INDEX

Experiment - 1

Q.Write a menu-based program to perform array operations: deletion of an element from the specified position, inserting an element at the specified position, printing the array elements.

Theory-

Array : An array is defined as finite ordered collection of homogenous data, stored in contiguous memory locations.

- finite means data range must be defined.
- ordered means data must be stored in continuous memory addresses.
- homogenous means data must be of similar data type.

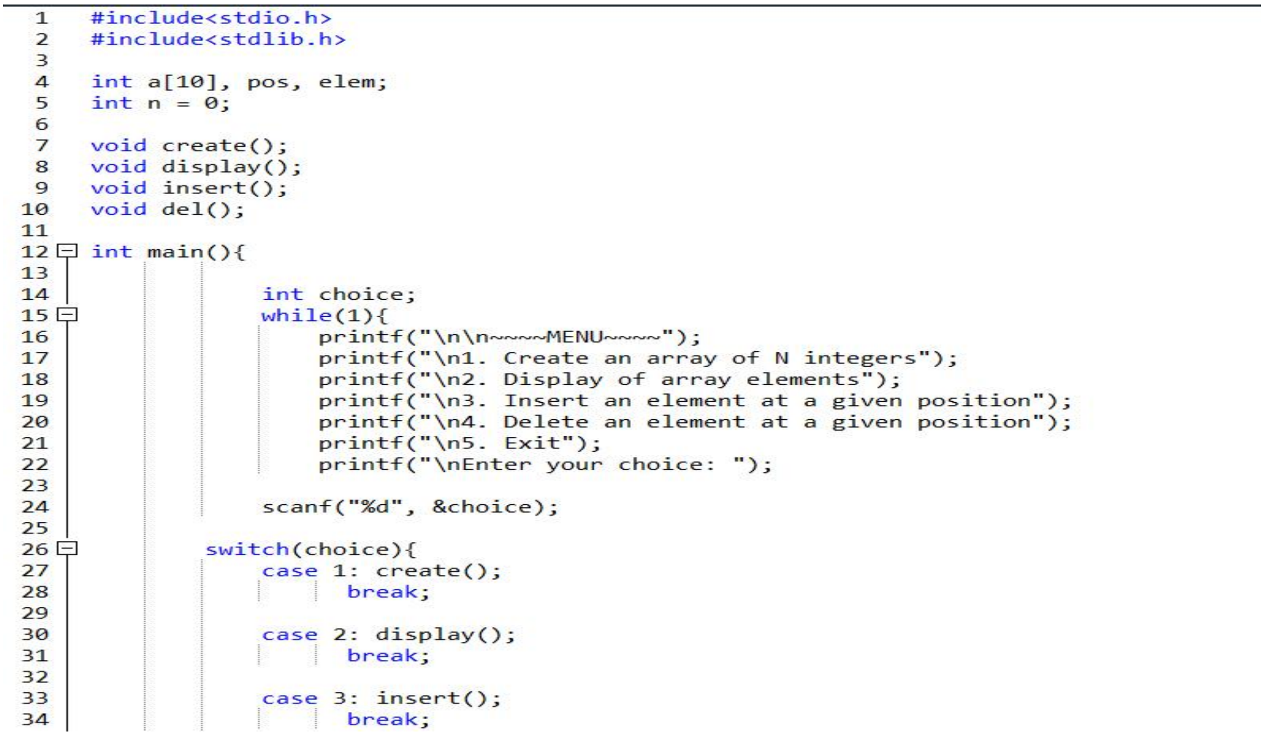
Advantages of Array :

- Less code to the access the data.
- By using the for loop, we can retrieve the elements of an array easily.
- To sort the elements of the array, we need a few lines of code only.
- We can access any element randomly using the array.

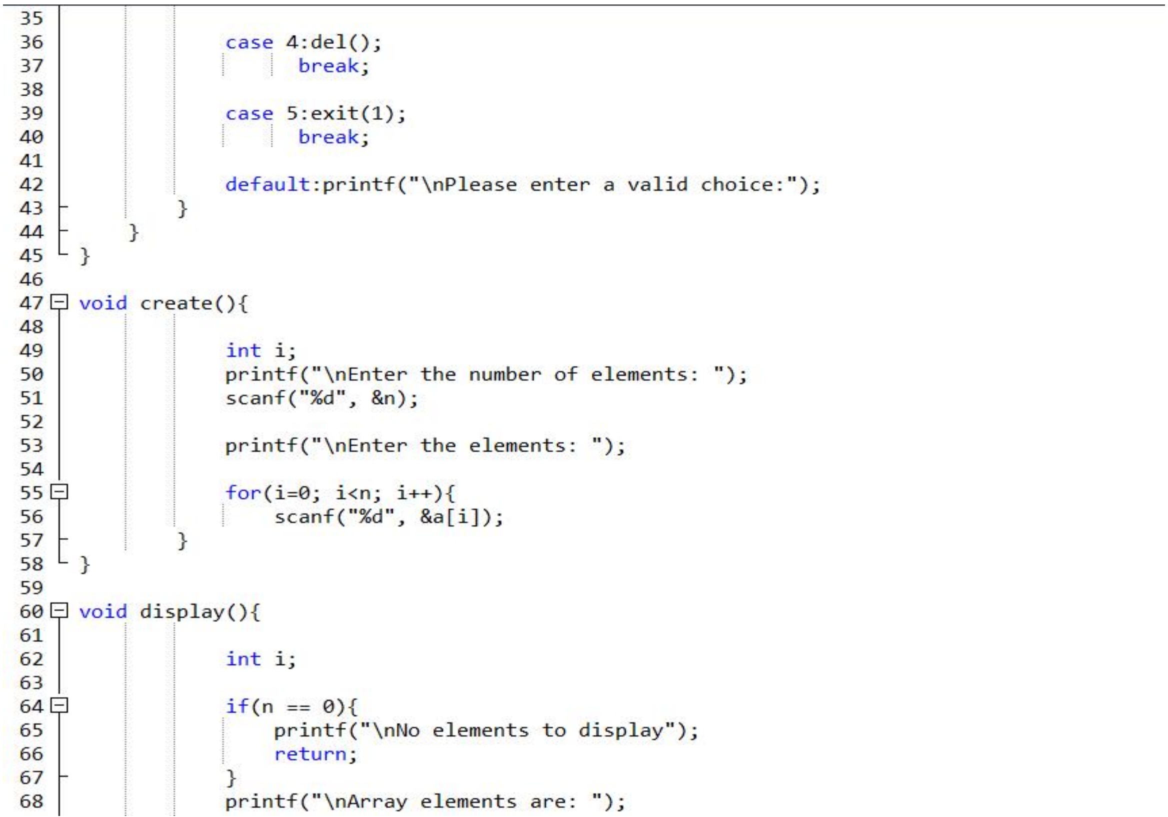
Disadvantages of Array :

- We must know in advance that how many elements are to be stored in array.
- Array is static structure. It means that array is of fixed size. The memory which is allocated to array can not be increased or reduced.
- Since array is of fixed size, if we allocate more memory than requirement then the memory space will be wasted. And if we allocate less memory than requirement, then it will create problem.
- The elements of array are stored in consecutive memory locations. So insertions and deletions are very difficult and time consuming.

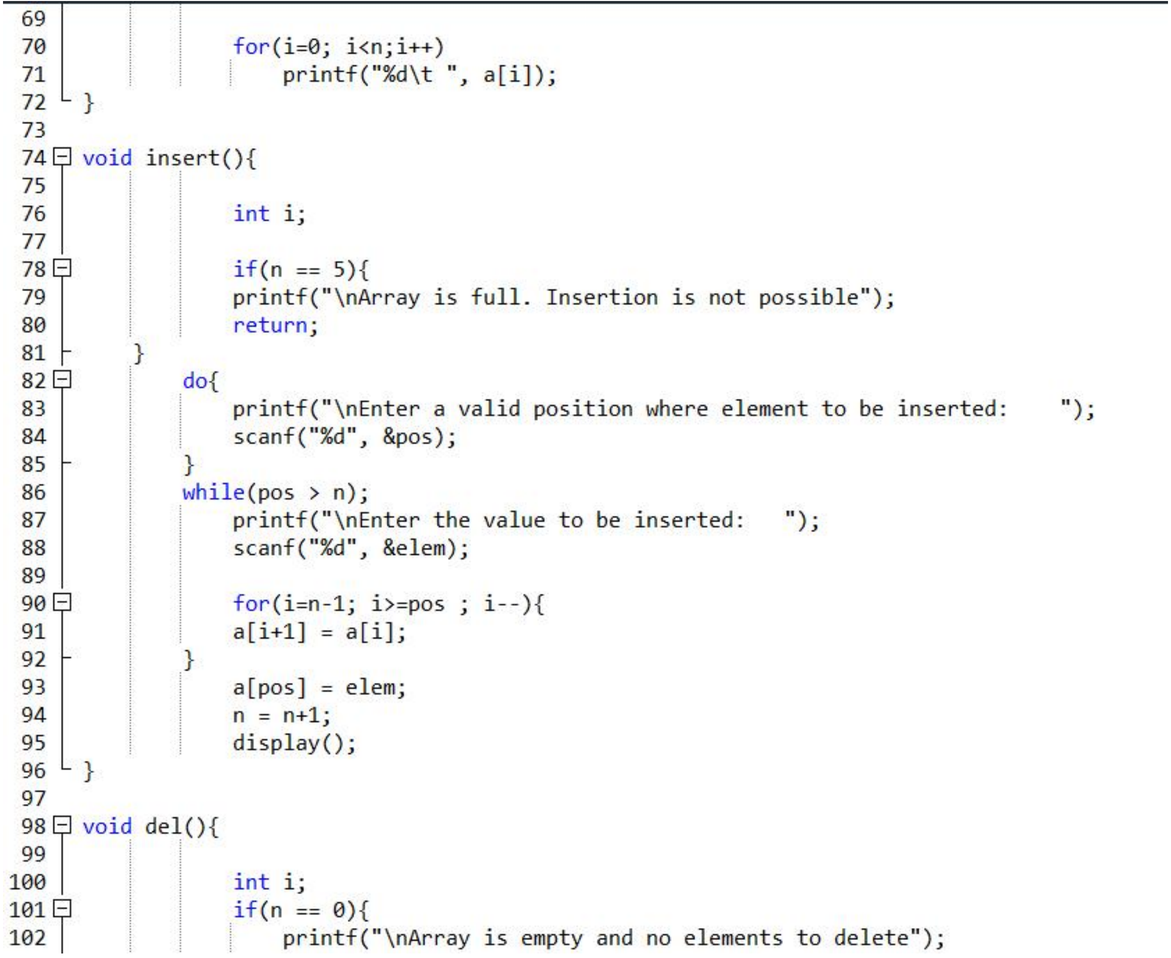
Source Code-



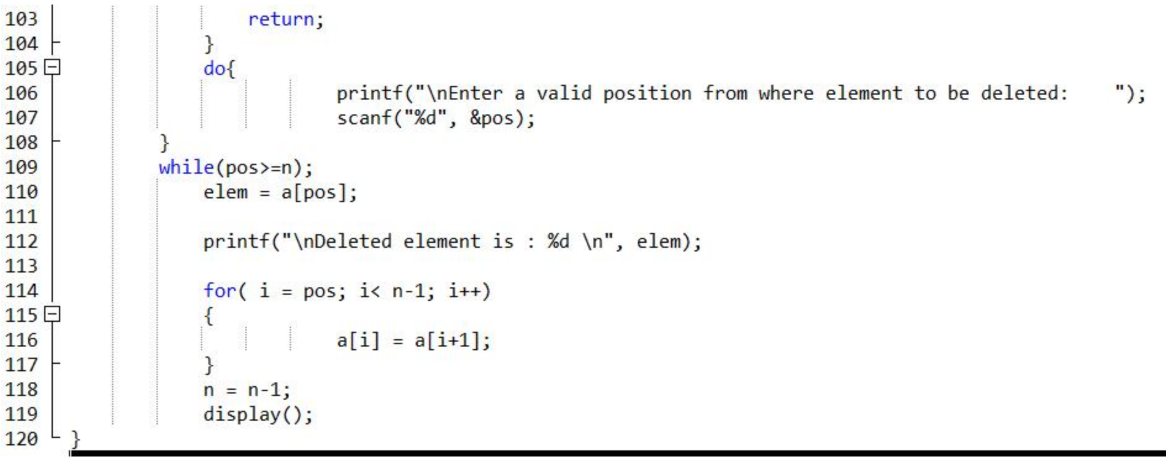
```c
#include<stdio.h>
#include<stdlib.h>

int a[10], pos, elem;
int n = 0;

void create();
void display();
void insert();
void del();

int main(){

            int choice;
            while(1){
                printf("\n\n~~~~MENU~~~~");
                printf("\n1. Create an array of N integers");
                printf("\n2. Display of array elements");
                printf("\n3. Insert an element at a given position");
                printf("\n4. Delete an element at a given position");
                printf("\n5. Exit");
                printf("\nEnter your choice: ");

            scanf("%d", &choice);

        switch(choice){
            case 1: create();
                  break;

            case 2: display();
                  break;

            case 3: insert();
                  break;
```



```

            case 4:del();
                break;

            case 5:exit(1);
                break;

            default:printf("\nPlease enter a valid choice:");
        }
    }
}

void create(){

            int i;
            printf("\nEnter the number of elements: ");
            scanf("%d", &n);

            printf("\nEnter the elements: ");

            for(i=0; i<n; i++){
                scanf("%d", &a[i]);
        }
}

void display(){

            int i;

            if(n == 0){
                printf("\nNo elements to display");
                return;
            }
            printf("\nArray elements are: ");
```

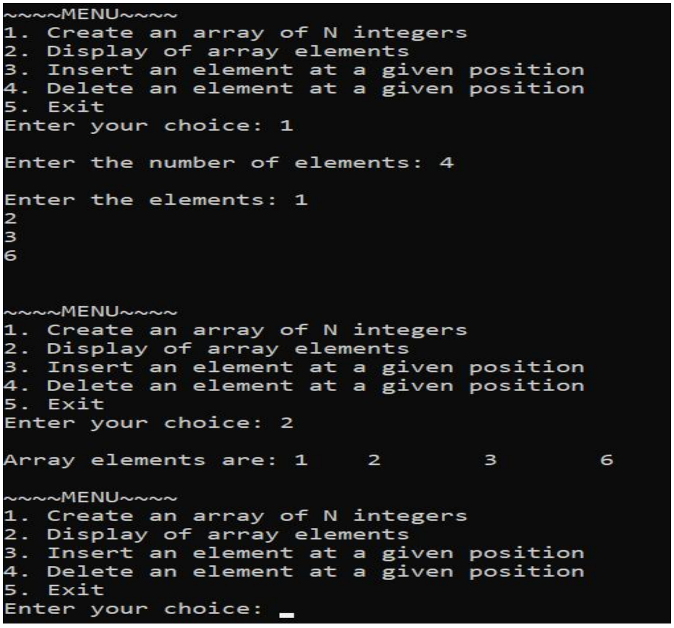


```

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

void insert(){

            int i;

            if(n == 5){
            printf("\nArray is full. Insertion is not possible");
            return;
    }
        do{
            printf("\nEnter a valid position where element to be inserted:    ");
            scanf("%d", &pos);
        }
        while(pos > n);
            printf("\nEnter the value to be inserted:   ");
            scanf("%d", &elem);

            for(i=n-1; i>=pos ; i--){
            a[i+1] = a[i];
        }
            a[pos] = elem;
            n = n+1;
            display();
}

void del(){

            int i;
            if(n == 0){
                printf("\nArray is empty and no elements to delete");
```



```
                return;
            }
            do{
                        printf("\nEnter a valid position from where element to be deleted:    ");
                        scanf("%d", &pos);
        }
        while(pos>=n);
            elem = a[pos];

            printf("\nDeleted element is : %d \n", elem);

            for( i = pos; i< n-1; i++)
            {
                        a[i] = a[i+1];
            }
            n = n-1;
            display();
}
```

