd array:");
    for(j=0; j<n; j++) {
        scanf ("%d", &br[j]);
    }
    for(i=0; i<m; i++)
    cr[i] = ar[i];
    for(j=0; j<n; j++)
    cr[i+j] = br[j];
```

Output :

```
Enter limit of 1st array : 2
Enter limit of 2nd array: 2
Enter elements of 1st array: 10 20
Enter elements of 2nd array: 30 50
After appending array is : 10 20 30 50
```

Prog. No.....

Date.....

Page No.....

```
printf("After appending array is:");
for(i=0 ; i<m+n ; i++)
{
    printf("%d\t", cr[i]);
}
}
```

Algorithm:

Step1 : Start

Step2 : Accept a value in max as the number of elements of the array.

Step3 : Accept max element into the array list.

Step4 : Accept the value to be searched in the variable item.

Step5 : Store the position of 1st element of the list in first and that of last in last

Step6 : Repeat Step 7 to 11 till while (first <= last)

Step7 : Find middle position using the formula (First/ last)/2 and store it in middle.

Step8 : Compare the search value in item with the element at middle of list.

Step9 : If the middle element contains the search value in item then stop search. display the position and go to Step 12.

Step10 : If search value is smaller than the middle element then set last = middle-1.

Step11 : If the search value is larger than middle element then set first = middle +1.

Step 12 : Stop.

Prog. No.....

Date.....

Page No.....

AIM :

program to search an element in the array using binary search.

SOURCE CODE :

```
#include <stdio.h>
#include <conio.h>
void main() {
    int list[25], max, first, last, middle,
                i, item, loc = -1;
    printf("Enter the limit:");
    scanf("%d", &max);
    printf("Enter array elements:");
    for(i=0; i<max; i++) {
        scanf("%d", &list[i]);
    }
    printf("Enter item to be searched:");
    scanf("%d", &item);
    first = 0;
    last = max-1;
    while (first <= last) {
    middle = (first + last)/2;
    if (item == list[middle]) {
        loc = middle;
        break;
    }
```

Output :

```
Enter the limit : 3
Enter array elements : 10 20 30
Enter the item to be searched : 20
The item is found at position 2
```

Prog. No.....

Date.....

Page No.....

```
if (item < list[middle])
    last = middle - 1;
else
    first = middle + 1;
}

if (loc != -1)
    printf("The item is found at
            position %d", loc+1);
else
    printf("not found");
}
```

Algorithm:

Step 1 : Start.
Step 2 : Read m,n to find order of matrix
Step 3 : Input element into matrix ar[10][10]
Step 4: Print entered matrix.
Step 5 : Repeat step 6-10 while (i<m)
Step 6 : Repeat step 7-10 while (j<n)
Step 7 : if (ar[i][j] != 0)
Step 8 : br[s][0] = i
Step 9 : br[s][1] = j
Step 10 : br[s][2] = ar[i][j]
Step 11 : Continue step 12,13 while (i<s)
Step 12 : Goto step 13 while (i<3)
Step 13 : Print br[i][j]
Step 14 : Stop.

Prog. No.
Date
Page No.

AIM:

Program to read a sparse matrix and display its triplet representation using array.

SOURCE CODE:

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

void main() {
    int i,j,m,n,ar[10][10], br[10][10],s=0;
    printf("Enter order of matrix:");
    scanf("%d%d", &m, &n);
    printf("Enter elements of matrix:");
    for(i=0; i<m; i++) {
        for(j=0; j<n; j++) {
            scanf("%d", &ar[i][j]);
        }
    }
    printf("The given matrix is:\n");
    for(i=0; i<m; i++) {
        for(j=0; j<n; j++) {
            printf("%d\t", ar[i][j]);
        }
        printf("\n");
    }
    for(i=0; i<m; i++) {
```

output :

```
Enter order of matrix : 2  2
Enter elements of matrix :
 1  2  3  4

The given matrix is:
 1  2
 3  4
The sparse matrix is :
 0   0   1
 0   1   2
 1   0   3
 1   0   4
```

Prog. No.

Date

Page No.

```
for (j=0; j<n; j++) {
        if (ar[i][j]!=0) {
            br[s][0] = i;
            br[s][1] =j;
            br[s][2] = ar[i][j];
            s++
        }
    }
}
printf("The sparse matrix is :\n");
for(i=0;i<s;i++) {
        for(j=0;j<3;j++) {
            printf("%d\t", br[i][j]);
        }
    }
}
```

Algorithm :

Step 1 : Start
Step 2 : include all the header files which are used in the program.
Step 3 : Declare all the user defined functions
Step 4 : Define a node structure.
Step 5 : Stop.

