

# INDEX

S. No.	Experiments	Date	Sign
1	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.		
2	Write a program to implement 2D Array and also write a program to search an element in a 2-dimensional array.		
3	Write a program to perform following operations in matrix: Addition, Subtraction, Multiplication, Transpose		
4	Write a menu-based program to implement stack operations: PUSH, POP using array implementation of stack.		
5	Write a Program To Implement Linear Search and Binary Search Algorithm		
6	Write menu driven program for all operations on singly linked list.		
7	Write a program to traverse a binary tree using PRE-ORDER, IN-ORDER, POST-ORDER traversal techniques and implementation of Binary Search Tree.		
8	Write a program to traverse a graph using breadth-first search (BFS), depth-first search (DFS).		
9	Write a program to implement Bubble Sort.		
10	Write a program to implement selection sort.		
11	Implement the Towers of Hanoi Problem using graphical view.		
12	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).		

## 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 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-

---

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  int a[10], pos, elem;
5  int n = 0;
6
7  void create();
8  void display();
9  void insert();
10 void del();
11
12 int main(){
13
14     int choice;
15     while(1){
16         printf("\n\n~~~~~MENU~~~~~");
17         printf("\n1. Create an array of N integers");
18         printf("\n2. Display of array elements");
19         printf("\n3. Insert an element at a given position");
20         printf("\n4. Delete an element at a given position");
21         printf("\n5. Exit");
22         printf("\nEnter your choice: ");
23
24         scanf("%d", &choice);
25
26         switch(choice){
27             case 1: create();
28                     break;
29
30             case 2: display();
31                     break;
32
33             case 3: insert();
34                     break;
```

```

35
36     case 4:del();
37         break;
38
39     case 5:exit(1);
40         break;
41
42     default:printf("\nPlease enter a valid choice:");
43 }
44 }
45 }
46
47 void create(){
48     int i;
49     printf("\nEnter the number of elements: ");
50     scanf("%d", &n);
51
52     printf("\nEnter the elements: ");
53
54     for(i=0; i<n; i++){
55         scanf("%d", &a[i]);
56     }
57 }
58
59 void display(){
60     int i;
61
62     if(n == 0){
63         printf("\nNo elements to display");
64         return;
65     }
66     printf("\nArray elements are: ");
67
68     for(i=0; i<n;i++)
69         printf("%d\t ", a[i]);
70 }
71
72 void insert(){
73     int i;
74
75     if(n == 5){
76         printf("\nArray is full. Insertion is not possible");
77         return;
78     }
79     do{
80         printf("\nEnter a valid position where element to be inserted: ");
81         scanf("%d", &pos);
82     }
83     while(pos > n);
84     printf("\nEnter the value to be inserted: ");
85     scanf("%d", &elem);
86
87     for(i=n-1; i>=pos ; i--){
88         a[i+1] = a[i];
89     }
90     a[pos] = elem;
91     n = n+1;
92     display();
93 }
94
95 void del(){
96     int i;
97     if(n == 0){
98         printf("\nArray is empty and no elements to delete");
99     }
100 }
101
102

```

```

103 |         return;
104 |     }
105 |     do{
106 |         printf("\nEnter a valid position from where element to be deleted: ");
107 |         scanf("%d", &pos);
108 |     }
109 |     while(pos>=n);
110 |     elem = a[pos];
111 |
112 |     printf("\nDeleted element is : %d \n", elem);
113 |
114 |     for( i = pos; i< n-1; i++)
115 |     {
116 |         a[i] = a[i+1];
117 |     }
118 |     n = n-1;
119 |     display();
120 | }

```

## 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