Output-



```
~~~~MENU~~~~
1. Create an array of N integers
2. Display of array elements
3. Insert an element at a given position
4. Delete an element at a given position
5. Exit
Enter your choice: 1

Enter the number of elements: 4

Enter the elements: 1
2
3
6


~~~~MENU~~~~
1. Create an array of N integers
2. Display of array elements
3. Insert an element at a given position
4. Delete an element at a given position
5. Exit
Enter your choice: 2

Array elements are: 1     2       3       6

~~~~MENU~~~~
1. Create an array of N integers
2. Display of array elements
3. Insert an element at a given position
4. Delete an element at a given position
5. Exit
Enter your choice: _
```

Experiment - 2

Q.Write a program to implement 2D Array and also write a program to search an element in a 2-dimensional array.

Theory-

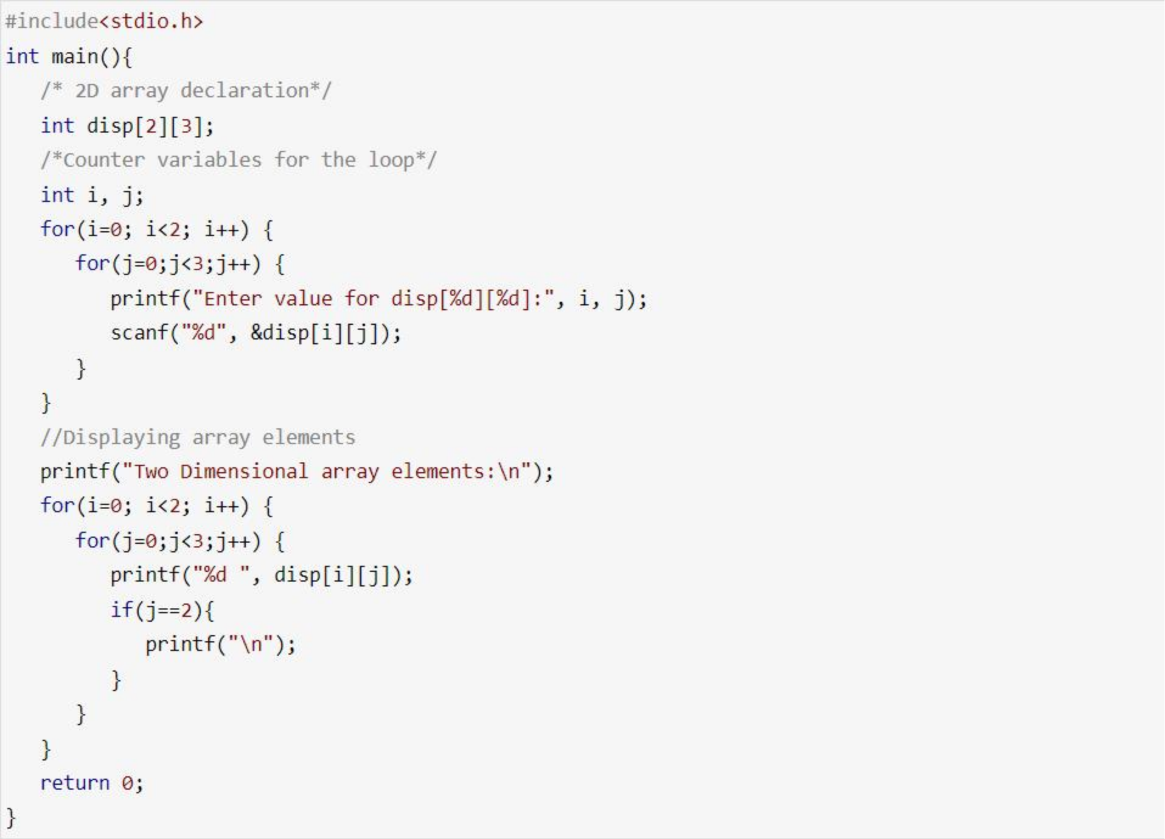
An array of arrays is known as 2D array. The two dimensional (2D) array is also known as matrix. A matrix can be represented as a table of rows and columns.

Algorithm to search any Element in 2D Array-

1. Create an array and initialize it with the elements.
2. Input the search element.
3. For loop from i=0 to i<rows:
4. For loop from j=0 to j<columns:
5. Check if search_element==array[i][j]:
6. If yes then output the row and column index position (i.e. (i, j)).
7. If not a single match was found then output ‘Not Found’.

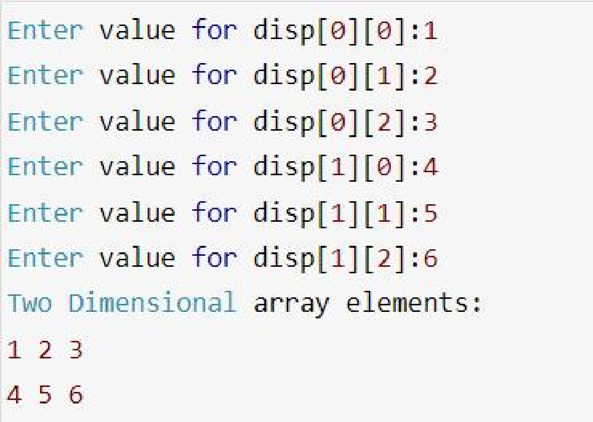
Source Code-

//Implementation of 2D Array



```
#include<stdio.h>
int main(){
   /* 2D array declaration*/
   int disp[2][3];
   /*Counter variables for the loop*/
   int i, j;
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("Enter value for disp[%d][%d]:", i, j);
         scanf("%d", &disp[i][j]);
      }
   }
   //Displaying array elements
   printf("Two Dimensional array elements:\n");
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("%d ", disp[i][j]);
         if(j==2){
            printf("\n");
         }
      }
   }
   return 0;
}
```

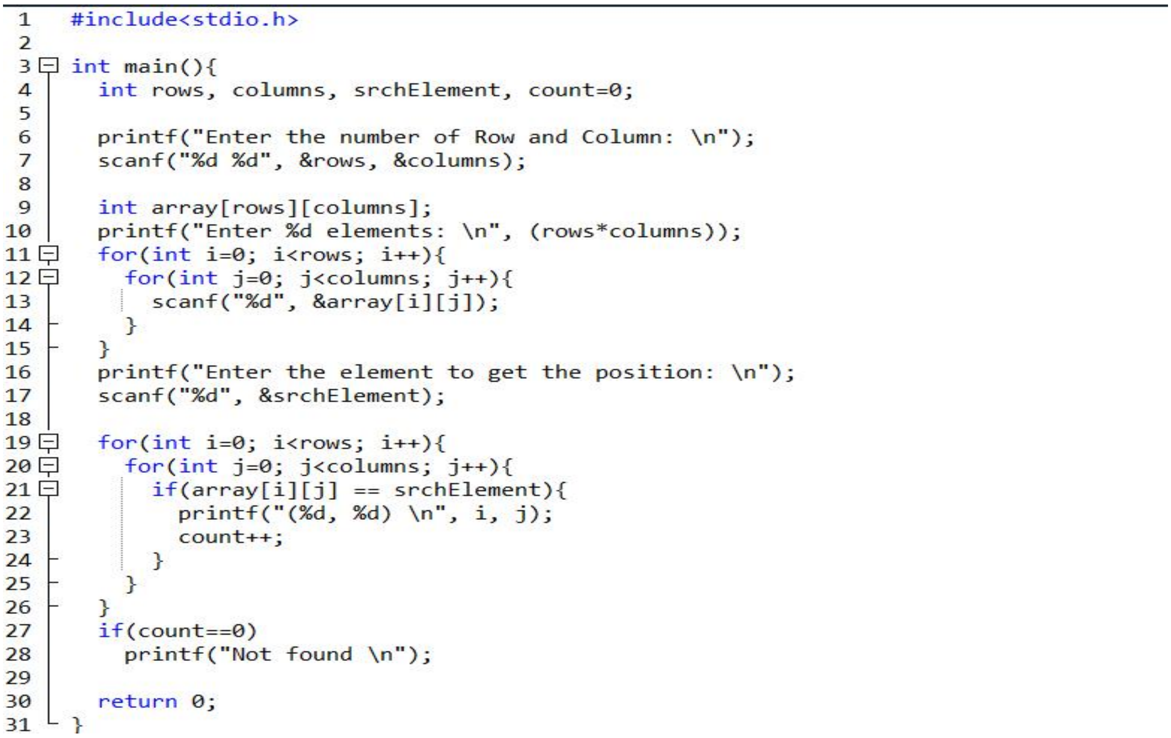
Output-



```
Enter value for disp[0][0]:1
Enter value for disp[0][1]:2
Enter value for disp[0][2]:3
Enter value for disp[1][0]:4
Enter value for disp[1][1]:5
Enter value for disp[1][2]:6
Two Dimensional array elements:
1 2 3
4 5 6
```

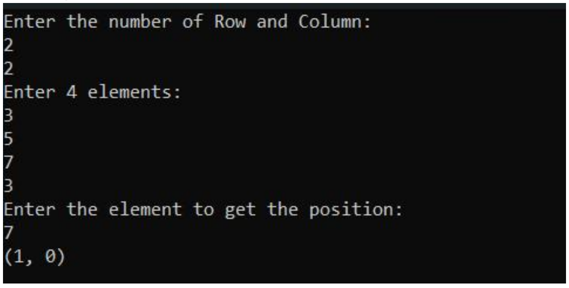
Source Code-

//Program to search an element in 2D array



```c
#include<stdio.h>

int main(){
  int rows, columns, srchElement, count=0;

  printf("Enter the number of Row and Column: \n");
  scanf("%d %d", &rows, &columns);

  int array[rows][columns];
  printf("Enter %d elements: \n", (rows*columns));
  for(int i=0; i<rows; i++){
    for(int j=0; j<columns; j++){
      scanf("%d", &array[i][j]);
    }
  }
  printf("Enter the element to get the position: \n");
  scanf("%d", &srchElement);

  for(int i=0; i<rows; i++){
    for(int j=0; j<columns; j++){
      if(array[i][j] == srchElement){
        printf("(%d, %d) \n", i, j);
        count++;
      }
    }
  }
  if(count==0)
    printf("Not found \n");