Void (create()).
Step 1 : Start
step 2 : create a new node with given value.
step 3 : check whether list is empty.
step 4 : If it is in NULL then print 'cannot be allocated, else
step 5 : add data for the first node & same in num.
step 6 : set stNode → data = n data ;
stNode → nextptr = NULL ;
ndBuffer = stNode.
Step 7 : Repeat step 8 to 12 (while i<=n)
step 8 : create next node with given value
step 9 : check whether list is empty
step 10 : If it is empty then print memory 'cannot be allocated.
Step 11 : else enter data for next node
Step 12 : nNode → data = n data
nNode → nextptr = NULL
ndBuffer → nd Buffer → nextptr
Step 13 : Stop.

Prog. No.....
Date.....
Page No.....

AIM :

Program to create a Singly linked list of n nodes and display it.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
  int data
  struct node *nextptr;
};
struct node *stNode;
static void createlist (int n);
static void displaylist ();
static void Createlist (int n);
{
struct node *nNode;
struct node *ndBuffer;
int nData;
int i;
stNode = (struct node*) malloc (sizeof
                    (struct node));
if (stNode == NULL)
{
```

void display(),

Step 1 : Start
Step 2 : Declare the variables
Step 3 : check whether the list is empty
Step 4 : print list is empty
Step 5 : else, repeat step 1 while (ndBuffer!=NULL)
Step 6 : print Data = %d, ndBuffer → data.
ndBuffer = ndBuffer → nextptr.
Step 7 : Stop.

Display list(),

Step 1 : Start
Step 2 : Input the numbers of n nodes listnum
Step 3 : create
Step 4 : print data entered in the list
Step 5 : Stop.

Prog. No.....
Page No.....
Date.....

```
    printf ("memory can not be allocated");
}
else
{
    printf("Input data for node 1:");
    scanf("%d", &nData);
    stNode → data = nData;
    stNode → nextptr = NULL;
    ndBuffer = stNode;
    for(i=2; i<=n; i++) {
    nNode = (struct node *) malloc (sizeof
                (struct node));
    if(nNode == NULL) {
        printf("memory can not be allocated");
        break;
    }
    else {
        printf("Input data for node %d:",i);
        scanf("%d", &nData);
        nNode → data = nData
        nNode → nextptr = NULL;
        ndBuffer → nextptr = nNode;
        ndBuffer = ndBuffer → nextptr;
    }
    }
}
```

Static void display list()

output:

```
Input the number of nodes : 3
Input data from node 1 : 10
Input data from node 2 : 20
Input data from node 3 : 30

Data entered in the list :
Data = 10
Data = 20
Data = 30
```

Prog. No.....

Date.....

Page No.....

```
{
  Struct node *ndBuffer();
  ndBuffer = stNode;
  if (ndBuffer == NULL) {
      printf (" list is empty ");
  }
  else {
      while (ndBuffer != NULL) {
          printf ("Data = %d \n", ndBuffer
                                  -> data);
          ndBuffer = ndBuffer -> nextptr;
      }
  }
}
void main() {
    int &num;
    printf (" Input the number
                of nodes : ");
    scanf ("%d", &num);
    Createlist (num);
    printf( "Data entered in the list");
    displaylist ();
}
```

Algorithm :

Step 1 : Start
Step 2 : Include all the header files which are used in the program
Step 3 : Declare all the user defined functions
Step 4 : Define a node structure.
Step 5 : Stop.

void (create ()),
Step 1 : Start
Step 2 : create a new node with given values.
Step 3 : check whether list is empty.
Step 4 : If it is NULL, print memory cannot be allocated.
Step 5 : else, add data to first node and save in num.
Step 6 : Set stnode → num = num
st node → nextptr = NULL
temp = stnode
Step 7 : Repeat step 8 to 12 while (i<=n)
Step 8 : create nextnode with given values.
Step 9 : Check whether list is empty
Step 10 : if it is empty then print. memory cannot be allocated.

Prog. No.....
Date.....
Page No.....

AIM :

program to delete a given node from a singly linked list.

SOURCE CODE :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct node
{
  int num;
  struct node *nextptr;
}*stnode;
void create (int n);
void delete (int pos);
void display ();
void create(int n);
{
  struct node *fnnode, *tmp;
  int num, i;
  stnode = (struct node *)malloc (sizeof
                (struct node));
  if (stnode == NULL) {
      printf ("memory cannot be
                allocated);
  }
```

Step 11 : else, enter the data for next node

Step 12 : Set fnnode → num = num
fnnode → nextptr = NULL
tmp → nextptr = fnnode
tmp = tmp → nextptr

Step 13 : Stop.

void (delete ())
Step 1 : Start
Step 2 : declare all the variables.
Step 3 : check if it is empty
Step 4 : else, define 2 node pointers to del and prenode and set prenode with stNode
Step 5 : Set prenode = to del
todel = todel → nextptr.
Step 6 : if todel == NULL then break.
Step 7 : if todel == stNode then set stNode = stNode → nextptr,
prenode → nextptr = todel → nextptr.
todel → nextptr = NULL
Step 8 : else print 'deletion cannot be possible from that position
Step 9 : Stop.

Prog. No.....
Date.....
Page No.....

```
else
{
  printf("Input data for node 1 : \n");
  scanf("%d", &num);
  stnode -> num = num;
  Stnode -> nextptr = NULL;
  tmp = Stnode;
  for(i=2; i<=n; i++) {
  fnnode = (struct node *) malloc (sizeof
                    (struct node));
  if (fnnode == NULL) {
      printf("memory can not be
                    allocated");
      break;
  }
  else {
      printf("Input data for node %d",
                    i);
      scanf("%d", &num);
      fnnode -> num = num;
      fnnode -> nextptr = NULL;
      tmp -> nextptr = fnnode;
      tmp = tmp -> nextptr;
  }
  }
}
void delete (int pos) {
```

void display()

Step 1 : Start

Step 2 : check whether the list is empty

Step 3 : if it is empty, print 'no data found in the list'.

Step 4 : else set temp = Stnode

Step 5 : Repeat Step 6 & 7 while (temp != NULL)

Step 6 : print the elements

Step 7 : Set tmp = tmp → nextptr

Step 8 : Stop

void main()

Step 1 : Start

Step 2 : declare all the variables.

Step 3 : Enter number of nodes, save in n

Step 4 : call create function with argument n.

Step 5 : print 'Data entered in the list.

Step 6 : call display function.

Step 7 : Enter the position of node to delete and save in pos.

Step 8 : check whether (pos <= 1 or pos >= n)

Step 9 : If it is true then print 'deletion cannot possible'

Prog. No.....

Date.....

Page No.....

```
int i;
struct node * todel, * prenode;
if (Stnode == NULL) {
    printf("there is no nodes
            in the list");
}
else {
    todel = Stnode;
    prenode = Stnode;
    for(i=2; i<=pos; i++) {
        prenode = todel;
        todel = todel → nextptr;
        if (todel == NULL)
            break;
    }
    if (todel != NULL) {
        if (todel == Stnode)
        Stnode = Stnode → nextptr;
        prenode → nextptr = todel →
                            nextptr;
        todel → nextptr = NULL;
        free (todel);
    }
    else {
        printf("deletion cannot be
        possible from that position");
    }
}
}
```

Step 10: check whether (pos >1 and pos <n)
Step 11: If it is true then print 'deletion completed successfully.
Step 12: call delete function
Step 13: print the new list
Step 14: call display function
Step 15: Stop.

Prog. No.....
Date.....
Page No.....

```
void display () {
    struct node *tmp;
    if (strode == NULL) {
        printf ("no data found in
                the list");
    }else {
        tmp = strode;
        while (tmp != NULL) {
            printf ("Data = %d \n",
                    tmp->num);
            tmp = tmp -> nextptr;
        }
    }
}
void main () {
    int n, num, pos;
    printf ("Input the number of
            nodes : ");
    scanf ("%d", &n);
    create(n);
    printf ("Data entered in the
            list are :");
    display ();
    printf ("Input the position of
            node to delete : ");
    scanf ("%d", &pos);
    if (pos <= 1 || pos >= n) {
        printf ("Deletion cannot be
```

Output:

```
Input the number of nodes : 4

Input data for node 1 : 10
Input data for node 2 : 20
Input data for node 3 : 30
Input data for node 4 : 40

Data entered in the list are:

Data = 10
Data = 20
Data = 30
Data = 40

Input the position of node to delete : 2
Deletion completed successfully

The new list are :

Data = 10
Data = 30
Data = 40
```

Prog. No.....

Date.....

Page No.....

```
            possible from that position");
    }
    if (pos > 1 && pos < n) {
        printf ("Deletion completed
                 successfully");
        delete (pos);
    }
    printf ("The new list are : \n");
    display ();

}
```

Algorithm:

Step1 : Start
Step2 : Struct node *lptr
Step3 : Read num to store data
Step4 : Struct node *head1, create Struct node Struct node **tail1, int data
Step5 : point Struct node newnode, temp
Step6 : is newnode = (Struct node *)malloc (sizeof (Struct node) newnode->data=dat
Step7 : newnode->rptr = NULL
Step8 : newnode->lptr = NULL
Step9 : Stop.

case 1 : head not equal to NULL.
Step1 : Start
Step2 : if (head != NULL) Repeat Step 3 to 10
Step 3 : point head != newnode
Step 4 : temp head1
Step 5 : Repeat step 5 to 9 while (temp->rptr = NULL).
Step 6 : print temp = temp->rptr
Step 7 : temp = newnode->lptr
Step 8 : newnode->rptr = NULL
Step 9 : *tail = newnode then temp->rptr

Prog. No.....
Date.....
Page No.....