  return 0;
}
```

Output-



```
Enter the number of Row and Column:
2
2
Enter 4 elements:
3
5
7
3
Enter the element to get the position:
7
(1, 0)
```

Experiment - 3

Q.Write a program to perform following operations in matrix:Addition, Subtraction, Multiplication, Transpose.

Theory-

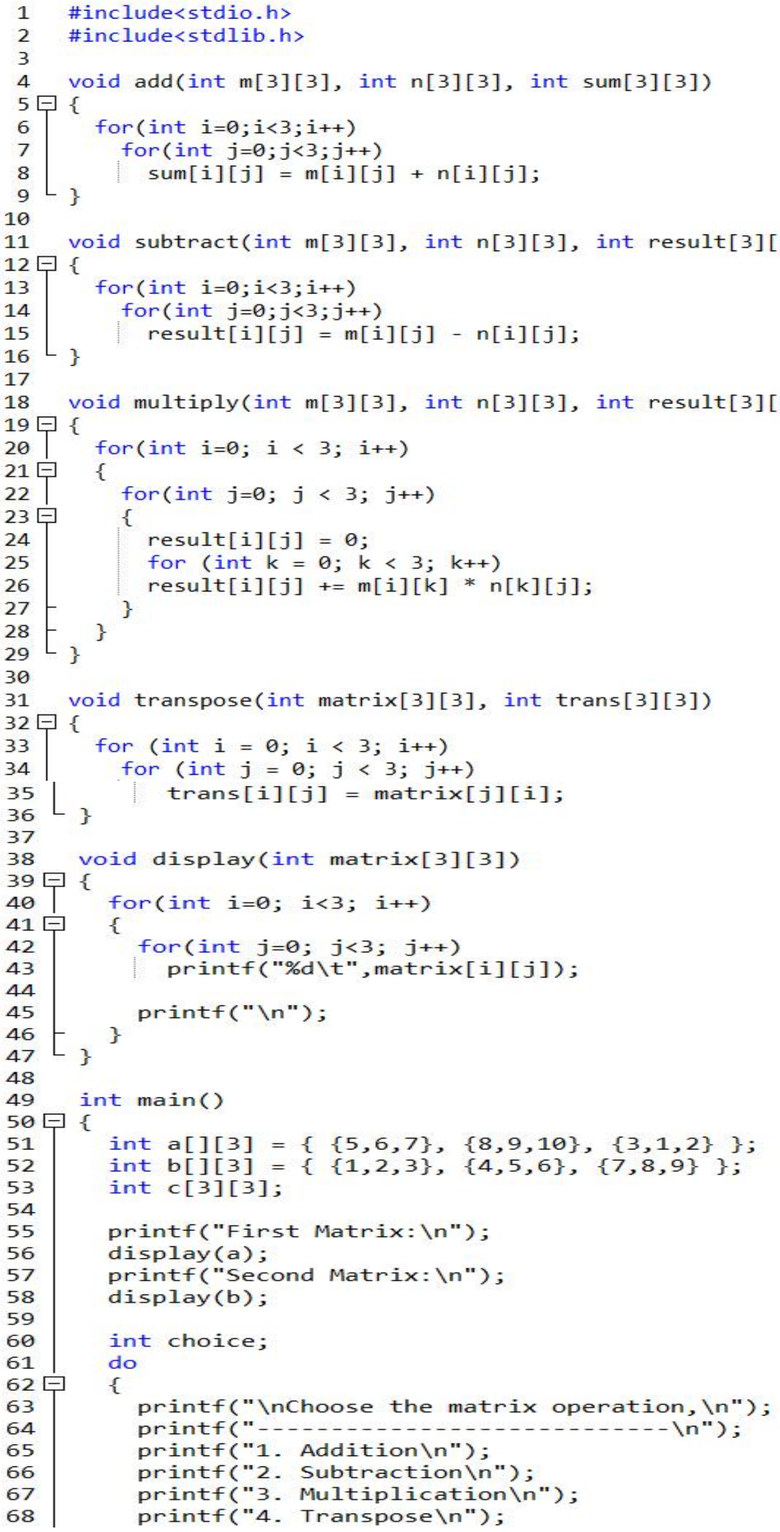
Matrix addition is the operation of adding two matrices by adding the corresponding entries together. The matrix can be added only when the number of rows and columns of the first matrix is equal to the number of rows and columns of the second matrix. In this program, we will take two square matrices of size 3×3. Matrix addition is very simple, just add the elements located at the same position with respect to the row and column.

The matrix subtraction is very similar to the matrix addition operation. Instead of the addition operator use the subtraction operator and remaining, things will remain the same.

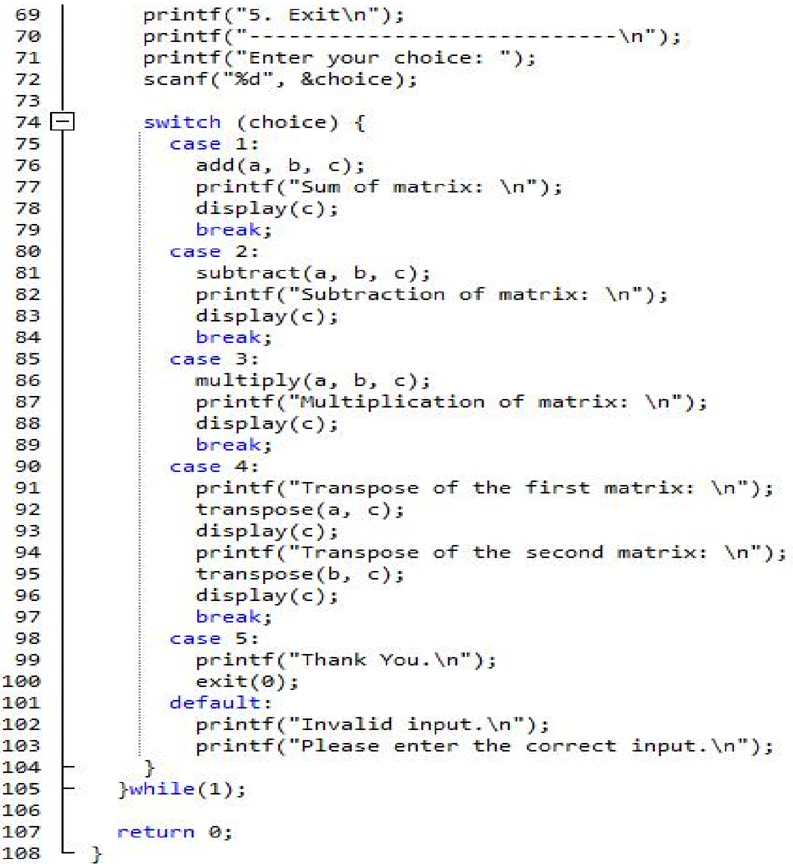
We can multiply two matrices if, and only if, the number of columns in the first matrix equals the number of rows in the second matrix. Otherwise, the product of two matrices is undefined.

If A=[aij] be a matrix of order m x n, then the matrix obtained by interchanging the rows and columns of A is known as Transpose of matrix A. Transpose of matrix A is represented by AT.

Source Code-

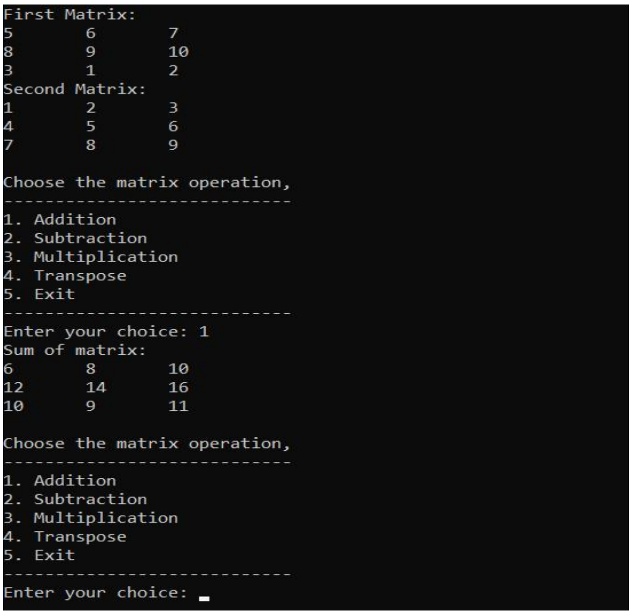


```c
#include<stdio.h>
#include<stdlib.h>

void add(int m[3][3], int n[3][3], int sum[3][3])
{
  for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      sum[i][j] = m[i][j] + n[i][j];
}

void subtract(int m[3][3], int n[3][3], int result[3][3])
{
  for(int i=0;i<3;i++)
    for(int j=0;j<3;j++)
      result[i][j] = m[i][j] - n[i][j];
}

void multiply(int m[3][3], int n[3][3], int result[3][3])
{
  for(int i=0; i < 3; i++)
  {
    for(int j=0; j < 3; j++)
    {
      result[i][j] = 0;
      for (int k = 0; k < 3; k++)
      result[i][j] += m[i][k] * n[k][j];
    }
  }
}

void transpose(int matrix[3][3], int trans[3][3])
{
  for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
      trans[i][j] = matrix[j][i];
}

void display(int matrix[3][3])
{
  for(int i=0; i<3; i++)
  {
    for(int j=0; j<3; j++)
      printf("%d\t",matrix[i][j]);

    printf("\n");
  }
}

int main()
{
  int a[][3] = { {5,6,7}, {8,9,10}, {3,1,2} };
  int b[][3] = { {1,2,3}, {4,5,6}, {7,8,9} };
  int c[3][3];

  printf("First Matrix:\n");
  display(a);
  printf("Second Matrix:\n");
  display(b);

  int choice;
  do
  {
    printf("\nChoose the matrix operation,\n");
    printf("----------------------------\n");
    printf("1. Addition\n");
    printf("2. Subtraction\n");
    printf("3. Multiplication\n");
    printf("4. Transpose\n");
```



```
        printf("5. Exit\n");
        printf("---------------------------\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
          case 1:
            add(a, b, c);
            printf("Sum of matrix: \n");
            display(c);
            break;
          case 2:
            subtract(a, b, c);
            printf("Subtraction of matrix: \n");
            display(c);
            break;
          case 3:
            multiply(a, b, c);
            printf("Multiplication of matrix: \n");
            display(c);
            break;
          case 4:
            printf("Transpose of the first matrix: \n");
            transpose(a, c);
            display(c);
            printf("Transpose of the second matrix: \n");
            transpose(b, c);
            display(c);
            break;
          case 5:
            printf("Thank You.\n");
            exit(0);
          default:
            printf("Invalid input.\n");
            printf("Please enter the correct input.\n");
        }
    }while(1);

    return 0;
}
```

Output-



```
First Matrix:
5       6       7
8       9       10
3       1       2
Second Matrix:
1       2       3
4       5       6
7       8       9

Choose the matrix operation,
----------------------------
1. Addition
2. Subtraction
3. Multiplication
4. Transpose
5. Exit
----------------------------
Enter your choice: 1
Sum of matrix:
6       8       10
12      14      16
10      9       11

Choose the matrix operation,
----------------------------
1. Addition
2. Subtraction
3. Multiplication
4. Transpose
5. Exit
----------------------------
Enter your choice: _
```

Experiment - 4

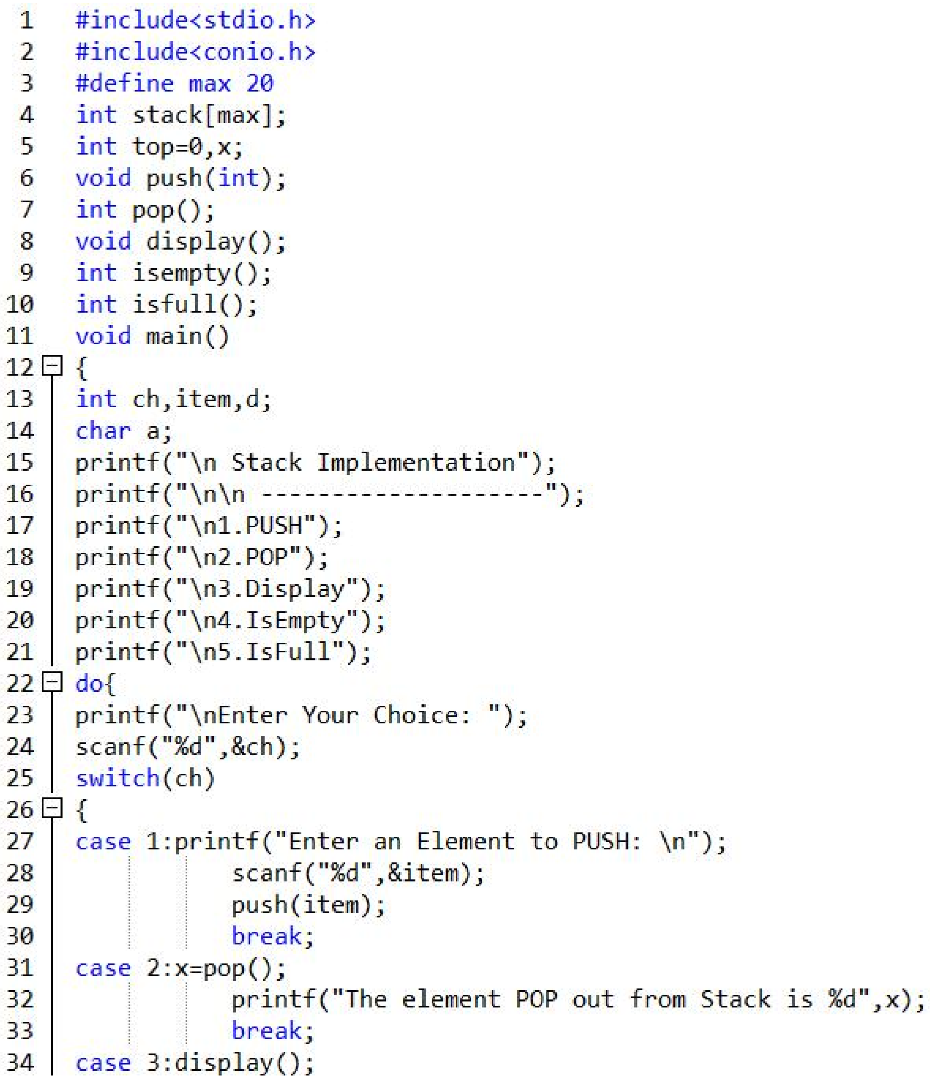
Q.Write a menu-based program to implement stack operations: PUSH, POP using array implementation of stack.

Theory-

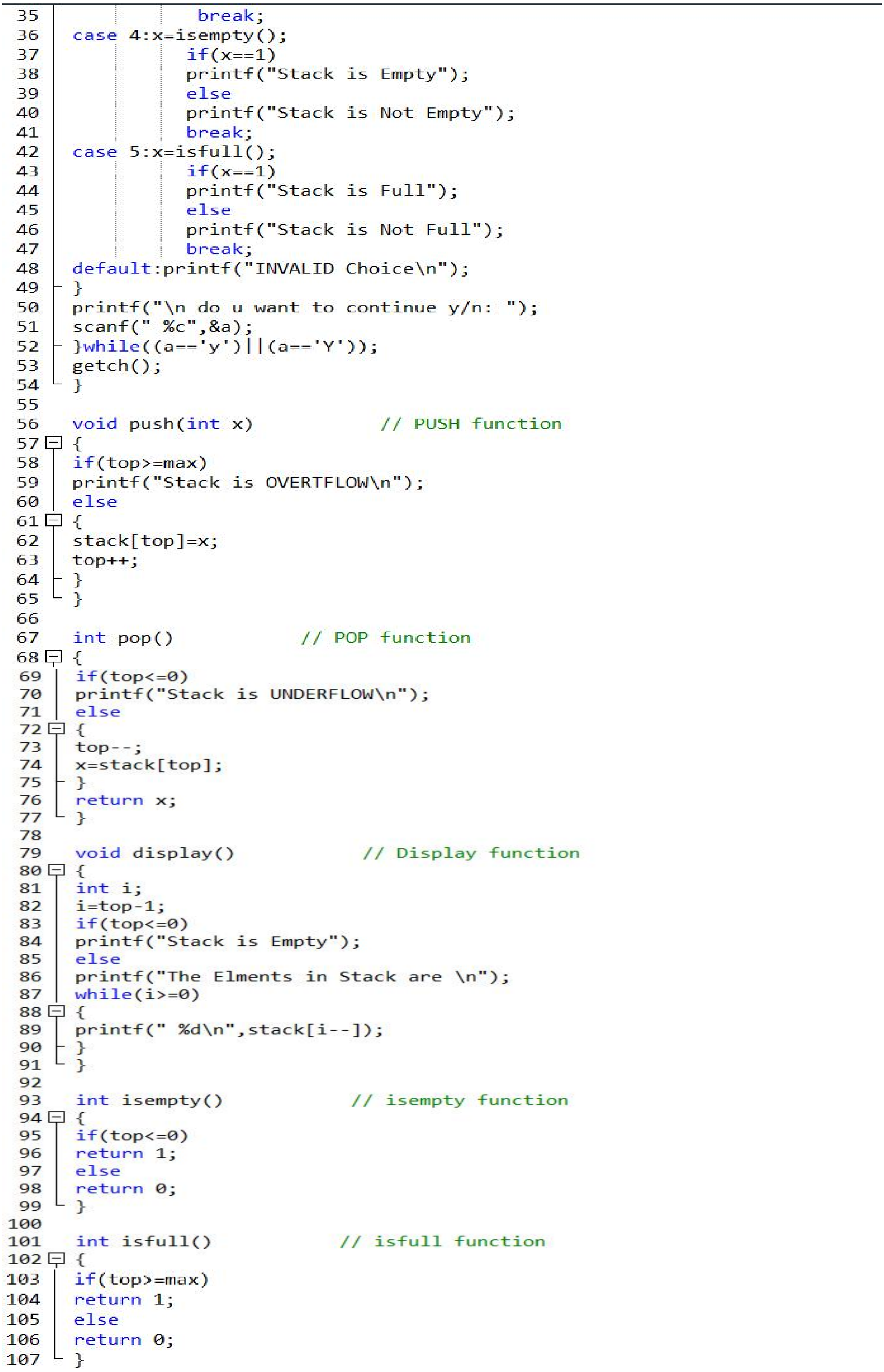
Menu-driven Stack Program

- The Stack is an Abstract Data Type in which the addition of an element to the collection is called PUSH and the removal of an element called POP.
- Stack follows a Last-In-First-Out (LIFO) data structure, the last element added to the structure must be the first one to be removed.
- A stack may be implemented to have a bounded capacity.
- If the stack is full and does not contain enough space to accept an entity to be pushed, the stack is then considered to be in an Overflow State.
- The pop operation removes an item from the top of the stack.
- A pop either reveals previously concealed items or results in an empty stack, but, if the stack is empty, it goes into an Underflow State, which means no items are present in the stack to be removed.

Source Code-

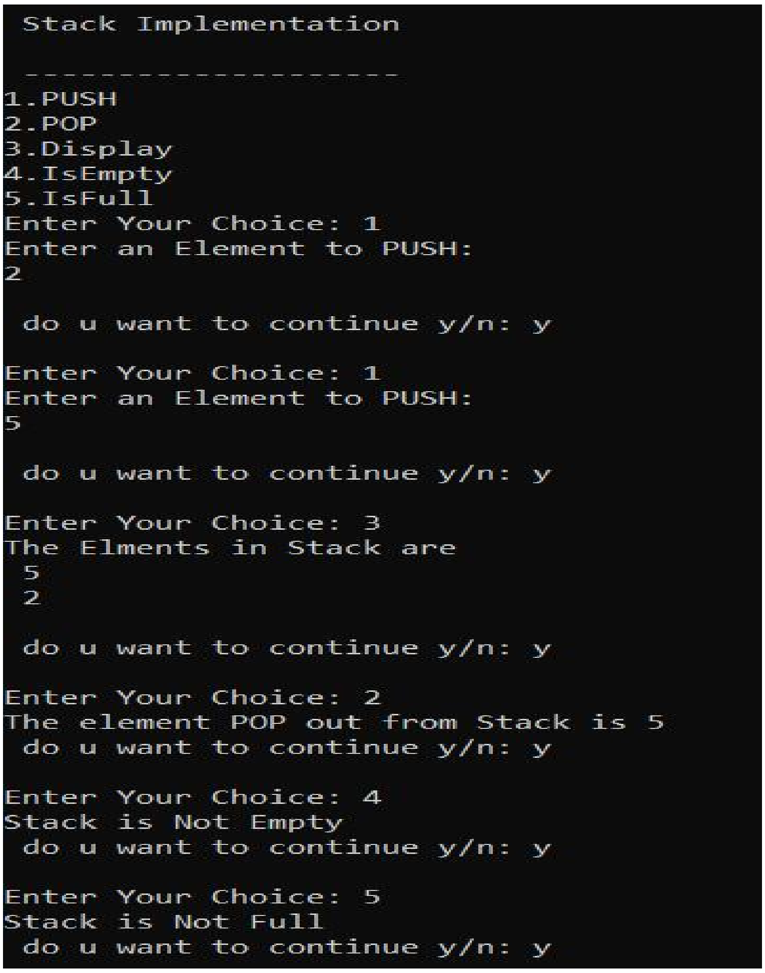