AIM:

program to create a doubly linked list of integers and display in forward and backward directions.

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
Struct node
{
   int data
   Struct node *rptr, *lptr;
};
Struct node * create (Struct node *,
                      Struct node **, int);
void display (Struct node *) ;
void displays( Struct node *, Struct node*);
Struct node * create (Struct node *head1,
          Struct node **tail1, int dat)
{
   Struct node *newnode, *temp ;
   newnode = (Struct node *) malloc (sizeof
                (Struct node)) ;
   newnode -> data -> dat;
   new node -> rptr = newnode->lptr = NULL;
```

Step 10 : return head 1

Step 11 : Stop.

case 2 : Read head and tail

Step 1 : Start

Step 2 : while (head != NULL) then print head = head → rptr.

Step 3 : while (tail != ~~NULL~~ head) then print tail = tail → lptr.

Step 4 : IF (tail == head) then print tail = tail → lptr

Step 5 : Stop.

case 3 : display ()

Step 1 : Start

Step 2 : Accept the variables i, n value.

Step 3 : Struct node *head, tail

Step 4 : print 'Enter the limit' & read n

Step 5 : Repeat Step 1 to 9 while (i<n)

Step 6 : print 'Enter the number' and read value

Step 7 : head = create (head, &tail, value)

Step 8 : print the data in the forward direction ~~and~~ is printed below.

Step 9 : write head and print the data in backward direction is printed below

Step 10 : display tail, head

Step 11 : Stop.

Prog. No.....

Page No.....

Date.....

```
if (head 1 == NULL) {
    newnode → lptr = newnode → rptr = NULL;
    head 1 = new node ;
}
temp = head 1 ;
while (temp → rptr != NULL)
temp = temp → rptr ;
temp → rptr = new node ;
newnode → lptr = temp ;
new node → rptr = NULL;
* tail 1 = newnode ;
temp = temp → rptr ;
return head 1;
}
void display (struct node *head)
{
    while (head != NULL) {
        printf ("%d \n", head → data) ;
        head = head → rptr ;
    }
}
void displays (struct node *tail,
               struct node * head)
{
    while (tail 1 != head) {
        printf ("%d \n", tail → data);
        tail = tail → lptr ;
    }
```

Output :

```
Enter the limit : 3
Enter       number : 10
Enter the number : 20
Enter the number : 30
The data in the forward direction is:
10
20
30
The data in the backward direction is:
30
20
10
```

Prog. No.....

Date.....

Page No.....

```
    if (tail == head)
        printf ("%d \n", tail -> data);
}
void main () {
    int i,n, value;
    struct node *head, *tail;
    head = NULL;
    tail = NULL;
    printf ("Enter the limit:");
    scanf ("%d", &n);
    for(i=0; i<n ; i++) {
        printf ("Enter the numbers:");
        scanf ("%d", &value);
        head = create (head, &tail, value);
    }
    printf ("\n The data in the forward
        direction is : \n");
    display (head);
    printf ("\n The data in the
        backward direction is : \n");
    displays (tail, head);

}
```

Algorithm:

Step 1 : Start
Step 2 : Set top = -1
Step 3 : Accept the variable n to store size of stack
Step 4 : Reach Choice
Step 5 : if (choice == 1)
push ()
Step 6 : else if (choice == 2)
call pop()
Step 7 : elseif (choice == 3)
display()
Step 8 : elseif (choice == 4)
print 'exit point'
Step 9 : else print 'invalid choice'
Step 10 : Repeat Step 5-9 while (Choice != 4)
Step 11 : Stop.

push ()
Step 1 : Start
Step 2 : if (top < n-1)
read the number to be pushed and stored to val.
Step 3 : top++ and stack[top] = val

Prog. No.....
Date.....
Page No.....

AIM :

program to implement Stack operation using array.

SOURCE CODE :

```
#include <stdio.h>
#include <conio.h>
int stack[100], choice, top, n, val, i;
void push();
void pop();
void display();
void main() {
    top = -1;
    printf("Enter size of the stack:");
    scanf("%d", &n);
    printf("---- stack operation ----");
    printf("1. push 2. pop 3. display
                4. exit \n");
    do {
        printf("\n Enter your choice:");
        scanf("%d", &choice);
        switch (choice)
    {
        case 1: push();
            break;
        case 2: pop();
```

Step 5 : else, print 'Stack overflow'.
Step 6 : Stop.

pop()
Step 1 : Start
Step 2 : If (top > -1) return stack[top] top--
Step 3 : else print 'Stack underflow'.
Step 4 : Stop.

display()
Step 1 : Start
Step 2 : If (top >= 0)
Step 3 : Repeat Step 4 while (i >= 0)
Step 4 : print stack[i]
Step 5 : else print 'Stack is empty'
Step 6 : Stop.

Prog. No.....
Date.....
Page No.....