```c
#include<stdio.h>
#include<conio.h>
#define max 20
int stack[max];
int top=0,x;
void push(int);
int pop();
void display();
int isempty();
int isfull();
void main()
{
int ch,item,d;
char a;
printf("\n Stack Implementation");
printf("\n\n --------------------");
printf("\n1.PUSH");
printf("\n2.POP");
printf("\n3.Display");
printf("\n4.IsEmpty");
printf("\n5.IsFull");
do{
printf("\nEnter Your Choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("Enter an Element to PUSH: \n");
           scanf("%d",&item);
           push(item);
           break;
case 2:x=pop();
           printf("The element POP out from Stack is %d",x);
           break;
case 3:display();
```



```c
                  break;
    case 4:x=isempty();
                 if(x==1)
                 printf("Stack is Empty");
                 else
                 printf("Stack is Not Empty");
                 break;
    case 5:x=isfull();
                 if(x==1)
                 printf("Stack is Full");
                 else
                 printf("Stack is Not Full");
                 break;
    default:printf("INVALID Choice\n");
    }
    printf("\n do u want to continue y/n: ");
    scanf(" %c",&a);
    }while((a=='y')||(a=='Y'));
    getch();
    }

    void push(int x)            // PUSH function
    {
    if(top>=max)
    printf("Stack is OVERTFLOW\n");
    else
    {
    stack[top]=x;
    top++;
    }
    }

    int pop()            // POP function
    {
    if(top<=0)
    printf("Stack is UNDERFLOW\n");
    else
    {
    top--;
    x=stack[top];
    }
    return x;
    }

    void display()            // Display function
    {
    int i;
    i=top-1;
    if(top<=0)
    printf("Stack is Empty");
    else
    printf("The Elments in Stack are \n");
    while(i>=0)
    {
    printf(" %d\n",stack[i--]);
    }
    }

    int isempty()            // isempty function
    {
    if(top<=0)
    return 1;
    else
    return 0;
    }

    int isfull()            // isfull function
    {
    if(top>=max)
    return 1;
    else
    return 0;
    }
```

Output-



```
 Stack Implementation

 --------------------
1.PUSH
2.POP
3.Display
4.IsEmpty
5.IsFull
Enter Your Choice: 1
Enter an Element to PUSH:
2

 do u want to continue y/n: y

Enter Your Choice: 1
Enter an Element to PUSH:
5

 do u want to continue y/n: y

Enter Your Choice: 3
The Elments in Stack are
 5
 2

 do u want to continue y/n: y

Enter Your Choice: 2
The element POP out from Stack is 5
 do u want to continue y/n: y

Enter Your Choice: 4
Stack is Not Empty
 do u want to continue y/n: y

Enter Your Choice: 5
Stack is Not Full
 do u want to continue y/n: y
```

Experiment - 5

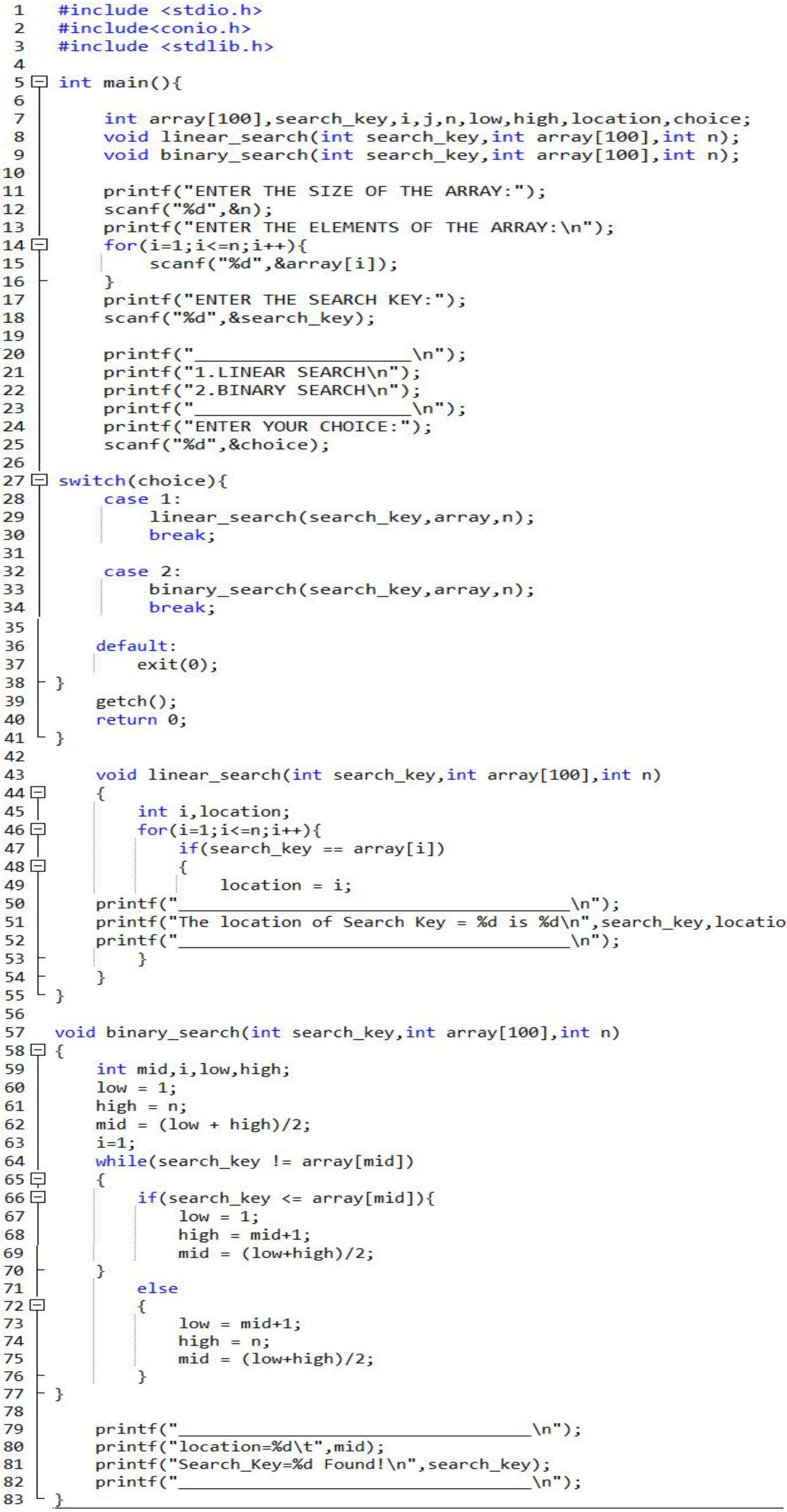
Q.Write a Program To Implement Linear Search and Binary Search Algorithm.

Theory-

Linear search and Binary search algorithm. The linear search is probably the oldest search algorithm, it goes through each and every element of the unsorted array and look for the key, you are searching for. However, the binary search, look for an element by dividing the array into two half, then compare the key element with a calculated mid value. If key is less than or equal to mid value, go left half and keep doing the same thing all over again. If the key is greater than the mid value, go to the right half of array, and perform the same steps again.

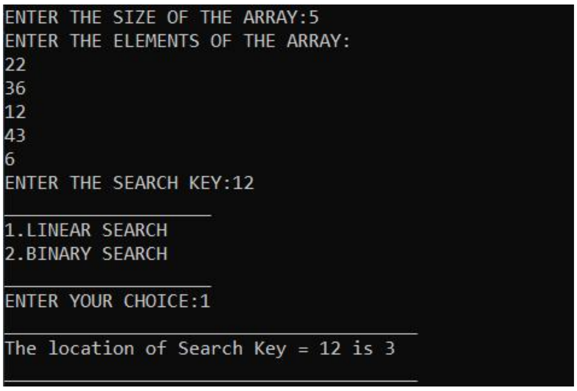
The binary search is faster, but it requires that the array is already sorted in ascending order or descending order. Otherwise it won’t work.

Source Code-

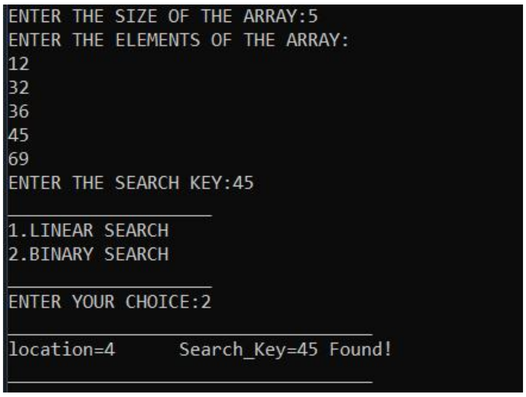


```c
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>

int main(){

    int array[100],search_key,i,j,n,low,high,location,choice;
    void linear_search(int search_key,int array[100],int n);
    void binary_search(int search_key,int array[100],int n);

    printf("ENTER THE SIZE OF THE ARRAY:");
    scanf("%d",&n);
    printf("ENTER THE ELEMENTS OF THE ARRAY:\n");
    for(i=1;i<=n;i++){
        scanf("%d",&array[i]);
    }
    printf("ENTER THE SEARCH KEY:");
    scanf("%d",&search_key);

    printf("___________________\n");
    printf("1.LINEAR SEARCH\n");
    printf("2.BINARY SEARCH\n");
    printf("___________________\n");
    printf("ENTER YOUR CHOICE:");
    scanf("%d",&choice);

switch(choice){
    case 1:
        linear_search(search_key,array,n);
        break;

    case 2:
        binary_search(search_key,array,n);
        break;

    default:
        exit(0);
}
    getch();
    return 0;
}

    void linear_search(int search_key,int array[100],int n)
    {
        int i,location;
        for(i=1;i<=n;i++){
            if(search_key == array[i])
            {
                location = i;
    printf("__________________________________\n");
    printf("The location of Search Key = %d is %d\n",search_key,location);
    printf("__________________________________\n");
        }
    }
}

void binary_search(int search_key,int array[100],int n)
{
    int mid,i,low,high;
    low = 1;
    high = n;
    mid = (low + high)/2;
    i=1;
    while(search_key != array[mid])
    {
        if(search_key <= array[mid]){
            low = 1;
            high = mid+1;
            mid = (low+high)/2;
    }
        else
        {
            low = mid+1;
            high = n;
            mid = (low+high)/2;
        }
}

    printf("______________________________\n");
    printf("location=%d\t",mid);
    printf("Search_Key=%d Found!\n",search_key);
    printf("______________________________\n");
}
```

Output-



```
ENTER THE SIZE OF THE ARRAY:5
ENTER THE ELEMENTS OF THE ARRAY:
22
36
12
43
6
ENTER THE SEARCH KEY:12
___________________
1.LINEAR SEARCH
2.BINARY SEARCH
___________________
ENTER YOUR CHOICE:1
______________________________________
The location of Search Key = 12 is 3
______________________________________
```