```
            break;
        case 3: display()
            break;
        case 4: Printf ("Exit point");
            break;
        default : printf ("Invalid choice");
    }
}
while (choice != 4);
    return 0;
}
    void push() {
        if (top < n-1) {
        printf ("\n Enter elements to
                be pushed: ");
        scanf ("%od", &val);
        top++;
        stack [top] = val;
        }
        else {
            print ("Stack overflow");
        }
    }
void pop() {
    if (top > -1) {
        printf ("The popped element
                is %od", stack [top]);
        top--;
    }
```

output:

```
Enter the size of stack : 3
- - - - Stack operation - - - -
1. push 2. pop 3. display 4. exit
Enter your choice : 1
Enter elements to be pushed : 10
Enter your choice : 1
Enter elements to be pushed : 2
Enter your choice : 2
the popped elements is 2
Enter your choice : 3
The elements of stack are:
10
Enter your choice : 4
exit point
```

Prog. No.....

Date.....

Page No.....

```
    else {
        printf("stack overflow");
    }
}
void display () {
    if (top >= 0) {
        printf("the elements of
            stack are: \n");
        for (i=top ; i>=0 ; i--)
            printf ("%d \n", stack[i]);
    }
    else {
        printf (" stack overflow");
    }

}.
```

Algorithm:

Step1 : Start
Step2 : Accept variable choice
step3 : if (choice == 1)
        push (val)
Step4 : elseif (choice == 2)
        pop()
Step5 : elseif (choice == 3)
        display()
Step6 : elseif (choice == 4)
        print 'exit point'.
Step7 : else print 'invalid choice'.
Step8 : Repeat steps from 3 to while (choice != 4)
Step9 : Stop.

push()
Step 1 : Start
Step 2 : if (top == NULL)
Step 3 : top = (struct node *) malloc (1 * sizeof (struct node))
Step 4 : top -> ptr = NULL
Step 5 : top -> info = a
Step 6 : else temp -> (struct node *) malloc (sizeof (struct node)).
Step 7 : temp -> info = a
Step 8 : temp -> ptr = NULL

Prog. No.....
Date.....
Page No.....

Aim:

Program for stack operation using linked list.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
struct node
{
   int info;
   struct node *ptr;
}*top, *top1, *temp;

int push (int a);
void pop ();
void display ();
int count = 0;
int main ()
{
    int choice, val;
    printf ("---- Stack operation ----");
    printf (" 1.push 2.pop 3.display 4.exit");
    do {
        printf ("Enter your choice: ");
        scanf ("%d", &choice);
        switch (choice) {
            case 1: printf ("Enter element
```

Step 9 : top = temp
Step 10 : count ++
Step 11 : Stop.

pop()
Step 1 : Start
Step 2 : Set tol = top
Step 3 : if (top != NULL) then print 'Stack underflow'
Step 4 : else top = top -> ptr.
Step 5 : return top -> info. and free (top)
Step 6 : top = top1
Step 7 : count --
Step 8 : Stop.

display().
Step 1 : Start
Step 2 : Set top1 = top
Step 3 : if (top1 == NULL)
Step 4 : else
Step 5 : Repeat step 6 and 7 while (top1 != NULL)
Step 6 : return top1 -> info
Step 7 : top1 = top1 -> ptr
Step 8 : Stop.

Prog. No.....
Date.....
Page No.....

```
            to be pushed : ");
        scanf("%d", &val);
        push(val);
        break;
    case 2: pop();
        break;
    case 3: display();
        break;
    case 4: printf("exit point");
        break;
    default : printf("Invalid choice");
    }
}
while (choice != 4);
    return 0;
}
int push(int a) {
    if (top == NULL) {
        top = (struct node *) malloc
            (sizeof(struct node));
        top -> ptr = NULL;
        top -> info = a;
    }
    else {
        temp = (struct node *) malloc
            (size of (struct node));
        temp -> info = a;
        temp -> ptr = top;
```

output :

```
---- Stack operation ----
1. push 2. pop 3. display 4. exit
Enter your choice: 1
Enter elements to be pushed : 10
Enter your choice : 1
Enter elements to be pushed : 30
Enter your choice : 2
the popped element is 30
Enter your choice : 3
the elements are :
10
Enter your choice : 4
exit point
```

Prog. No.....

Date.....