```
ENTER THE SIZE OF THE ARRAY:5
ENTER THE ELEMENTS OF THE ARRAY:
12
32
36
45
69
ENTER THE SEARCH KEY:45
___________________
1.LINEAR SEARCH
2.BINARY SEARCH
___________________
ENTER YOUR CHOICE:2
__________________________________
location=4      Search_Key=45 Found!
__________________________________
```

Experiment - 6

Q.Write menu driven program for all operations on singly linked list.

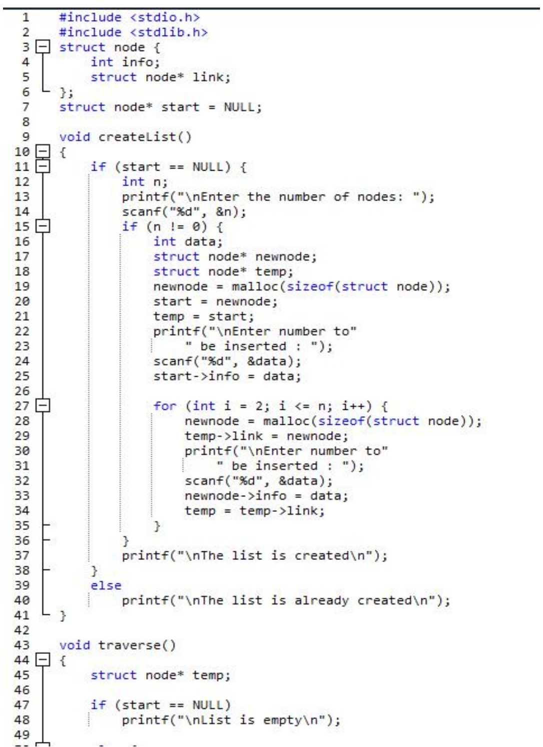
Theory-

A Linked List is a linear data structure that consists of two parts: one is the data part and the other is the address part.

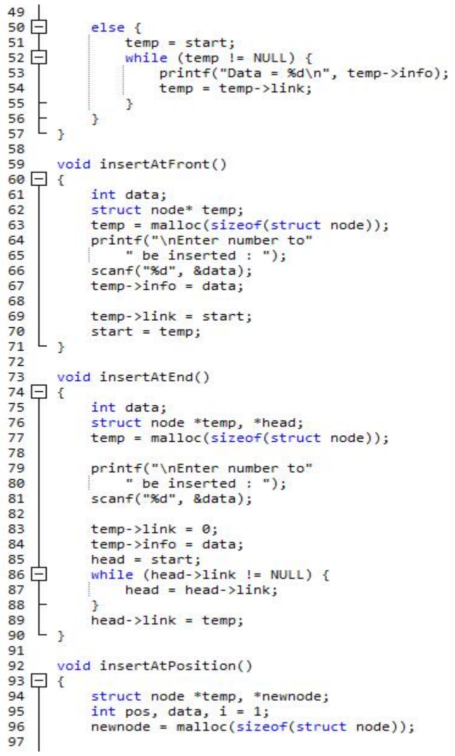
Operations to be performed:

- createList(): To create the list with ‘n’ number of nodes initially as defined by the user.
- traverse(): To see the contents of the linked list, it is necessary to traverse the given linked list. The given traverse() function traverses and prints the content of the linked list.
- insertAtFront(): This function simply inserts an element at the front/beginning of the linked list.
- insertAtEnd(): This function inserts an element at the end of the linked list.
- insertAtPosition(): This function inserts an element at a specified position in the linked list.
- deleteFirst(): This function simply deletes an element from the front/beginning of the linked list.
- deleteEnd(): This function simply deletes an element from the end of the linked list.
- deletePosition(): This function deletes an element from a specified position in the linked list.
- maximum(): This function finds the maximum element in a linked list.
- mean(): This function finds the mean of the elements in a linked list.
- sort(): This function sort the given linked list in ascending order.
- reverseLL(): This function reverses the given linked list.

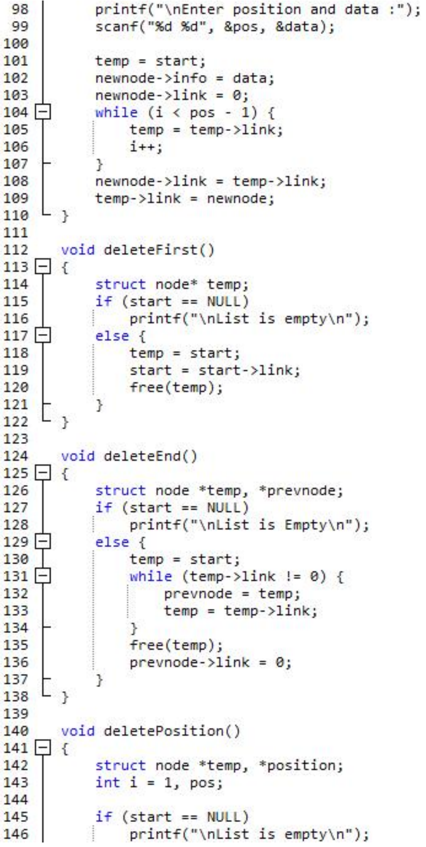
Source Code-



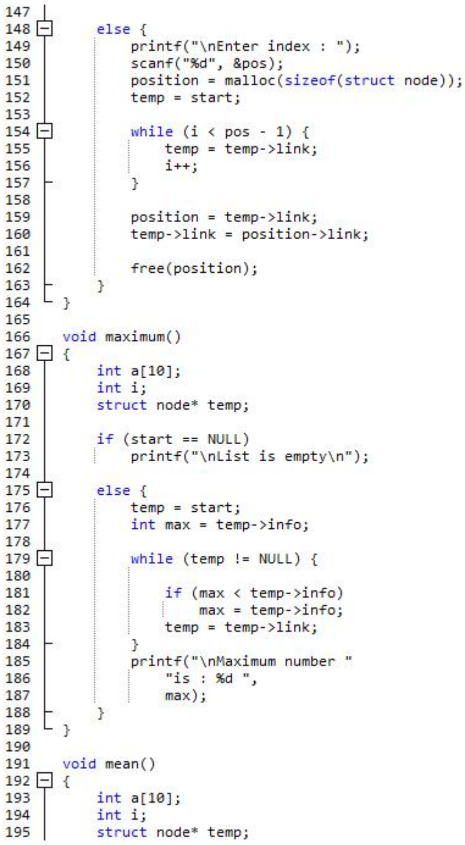
```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int info;
    struct node* link;
};
struct node* start = NULL;

void createList()
{
    if (start == NULL) {
        int n;
        printf("\nEnter the number of nodes: ");
        scanf("%d", &n);
        if (n != 0) {
            int data;
            struct node* newnode;
            struct node* temp;
            newnode = malloc(sizeof(struct node));
            start = newnode;
            temp = start;
            printf("\nEnter number to"
                " be inserted : ");
            scanf("%d", &data);
            start->info = data;

            for (int i = 2; i <= n; i++) {
                newnode = malloc(sizeof(struct node));
                temp->link = newnode;
                printf("\nEnter number to"
                    " be inserted : ");
                scanf("%d", &data);
                newnode->info = data;
                temp = temp->link;
            }
        }
        printf("\nThe list is created\n");
    }
    else
        printf("\nThe list is already created\n");
}

void traverse()
{
    struct node* temp;

    if (start == NULL)
        printf("\nList is empty\n");

```



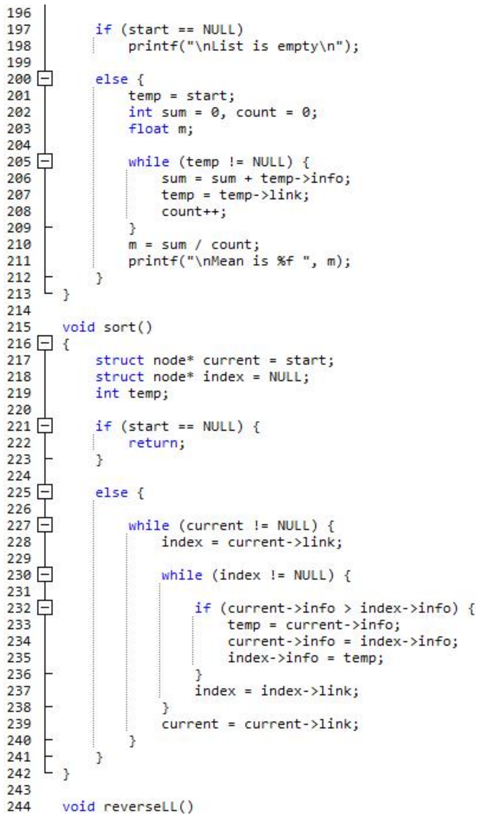
```

    else {
        temp = start;
        while (temp != NULL) {
            printf("Data = %d\n", temp->info);
            temp = temp->link;
        }
    }
}

void insertAtFront()
{
    int data;
    struct node* temp;
    temp = malloc(sizeof(struct node));
    printf("\nEnter number to"
        " be inserted : ");
    scanf("%d", &data);
    temp->info = data;

    temp->link = start;
    start = temp;
}

void insertAtEnd()
{
    int data;
    struct node *temp, *head;
    temp = malloc(sizeof(struct node));

    printf("\nEnter number to"
        " be inserted : ");
    scanf("%d", &data);

    temp->link = 0;
    temp->info = data;
    head = start;
    while (head->link != NULL) {
        head = head->link;
    }
    head->link = temp;
}

void insertAtPosition()
{
    struct node *temp, *newnode;
    int pos, data, i = 1;
    newnode = malloc(sizeof(struct node));

```



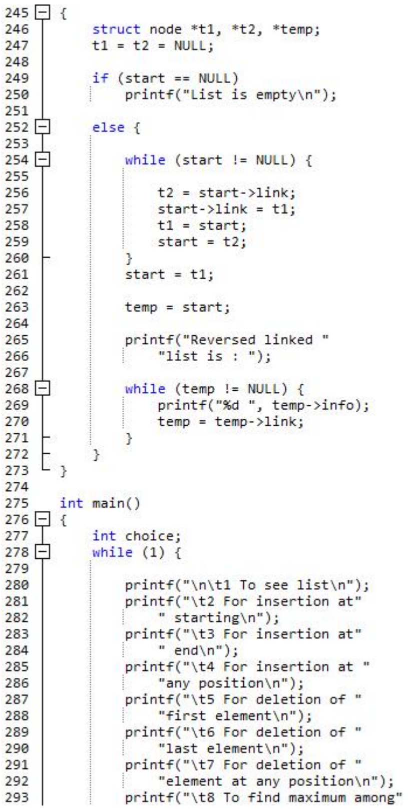
```
    printf("\nEnter position and data :");
    scanf("%d %d", &pos, &data);

    temp = start;
    newnode->info = data;
    newnode->link = 0;
    while (i < pos - 1) {
        temp = temp->link;
        i++;
    }
    newnode->link = temp->link;
    temp->link = newnode;
}

void deleteFirst()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else {
        temp = start;
        start = start->link;
        free(temp);
    }
}

void deleteEnd()
{
    struct node *temp, *prevnode;
    if (start == NULL)
        printf("\nList is Empty\n");
    else {
        temp = start;
        while (temp->link != 0) {
            prevnode = temp;
            temp = temp->link;
        }
        free(temp);
        prevnode->link = 0;
    }
}

void deletePosition()
{
    struct node *temp, *position;
    int i = 1, pos;

    if (start == NULL)
        printf("\nList is empty\n");
```



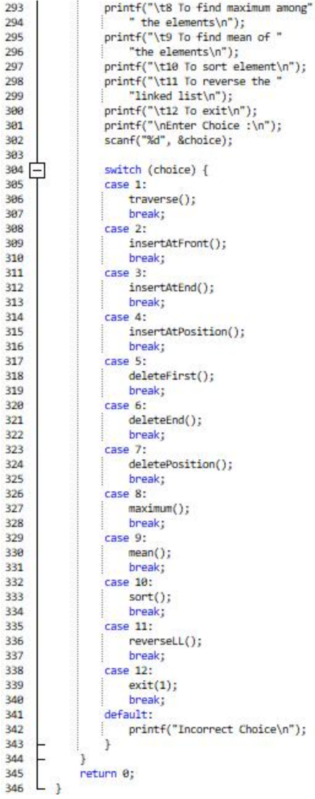
```

    else {
        printf("\nEnter index : ");
        scanf("%d", &pos);
        position = malloc(sizeof(struct node));
        temp = start;

        while (i < pos - 1) {
            temp = temp->link;
            i++;
        }

        position = temp->link;
        temp->link = position->link;

        free(position);
    }
}

void maximum()
{
    int a[10];
    int i;
    struct node* temp;

    if (start == NULL)
        printf("\nList is empty\n");

    else {
        temp = start;
        int max = temp->info;

        while (temp != NULL) {

            if (max < temp->info)
                max = temp->info;
            temp = temp->link;
        }
        printf("\nMaximum number "
            "is : %d ",
            max);
    }
}

void mean()
{
    int a[10];
    int i;
    struct node* temp;
```



```c

    if (start == NULL)
        printf("\nList is empty\n");

    else {
        temp = start;
        int sum = 0, count = 0;
        float m;

        while (temp != NULL) {
            sum = sum + temp->info;
            temp = temp->link;
            count++;
        }
        m = sum / count;
        printf("\nMean is %f ", m);
    }
}

void sort()
{
    struct node* current = start;
    struct node* index = NULL;
    int temp;

    if (start == NULL) {
        return;
    }

    else {

        while (current != NULL) {
            index = current->link;

            while (index != NULL) {

                if (current->info > index->info) {
                    temp = current->info;
                    current->info = index->info;
                    index->info = temp;
                }
                index = index->link;
            }
            current = current->link;
        }
    }
}

void reverseLL()
```



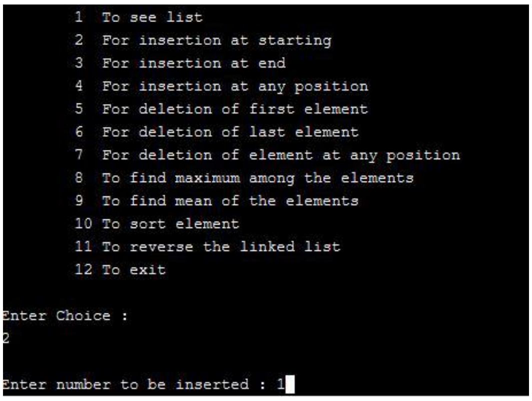
```c
{
    struct node *t1, *t2, *temp;
    t1 = t2 = NULL;

    if (start == NULL)
        printf("List is empty\n");

    else {

        while (start != NULL) {

            t2 = start->link;
            start->link = t1;
            t1 = start;
            start = t2;
        }
        start = t1;

        temp = start;

        printf("Reversed linked "
            "list is : ");

        while (temp != NULL) {
            printf("%d ", temp->info);
            temp = temp->link;
        }
    }
}

int main()
{
    int choice;
    while (1) {

        printf("\n\t1 To see list\n");
        printf("\t2 For insertion at"
            " starting\n");
        printf("\t3 For insertion at"
            " end\n");
        printf("\t4 For insertion at "
            "any position\n");
        printf("\t5 For deletion of "
            "first element\n");
        printf("\t6 For deletion of "
            "last element\n");
        printf("\t7 For deletion of "
            "element at any position\n");
        printf("\t8 To find maximum among"
```



```c
        printf("\t8 To find maximum among"
            " the elements\n");
        printf("\t9 To find mean of "
            "the elements\n");
        printf("\t10 To sort element\n");
        printf("\t11 To reverse the "
            "linked list\n");
        printf("\t12 To exit\n");
        printf("\nEnter Choice :\n");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            traverse();
            break;
        case 2:
            insertAtFront();
            break;
        case 3:
            insertAtEnd();
            break;
        case 4:
            insertAtPosition();
            break;
        case 5:
            deleteFirst();
            break;
        case 6:
            deleteEnd();
            break;
        case 7:
            deletePosition();
            break;
        case 8:
            maximum();
            break;
        case 9:
            mean();
            break;
        case 10:
            sort();
            break;
        case 11:
            reverseLL();
            break;
        case 12:
            exit(1);
            break;
        default:
            printf("Incorrect Choice\n");
        }
    }
    return 0;
}
```

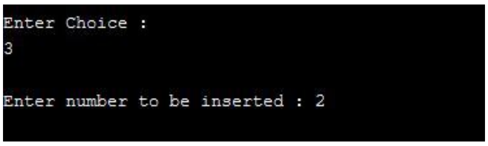
Output-

Insertion at the starting:



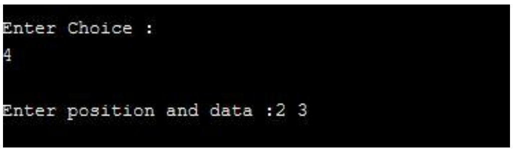
```
        1  To see list
        2  For insertion at starting
        3  For insertion at end
        4  For insertion at any position
        5  For deletion of first element
        6  For deletion of last element
        7  For deletion of element at any position
        8  To find maximum among the elements
        9  To find mean of the elements
        10 To sort element
        11 To reverse the linked list
        12 To exit

Enter Choice :
2

Enter number to be inserted : 1
```

Insertion at the end:



```
Enter Choice :
3

Enter number to be inserted : 2
```

Insertion at specific position:

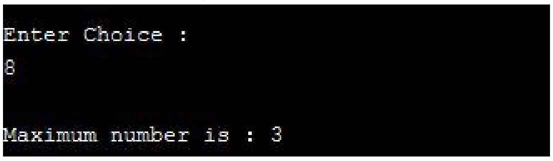


```
Enter Choice :
4

Enter position and data :2 3
```

Print the Linked List:

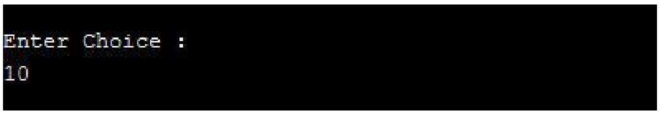
```
Enter Choice :
1
Data = 1
Data = 3
Data = 2
```

Maximum among Linked List:



```
Enter Choice :
8

Maximum number is : 3
```

Sorting the Linked List:



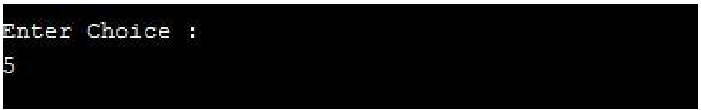
```
Enter Choice :
10
```

```
Enter Choice :
1
Data = 1
Data = 2
Data = 3
```

Reverse the Linked List:

```
Enter Choice :
11
Reversed linked list is : 3 2 1
```

Delete the first and last element with choice 5 and 6:



```
Enter Choice :
5
```

```
Enter Choice :
6
```

```
Enter Choice :
1
Data = 2
```

Experiment - 7

Q.Write a program to traverse a binary tree using PRE-ORDER, IN-ORDER, POST-ORDER traversal techniques and Binary Search Tree implementation.

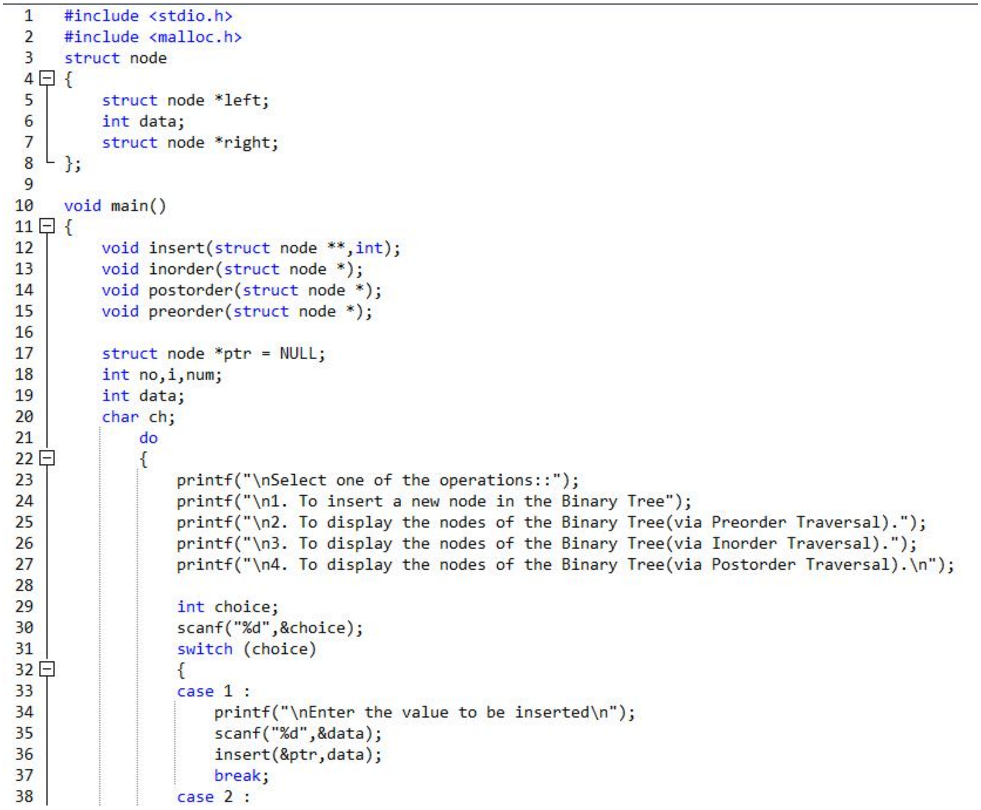
Theory-

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot random access a node in a tree.

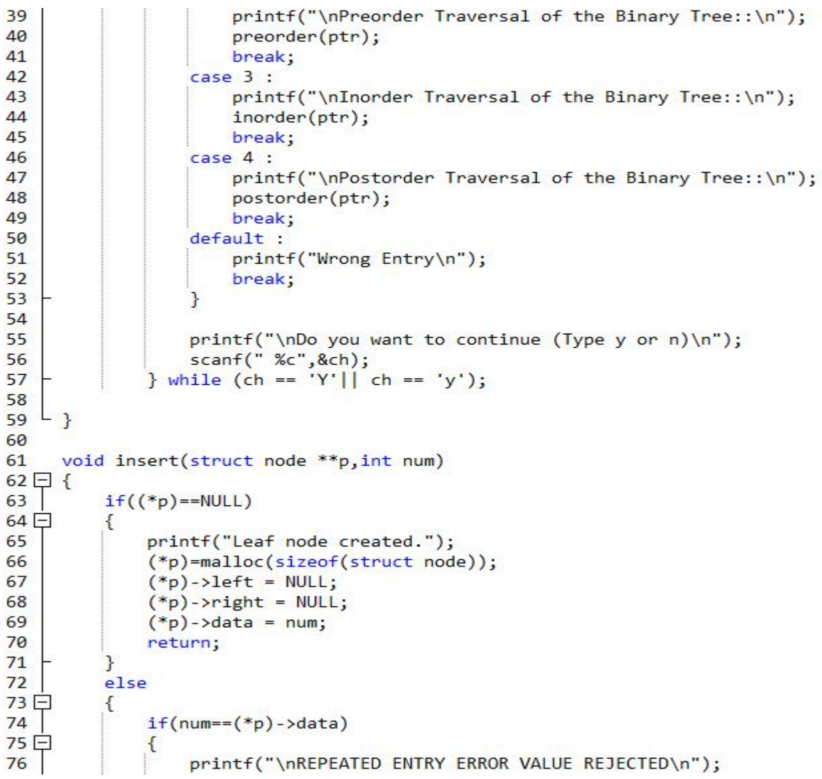
There are three ways which we use to traverse a tree –

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Source Code-

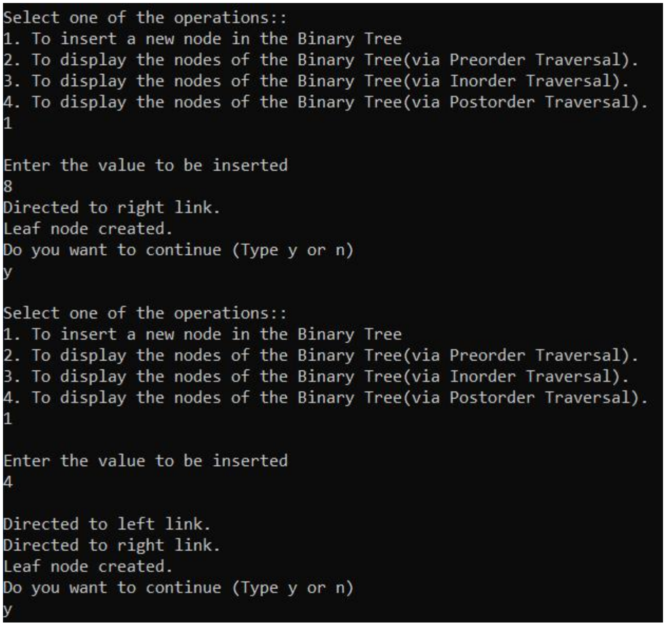


```
#include <stdio.h>
#include <malloc.h>
struct node
{
    struct node *left;
    int data;
    struct node *right;
};

void main()
{
    void insert(struct node **,int);
    void inorder(struct node *);
    void postorder(struct node *);
    void preorder(struct node *);

    struct node *ptr = NULL;
    int no,i,num;
    int data;
    char ch;
        do
        {
            printf("\nSelect one of the operations::");
            printf("\n1. To insert a new node in the Binary Tree");
            printf("\n2. To display the nodes of the Binary Tree(via Preorder Traversal).");
            printf("\n3. To display the nodes of the Binary Tree(via Inorder Traversal).");
            printf("\n4. To display the nodes of the Binary Tree(via Postorder Traversal).\n");

            int choice;
            scanf("%d",&choice);
            switch (choice)
            {
            case 1 :
                printf("\nEnter the value to be inserted\n");
                scanf("%d",&data);
                insert(&ptr,data);
                break;
            case 2 :
```

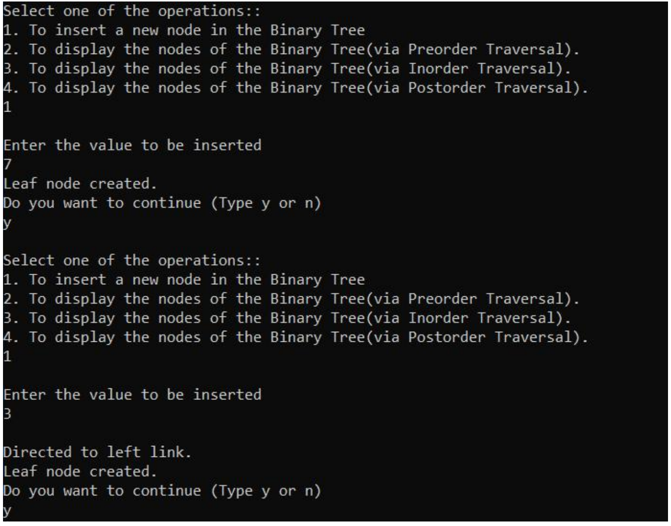


```
                printf("\nPreorder Traversal of the Binary Tree::\n");
                preorder(ptr);
                break;
            case 3 :
                printf("\nInorder Traversal of the Binary Tree::\n");
                inorder(ptr);
                break;
            case 4 :
                printf("\nPostorder Traversal of the Binary Tree::\n");
                postorder(ptr);
                break;
            default :
                printf("Wrong Entry\n");
                break;
            }

            printf("\nDo you want to continue (Type y or n)\n");
            scanf(" %c",&ch);
        } while (ch == 'Y'|| ch == 'y');

}

void insert(struct node **p,int num)
{
    if((*p)==NULL)
    {
        printf("Leaf node created.");
        (*p)=malloc(sizeof(struct node));
        (*p)->left = NULL;
        (*p)->right = NULL;
        (*p)->data = num;
        return;
    }
    else
    {
        if(num==(*p)->data)
        {
            printf("\nREPEATED ENTRY ERROR VALUE REJECTED\n");
```

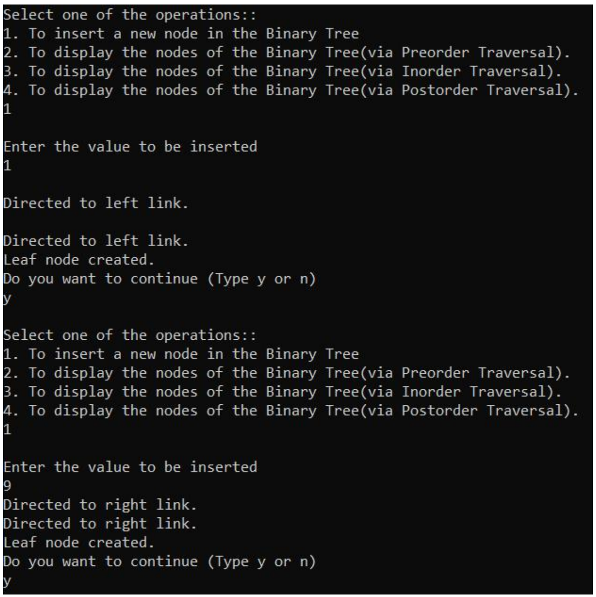
Output-



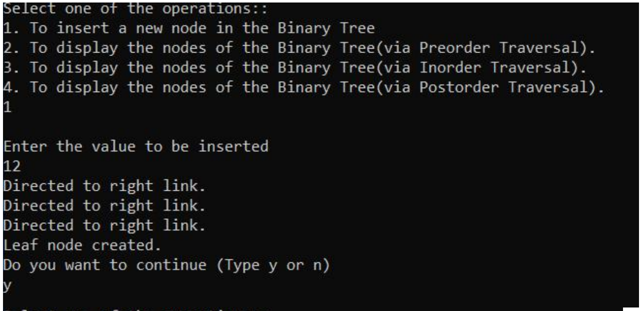
```
Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
1

Enter the value to be inserted
8
Directed to right link.
Leaf node created.
Do you want to continue (Type y or n)
y

Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
1

Enter the value to be inserted
4

Directed to left link.
Directed to right link.
Leaf node created.
Do you want to continue (Type y or n)
y
```



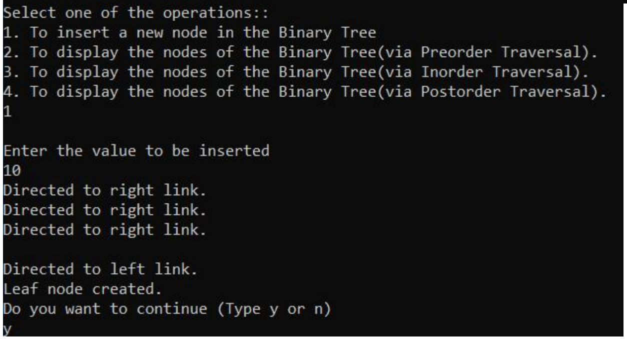
```
Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
1

Enter the value to be inserted
7
Leaf node created.
Do you want to continue (Type y or n)
y

Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
1

Enter the value to be inserted
3

Directed to left link.
Leaf node created.
Do you want to continue (Type y or n)
y
```



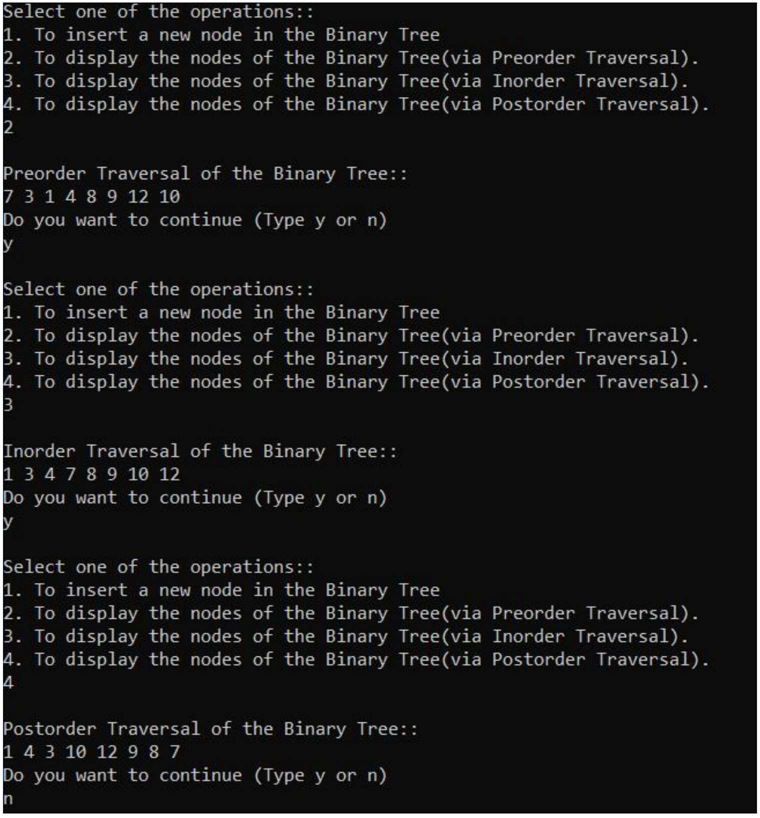
```
Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
1

Enter the value to be inserted
1

Directed to left link.