Page No.....

```
        top = temp;
    }
    count ++;
    return 0;
}
void pop () {
    top1 = top
    if (top1 == NULL) {
        printf("Stack overflow");
    }else {
        top1 = top1 -> ptr;
        printf ("the popped element
                is %d \n", top -> info);
        free (top);
        top = top1;
        count --;
    }
}
void display () {
    top1 = top;
    if (top1 == NULL) {
        printf ("Stack is empty");
    }
    else {
        printf ("the elements are: \n");
        while (top1 != NULL) {
            printf("%d\n", top1->info);
            top1 = top1 -> ptr;
        }}}.
```

Algorithm:

Step1 : Start
Step2 : Initialize the stack
Step3 : Scan the given expression from left to right
Step4 : a) if the scanned character is an operand, push it into the stack.
b) if the scanned character is an operation pop 2 operands from Stack
Step5 : Repeat Step 3 till all the characters are scanned
Step6 : When the expression is ended, the numbers in the stack is the final result.
Step7 : Stop.

Prog. No.....
Date.....
Page No.....

AIM:

Program for evaluation of postfix expression.

SOURCE CODE:

```
# include <stdio.h>
# include <conio.h>
int stack[20];
int top=-1;
void push(int x) {
    stack[++top] = x;
}
int pop() {
    return stack[top--];
}
void main() {
    char exp[20];
    char *e;
    int n1, n2, n3, num;
    printf("Enter the postfix
                expression:\n");
    scanf("%s", exp);
    e=exp;
    while(*e != '\0') {
        if(isdigit(*e)) {
            num = *e - 48;
```

Prog. No.

Date

Page No.

```
        push(num);
    }
    else {
        n1 = pop();
        n2 = pop();
        switch(*e) {
            case '+':
            {
                n3 = n1 + n2;
                break;
            }
            case '-': {
                n3 = n2 - n1;
                break;
            }case '*':
            {
                n3 = n1 * n2;
                break;
            }
            case '/': {
                n3 = n2 / n1;
                break;
            }
            push(n3);
        }
        e++;
    }
```

output :

```
Enter the postfix expression:
5 23 * +4 -
The result of the postfix expression:
523*+4- = 7
```

Prog. No.....

Date.....

Page No.....

```
printf("The result of the postfix
        expression %d\n = %d \n",
                exp, pop());
}.
```

Algorithm:

Step1 : Start
Step2 : Input variable choice
Step3 : If (choice == 1)
        insert()
Step4 : elseif (choice == 2)
        del()
Step5 : elseif (choice == 3)
        display()
Step6 : elseif (choice == 4)
        exit point
Step7 : print 'invalid choice'
Step8 : Stop.

insert()

Step1 : Start
Step2 : If (rear == n) print 'overflow'
Step3 : else
Step4 : if (front == -1) then set front = 0
Step5 : accept variable val to
        input element.
Step6 : queue[rear] = val
Step7 : increment rear
Step8 : Stop.

Prog. No.....
Page No.....
Date.....

AIM:

program to implement queue using array.

SOURCE CODE:

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

int array[100], n, front = -1, rear = -1, val,
                                choice, i;

void insert ();
void delete ();
void display();
int main () {
    printf ("Enter size of queue: ");
    scanf ("%d", &n);
    printf ("\n 1. insert 2. delete
            3. display 4. exit \n");

    do {
        printf ("\n Enter your choice:");
        scanf ("%d", &choice);
        switch (choice) {
            case 1 : insert();
                     break;
            case 2 : delete();
                     break;
```

del()

Step1 : Start
Step 2 : if (front = -1 && front rear)
Step 3 : return queue [front]
Step 4 : front ++
Step 5 : else print 'queue underflow'.
Step 6 : Stop.

display()

Step 1 : Start
Step 2 : if (front == rear)
then print 'queue empty'.
Step 3 : else then print element of queue.
Step 4 : Stop.

Prog. No.....

Date.....

Page No.....

```
case 3 : display ();
         break
case 4 : printf ("exit point");
         break
default : printf ("Invalid choice");
}
}
while (choice != 4);
    return 0;
}
void insert () {
    if (rear == n-1)
        print ("overflow");
    else {
        if (front == -1)
            front = 0;
        printf ("Enter element to be inserted:");
        scanf ("%d", &val);
        rear = rear + 1
        array [rear] = val;
    }
}
void delete () {
    if (front == -1) {
        print ("overunderflow");
        return;
}
```

output :

```
Enter size of queue : 3
1. insert 2. delete 3. display 4. exit
Enter your choice : 1
Enter elements to be inserted : 10
Enter your choice : 1
Enter elements to be inserted : 20
Enter your choice : 2
The deleted element is 20
Enter your choice : 3
The elements of queue are :
20
Enter your choice : 4
exit point
```

Prog. No.....

Date.....

Page No.....