Directed to left link.
Leaf node created.
Do you want to continue (Type y or n)
y

Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
1

Enter the value to be inserted
9
Directed to right link.
Directed to right link.
Leaf node created.
Do you want to continue (Type y or n)
y
```



```
Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
1

Enter the value to be inserted
12
Directed to right link.
Directed to right link.
Directed to right link.
Leaf node created.
Do you want to continue (Type y or n)
y
```



```
Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
1

Enter the value to be inserted
10
Directed to right link.
Directed to right link.
Directed to right link.

Directed to left link.
Leaf node created.
Do you want to continue (Type y or n)
y
```



```
Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
2

Preorder Traversal of the Binary Tree::
7 3 1 4 8 9 12 10
Do you want to continue (Type y or n)
y

Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
3

Inorder Traversal of the Binary Tree::
1 3 4 7 8 9 10 12
Do you want to continue (Type y or n)
y

Select one of the operations::
1. To insert a new node in the Binary Tree
2. To display the nodes of the Binary Tree(via Preorder Traversal).
3. To display the nodes of the Binary Tree(via Inorder Traversal).
4. To display the nodes of the Binary Tree(via Postorder Traversal).
4

Postorder Traversal of the Binary Tree::
1 4 3 10 12 9 8 7
Do you want to continue (Type y or n)
n
```

Experiment - 8

Q.Write a program to traverse a graph using breadth-first search (BFS), depth-first search (DFS).

Theory-

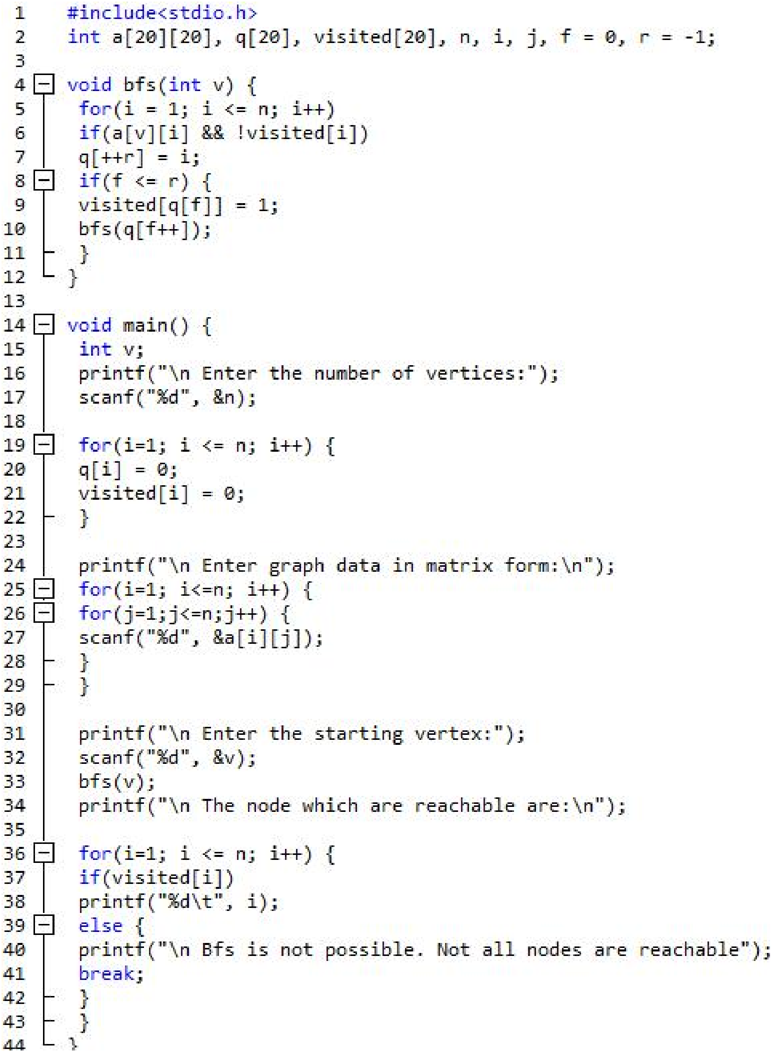
Breadth First Search is an algorithm used to search a Tree or Graph. BFS search starts from root node then traverses into next level of graph or tree, if item found it stops other wise it continues with other nodes in the same level before moving on to the next level. The algorithm can also be used for just Tree/Graph traversal, without actually searching for a value.

Depth First Search is an algorithm used to search the Tree or Graph. DFS search starts from root node then traversal into left child node and continues, if item found it stops other wise it continues.

The advantage of DFS is it requires less memory compare to Breadth First Search(BFS).

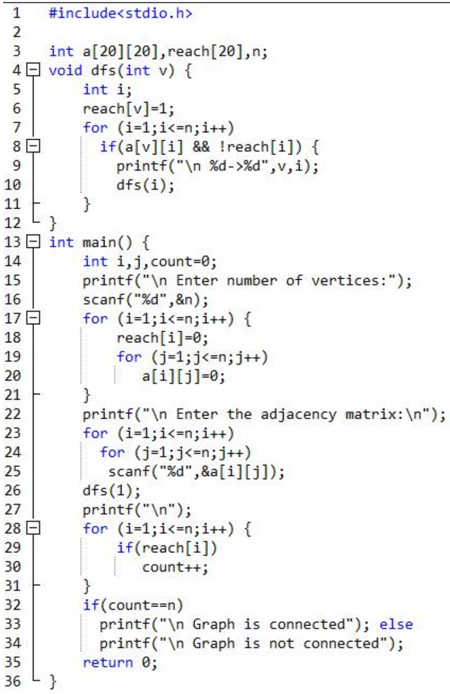
Source code-

//graph traversal using bfs



```
#include<stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v) {
 for(i = 1; i <= n; i++)
 if(a[v][i] && !visited[i])
 q[++r] = i;
 if(f <= r) {
 visited[q[f]] = 1;
 bfs(q[f++]);
 }
}

void main() {
 int v;
 printf("\n Enter the number of vertices:");
 scanf("%d", &n);

 for(i=1; i <= n; i++) {
 q[i] = 0;
 visited[i] = 0;
 }

 printf("\n Enter graph data in matrix form:\n");
 for(i=1; i<=n; i++) {
 for(j=1;j<=n;j++) {
 scanf("%d", &a[i][j]);
 }
 }

 printf("\n Enter the starting vertex:");
 scanf("%d", &v);
 bfs(v);
 printf("\n The node which are reachable are:\n");

 for(i=1; i <= n; i++) {
 if(visited[i])
 printf("%d\t", i);
 else {
 printf("\n Bfs is not possible. Not all nodes are reachable");
 break;
 }
 }
}
```

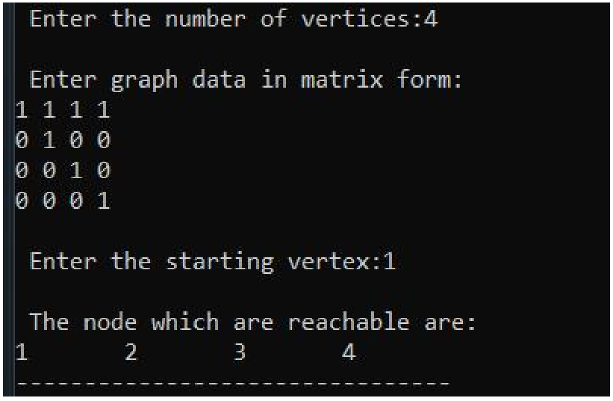
//graph traversal using dfs



```c
#include<stdio.h>

int a[20][20],reach[20],n;
void dfs(int v) {
    int i;
    reach[v]=1;
    for (i=1;i<=n;i++)
      if(a[v][i] && !reach[i]) {
        printf("\n %d->%d",v,i);
        dfs(i);
    }
}
int main() {
    int i,j,count=0;
    printf("\n Enter number of vertices:");
    scanf("%d",&n);
    for (i=1;i<=n;i++) {
        reach[i]=0;
        for (j=1;j<=n;j++)
           a[i][j]=0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
       scanf("%d",&a[i][j]);
    dfs(1);
    printf("\n");
    for (i=1;i<=n;i++) {
        if(reach[i])
           count++;
    }
    if(count==n)
      printf("\n Graph is connected"); else
      printf("\n Graph is not connected");
    return 0;
}
```

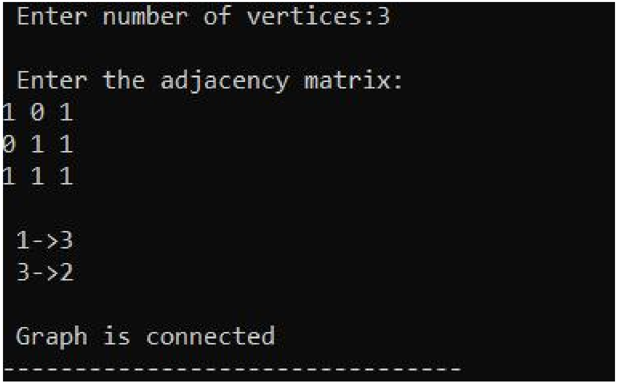
Output-

BFS implementation



```
 Enter the number of vertices:4

 Enter graph data in matrix form:
1 1 1 1
0 1 0 0
0 0 1 0
0 0 0 1

 Enter the starting vertex:1

 The node which are reachable are:
1       2       3       4
--------------------------------
```

DFS implementation



```
 Enter number of vertices:3

 Enter the adjacency matrix:
1 0 1
0 1 1
1 1 1

 1->3
 3->2

 Graph is connected
--------------------------------
```

Experiment - 9

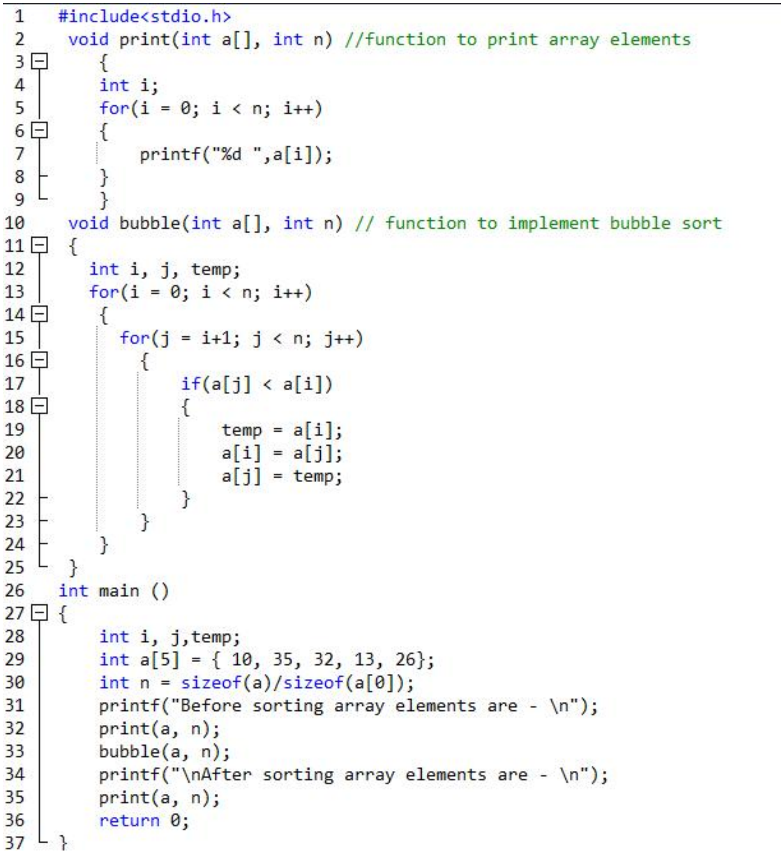
Q.Write a program to implement Bubble Sort.

Algorithm-

In the algorithm, suppose arr is an array of n elements. The assumed swap function in the algorithm will swap the values of given array elements.

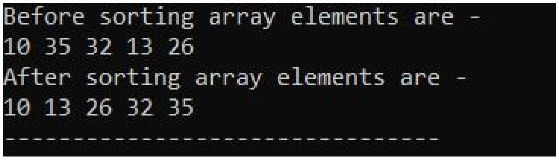
```
1.   begin BubbleSort(arr)
2.       for all array elements
3.           if arr[i] > arr[i+1]
4.               swap(arr[i], arr[i+1])
5.           end if
6.       end for
7.       return arr
8.   end BubbleSort
```

Source Code-



```c
#include<stdio.h>
 void print(int a[], int n) //function to print array elements
    {
    int i;
    for(i = 0; i < n; i++)
    {
        printf("%d ",a[i]);
    }
    }
 void bubble(int a[], int n) // function to implement bubble sort
 {
   int i, j, temp;
   for(i = 0; i < n; i++)
    {
      for(j = i+1; j < n; j++)
        {
            if(a[j] < a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
 }
int main ()
{
    int i, j,temp;
    int a[5] = { 10, 35, 32, 13, 26};
    int n = sizeof(a)/sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    print(a, n);
    bubble(a, n);
    printf("\nAfter sorting array elements are - \n");
    print(a, n);
    return 0;
}
```

Output-



```
Before sorting array elements are -
10 35 32 13 26
After sorting array elements are -
10 13 26 32 35
--------------------------------
```

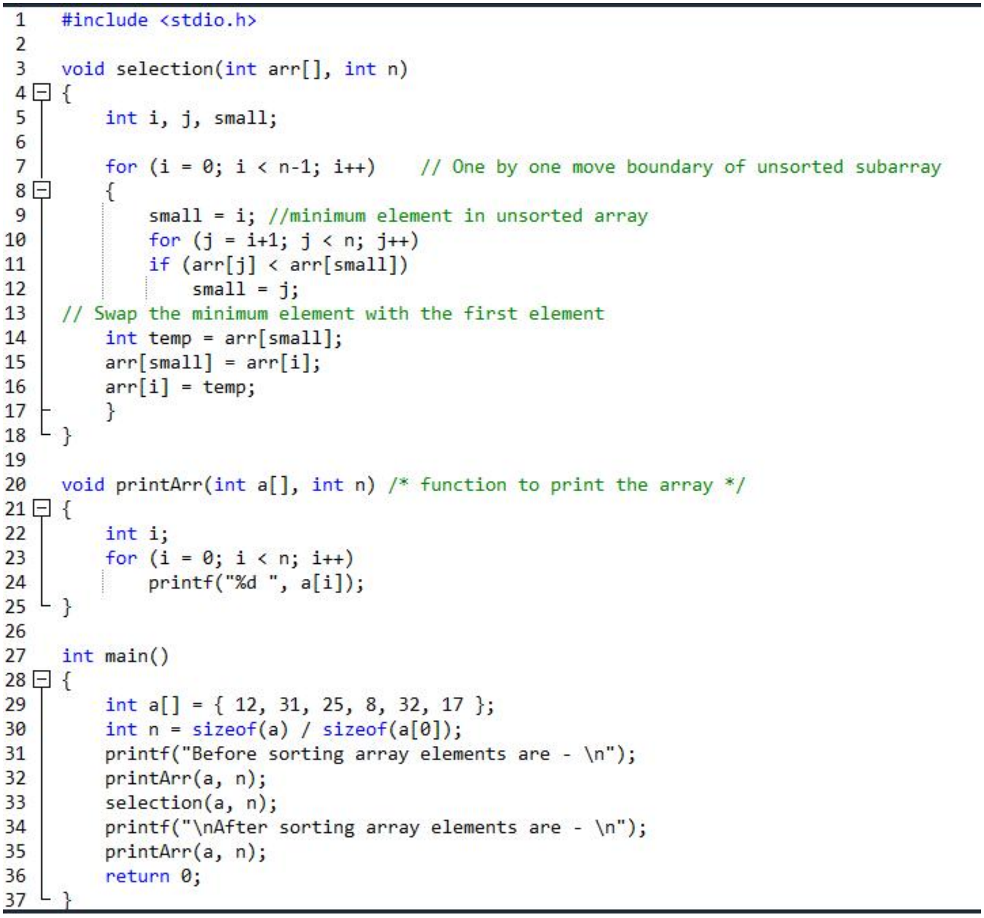
Experiment - 10

Q.Write a program to implement selection sort.

Algorithm-

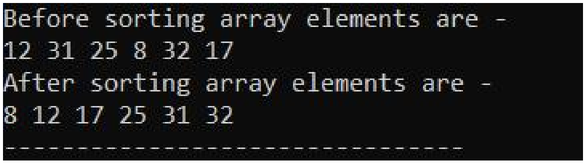
```
SELECTION SORT(arr, n)

Step 1: Repeat Steps 2 and 3 for i = 0 to n-1
Step 2: CALL SMALLEST(arr, i, n, pos)
Step 3: SWAP arr[i] with arr[pos]
[END OF LOOP]
Step 4: EXIT

SMALLEST (arr, i, n, pos)
Step 1: [INITIALIZE] SET SMALL = arr[i]
Step 2: [INITIALIZE] SET pos = i
Step 3: Repeat for j = i+1 to n
if (SMALL > arr[j])
      SET SMALL = arr[j]
SET pos = j
[END OF if]
[END OF LOOP]
Step 4: RETURN pos
```

Source Code-



```c
#include <stdio.h>

void selection(int arr[], int n)
{
    int i, j, small;

    for (i = 0; i < n-1; i++)    // One by one move boundary of unsorted subarray
    {
        small = i; //minimum element in unsorted array
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[small])
            small = j;
// Swap the minimum element with the first element
    int temp = arr[small];
    arr[small] = arr[i];
    arr[i] = temp;
    }
}

void printArr(int a[], int n) /* function to print the array */
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}

int main()
{
    int a[] = { 12, 31, 25, 8, 32, 17 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    selection(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);
    return 0;
}
```

Output-



```
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32
--------------------------------
```

Experiment - 12

Q.Design, develop, and execute a program in C to convert a given valid parenthesized infix arithmetic expression to postfix expression and then to print both the expressions and then to evaluate resultant expression using Stack. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide).

Algorithm -

Algorithm to convert Infix To Postfix

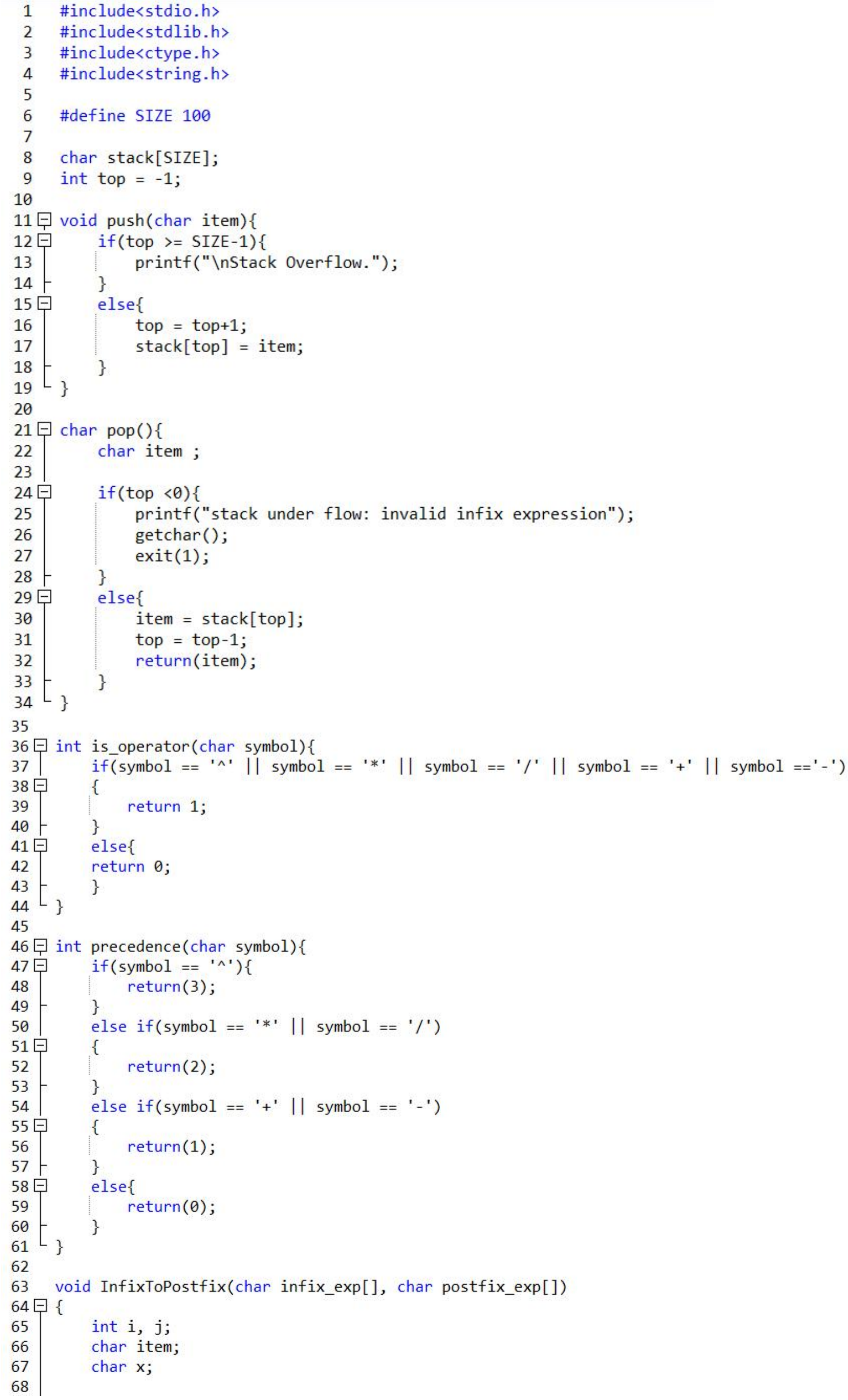
Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push “(“onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
    2. Add operator to Stack.
       [End of If]
6. If a right parenthesis is encountered ,then:

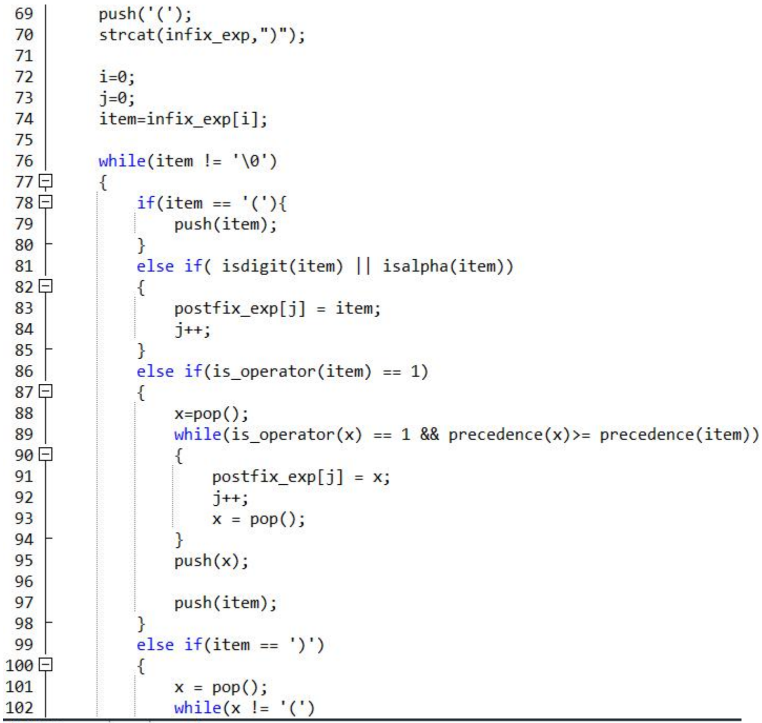
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
    2. Remove the left Parenthesis.
       [End of If]
       [End of If]

7. END.

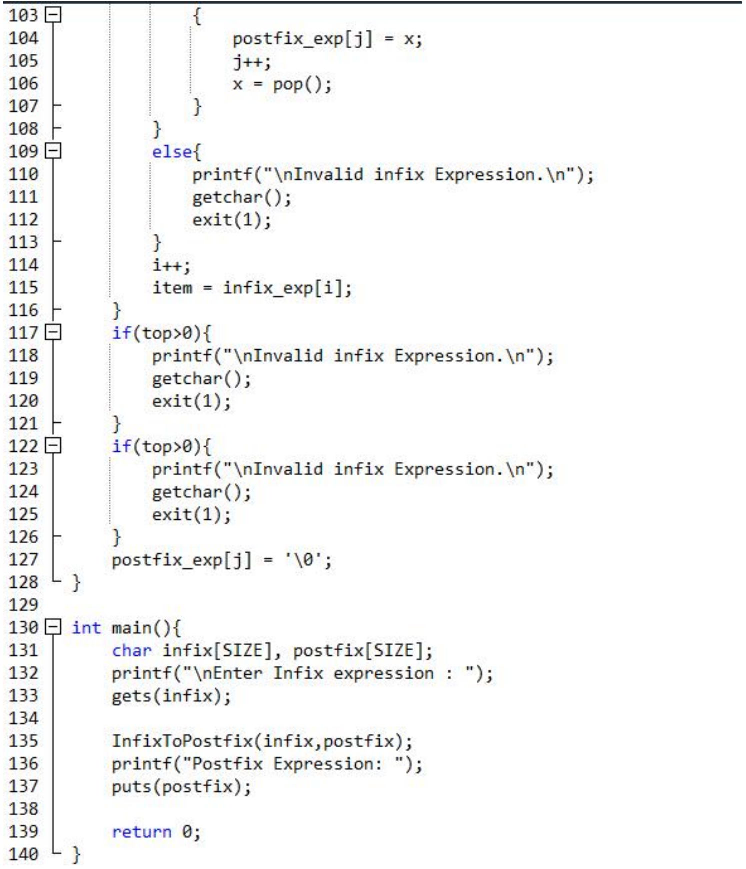
Source Code-



```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>

#define SIZE 100

char stack[SIZE];
int top = -1;

void push(char item){
    if(top >= SIZE-1){
        printf("\nStack Overflow.");
    }
    else{
        top = top+1;
        stack[top] = item;
    }
}

char pop(){
    char item ;

    if(top <0){
        printf("stack under flow: invalid infix expression");
        getchar();
        exit(1);
    }
    else{
        item = stack[top];
        top = top-1;
        return(item);
    }
}

int is_operator(char symbol){
    if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
    {
        return 1;
    }
    else{
    return 0;
    }
}

int precedence(char symbol){
    if(symbol == '^'){
        return(3);
    }
    else if(symbol == '*' || symbol == '/')
    {
        return(2);
    }
    else if(symbol == '+' || symbol == '-')
    {
        return(1);
    }
    else{
        return(0);
    }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
    int i, j;
    char item;
    char x;

```

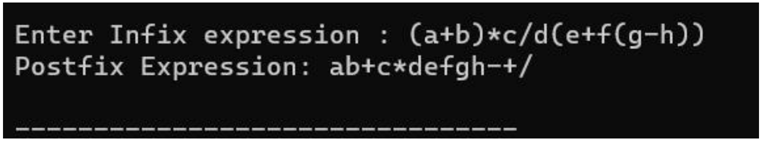


```c
    push('(');
    strcat(infix_exp,")");

    i=0;
    j=0;
    item=infix_exp[i];

    while(item != '\0')
    {
        if(item == '('){
            push(item);
        }
        else if( isdigit(item) || isalpha(item))
        {
            postfix_exp[j] = item;
            j++;
        }
        else if(is_operator(item) == 1)
        {
            x=pop();
            while(is_operator(x) == 1 && precedence(x)>= precedence(item))
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
            push(x);

            push(item);
        }
        else if(item == ')')
        {
            x = pop();
            while(x != '(')
```



```c
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
        }
        else{
            printf("\nInvalid infix Expression.\n");
            getchar();
            exit(1);
        }
        i++;
        item = infix_exp[i];
    }
    if(top>0){
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
    }
    if(top>0){
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
    }
    postfix_exp[j] = '\0';
}

int main(){
    char infix[SIZE], postfix[SIZE];
    printf("\nEnter Infix expression : ");
    gets(infix);

    InfixToPostfix(infix,postfix);
    printf("Postfix Expression: ");
    puts(postfix);

    return 0;
}
```

Output-



```
Enter Infix expression : (a+b)*c/d(e+f(g-h))
Postfix Expression: ab+c*defgh-+/


-----------------------------------
```