```
    else {
        printf ("the deleted element is
                %d ", array[front]);
        front = front + 1;
    }
}
void display () {
    if (front == -1)
        printf ("queue is empty");
    else {
        printf ("the elements of
                queue are: \n");
        for (i = front; i <= rear; i++)
            printf ("%d \n", array[i]);
    }
}
```

Algorithm:

Step 1: Start
Step 2: include all the header files used in the program and declare all functions
Step 3: define a node structure with 2 members data and next.
Step 4: define 2 node pointers 'front' and 'rear' and set both to NULL.
Step 5: Implement the main method by displaying of list of operations and make suitable function calls in the main method.
Step 6: Stop.

Insert ()

Step 1 : Start
Step 2: Create a newnode with given value and set newnode → next to NULL
Step 3: check whether queue is empty (rear == NULL)
Step 4: if it is empty then, set front = new node and rear = new node.
Step 5: If it is not empty then, Set rear → next = newnode and rear = newnode
Step 6: Stop.

Prog. No.....
Date.....
Page No.....

AIM:

program to implement queue using linked list.

SOURCE CODE:

```
#include <stdio.h>
#include <conio.h>
#include<stdlib.h>
struct node
{
  int info;
  struct node * next;
};
typedef struct node *link;
link q;
link getnode()
{
  link q;
  q = (link) malloc (sizeof (struct node));
  return q;
}
void insert (link s, int y) {
      link p;
      p = getnode();
      p→ info = y;
      p→ next = NULL;
```

delete ()

Step 1 : Start

Step 2: Check whether queue is empty

Step 3: If it is empty, then display queue is empty.

Step 4: If it is not empty, define a node pointer 'temp' and set it to 'front'.

Step 5: Then set front = front → next and delete temp

Step 6 : Stop.

display ()

Step1 : Start

Step2: Check whether queue is empty

Step3: If it is not empty, define a a node pointer temp and initialise with front.

Step4: display temp → data and move it to the next node.

Step5 : finally display temp → data → NULL

Step 6 : Stop.

Prog. No.....

Date.....

Page No.....

```
if (s->next = NULL)
    s -> next = p;
else
    q -> next = p;
    q = p;
}
void display (link s) {
    link p;
    p = get code ();
    p = s -> next;
    while (p != NULL) {
        printf ("%d \t", p -> info);
        p = p -> next;
    }
}
void freenode (link p) {
    free (p);
}
int delete (link s) {
    link p;
    int y;
    p = get code ();
    p = s -> next;
    if (p == q)
        q = s
    else
        s -> next = p -> next;
        y = p -> info;
```

Prog. No.....

Page No.....

Date.....

```
    freenode (p);
    return (y);
}
void main () {
    link s;
    int x, y, w;
    s = get node();
    q = s;
    printf ("Enter Size of queue:");
    scanf ("%d", &w);
    printf (" 1. insert 2. delete
            3. display 4. exit (n");
    do {
        printf ("\n Enter your choice:");
        scanf ("%d", &x);
        switch (x) {
        case 1:
            printf ("Enter number
                    to insert:");
            scanf ("%d", &y);
            insert (s, y);
            break;
        case 2: If (q != s)
            {
                y = delete(s);
                printf (" deleted number
                        is %d", y);
            } else { printf ("underflow");
                break;
```

output:

```
Enter the size of queue : 3
1. insert 2. delete 3. display 4. exit

Enter your choice : 1
Enter the number to insert : 10
Enter your choice : 1
Enter the number to insert : 20
Enter your choice : 2
Deleted number is 10
Enter your choice : 3
20
Enter your choice : 4
Exit point
```

Prog. No.....

Page No.....

Date.....

```
        case 3:
            display (5);
            break;
        case 4:
            printf("exit point");
            break;
        }
    }
    while (x<4);
    getch();
}
```

Algorithm:

Step1 : Start

Step2 : Allocate memory for tree

Step3: Set data part to the value and set the left and right pointers of tree.

Step 4: if the item to be inserted, will be the first element of the tree

Step 5 : else check if the item is less than the root element of tree.

Step 6 : If this false, then perform this operation recursively with the right Sub-tree of the root

Step 7 : Stop.

Prog. No.....

Date.....

Page No.....

AIM:

Program to search an element in a binary search tree.

SOURCE CODE:

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

void main() {
    int i, first, last, middle, n, search,
                                array[100];
    printf("Enter number of elements:");
    scanf("%d", &n);
    printf("In Enter %d integers:\n", n);
    for(i=0; i<n; i++)
        scanf("%d", array[i]);
        printf("Enter value to find:");
        scanf("%d", &search);
        first=0
        last=n-1
        middle = (first+last)/2;
        while (first<=last) {
            if (array[middle]<search)
                first = middle+1;
            else if (array[middle]==search)
                {
```

Output:

```
Enter number of elements : 3
Enter 3 integers :
10
30
60
Enter value to find :
60
60 found at location 3
```

Prog. No.....

Date.....

Page No.....

```
            printf("%d found at location %d
                    \n", search, middle+1);
            break;
        }
        else
            last = middle-1;
            middle = (first + last)/2;
        }
        if (first > last)
            printf("not found!");

    }.
```

Algorithm:

Step1 : Start
Step2: Accept a value in n as no. of elements
Step3: Accept n elements into array a
Step4: Repeat step 5–7 (n-1) times.
Step5: Repeat step 6 until 2nd last element of the list
Step6: Starting from 1st position, compare 2 adjacent elements of the list.
Step7: Revise the list by excluding last element in the current list
Step8: print sorted array a
Step9 : Stop.

Output:

```
Enter a limit :4
Enter the elements to be sorted:
10
5
3
80
The sorted list is :
3
5
10
80
```

Prog. No.....
Date.....
Page No.....

Aim :

program to implement Exchange sort

SOURCE CODE :

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

void main () {
    int a[100], i, n, j, t;
    printf("enter a limit :\n");
    scanf("%d", &n);
    printf("enter the elements to
                    be sorted :\n");
    for(i=0; i<n; i++)
        scanf("%d", &a[i]);
        for(i=0; i<n-1; j++)
            for (j=i+1; j<n; j++)
                if(a[i] > a[j]) {
                    t = a[i];
                    a[i] = a[j];
                    a[j] = t;
                }
    printf("The sorted list is:\n");
    for(i=0; i<n; i++)
        printf("%d\n", a[i]);
}
```

Algorithm:

Step 1 : Start

Step 2 : Accept 2 value in N as numbers of element of array.

Step 3 : Repeat Step 5 - 9 (N-1) times.

Step 4 : Assume 1st element in the list as the smallest and store it in min and its position in pos.

Step 5 : Repeat step 7 until last Element in the list.

Step 6 : Compare next element in the list if it is found smaller, store it in min and position in pos.

Step 7 : if it is not same, swap the 1st element with the element at position pos.

Step 8 : Revise list by including 1st element in the current list

Step 9 : print sorted array

Step 10 : Stop.

Prog. No.

Date

Page No.

AIM:

program to implement selection sort

SOURCE CODE:

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

void main () {
    int ar[25], n, i, j, min, pos;
    printf ("Enter the limit : \n");
    scanf ("%d", &n);
    printf ("Enter elements : \n");
    for (i=0; i<n-1; i++) {
        min = ar[i];
        pos = i;
        for (j=i+1; j<n; j++) {
            if (ar[j] < min) {
                min = ar[j];
                pos = j;
            }
        }
        if (pos != i) {
            ar[pos] = ar[i];
            ar[i] = min;
        }
    }
```

Output:

```
Enter the limit :
3
Enter elements :
2
50
3
Sorted array is:
2   3   50
```

Prog. No.....

Date.....

Page No.....

```
    printf ("Sorted array is : \n");
    for(i=0; i<n ; i++)
    {
      printf ("%d \t", ar[i]);
    }

}.
```

Algorithm :

Step1 : Start
Step 2 : Read n to store total number of elements
Step 3 : input elements in to array
Step 4 : Repeat 5-10 while (i<n-1)
Step 5 : Set j=i
Step 6 : Repeat 7-10 while (j>0 && a[j-1] > a[j])
Step 7 : temp = a[j]
Step 8 : a[j] = a[j-1]
Step 9 : a[j-1] = temp
Step 10 : j--
Step 11 : print a[i]
Step 12 : Stop.

Prog. No.....
Date.....
Page No.....

AIM :

program to implement insertion sort

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
void main () {
    int a[100], i, n, j, temp;
    printf ("Enter number of elements:");
    scanf ("%od ", &n);
    printf ("Enter the elements:\n");
    for (i=0 ; i<n ; i++) {
        scanf ("%od", &a[i]);
    }
    for (i=1; i<=n-1; i++) {
        j=i
        while (j>0 && a[j-1]>a[j])
        {
            temp = a[j];
            a[j] = a[j-1];
            a[j-1] = temp;
            j--;
        }
    }
    printf ("The sorted elements
                are :\n");
```

Output:

```
Enter number of elements : 3
Enter the elements :
20
6
2
The sorted elements are :
2
6
20
```

Prog. No.....

Date.....

Page No.....

```
    for(i=0; i<=n-1; i++)
    {
        printf("%d\n", a[i]);
    }
}
```