```

1  #include<stdio.h>
2
3  int main(){
4      int rows, columns, srchElement, count=0;
5
6      printf("Enter the number of Row and Column: \n");
7      scanf("%d %d", &rows, &columns);
8
9      int array[rows][columns];
10     printf("Enter %d elements: \n", (rows*columns));
11     for(int i=0; i<rows; i++){
12         for(int j=0; j<columns; j++){
13             scanf("%d", &array[i][j]);
14         }
15     }
16     printf("Enter the element to get the position: \n");
17     scanf("%d", &srchElement);
18
19     for(int i=0; i<rows; i++){
20         for(int j=0; j<columns; j++){
21             if(array[i][j] == srchElement){
22                 printf("(%d, %d) \n", i, j);
23                 count++;
24             }
25         }
26     }
27     if(count==0)
28         printf("Not found \n");
29
30     return 0;
31 }

```

### 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  $A=[a_{ij}]$  be a matrix of order  $m \times n$ , 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  $A^T$ .



## Source Code-

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  void add(int m[3][3], int n[3][3], int sum[3][3])
5  {
6      for(int i=0;i<3;i++)
7          for(int j=0;j<3;j++)
8              sum[i][j] = m[i][j] + n[i][j];
9  }
10
11 void subtract(int m[3][3], int n[3][3], int result[3][3])
12 {
13     for(int i=0;i<3;i++)
14         for(int j=0;j<3;j++)
15             result[i][j] = m[i][j] - n[i][j];
16 }
17
18 void multiply(int m[3][3], int n[3][3], int result[3][3])
19 {
20     for(int i=0; i < 3; i++)
21     {
22         for(int j=0; j < 3; j++)
23         {
24             result[i][j] = 0;
25             for (int k = 0; k < 3; k++)
26                 result[i][j] += m[i][k] * n[k][j];
27         }
28     }
29 }
30
31 void transpose(int matrix[3][3], int trans[3][3])
32 {
33     for (int i = 0; i < 3; i++)
34         for (int j = 0; j < 3; j++)
35             trans[i][j] = matrix[j][i];
36 }
37
38 void display(int matrix[3][3])
39 {
40     for(int i=0; i<3; i++)
41     {
42         for(int j=0; j<3; j++)
43             printf("%d\t",matrix[i][j]);
44
45         printf("\n");
46     }
47 }
48
49 int main()
50 {
51     int a[][3] = { {5,6,7}, {8,9,10}, {3,1,2} };
52     int b[][3] = { {1,2,3}, {4,5,6}, {7,8,9} };
53     int c[3][3];
54
55     printf("First Matrix:\n");
56     display(a);
57     printf("Second Matrix:\n");
58     display(b);
59
60     int choice;
61     do
62     {
63         printf("\nChoose the matrix operation,\n");
64         printf("-----\n");
65         printf("1. Addition\n");
66         printf("2. Subtraction\n");
67         printf("3. Multiplication\n");
68         printf("4. Transpose\n");
```

```

69     printf("5. Exit\n");
70     printf("-----\n");
71     printf("Enter your choice: ");
72     scanf("%d", &choice);
73
74     switch (choice) {
75     case 1:
76         add(a, b, c);
77         printf("Sum of matrix: \n");
78         display(c);
79         break;
80     case 2:
81         subtract(a, b, c);
82         printf("Subtraction of matrix: \n");
83         display(c);
84         break;
85     case 3:
86         multiply(a, b, c);
87         printf("Multiplication of matrix: \n");
88         display(c);
89         break;
90     case 4:
91         printf("Transpose of the first matrix: \n");
92         transpose(a, c);
93         display(c);
94         printf("Transpose of the second matrix: \n");
95         transpose(b, c);
96         display(c);
97         break;
98     case 5:
99         printf("Thank You.\n");
100        exit(0);
101    default:
102        printf("Invalid input.\n");
103        printf("Please enter the correct input.\n");
104    }
105    }while(1);
106
107    return 0;
108 }

```

## 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-**

```

1  #include<stdio.h>
2  #include<conio.h>
3  #define max 20
4  int stack[max];
5  int top=0,x;
6  void push(int);
7  int pop();
8  void display();
9  int isempty();
10 int isfull();
11 void main()
12 {
13     int ch,item,d;
14     char a;
15     printf("\n Stack Implementation");
16     printf("\n\n -----");
17     printf("\n1.PUSH");
18     printf("\n2.POP");
19     printf("\n3.Display");
20     printf("\n4.IsEmpty");
21     printf("\n5.IsFull");
22 do{
23     printf("\nEnter Your Choice: ");
24     scanf("%d",&ch);
25     switch(ch)
26     {
27     case 1:printf("Enter an Element to PUSH: \n");
28             scanf("%d",&item);
29             push(item);
30             break;
31     case 2:x=pop();
32             printf("The element POP out from Stack is %d",x);
33             break;
34     case 3:display();

```

```

35         break;
36     case 4:x=isempty();
37         if(x==1)
38             printf("Stack is Empty");
39         else
40             printf("Stack is Not Empty");
41         break;
42     case 5:x=isfull();
43         if(x==1)
44             printf("Stack is Full");
45         else
46             printf("Stack is Not Full");
47         break;
48     default:printf("INVALID Choice\n");
49 }
50 printf("\n do u want to continue y/n: ");
51 scanf(" %c",&a);
52 }while((a=='y')||(a=='Y'));
53 getch();
54 }
55
56 void push(int x)                // PUSH function
57 {
58     if(top>=max)
59         printf("Stack is OVERTFLOW\n");
60     else
61     {
62         stack[top]=x;
63         top++;
64     }
65 }
66
67 int pop()                      // POP function
68 {
69     if(top<=0)
70         printf("Stack is UNDERFLOW\n");
71     else
72     {
73         top--;
74         x=stack[top];
75     }
76     return x;
77 }
78
79 void display()                 // Display function
80 {
81     int i;
82     i=top-1;
83     if(top<=0)
84         printf("Stack is Empty");
85     else
86         printf("The Elments in Stack are \n");
87     while(i>=0)
88     {
89         printf(" %d\n",stack[i--]);
90     }
91 }
92
93 int isempty()                 // isempty function
94 {
95     if(top<=0)
96         return 1;
97     else
98         return 0;
99 }
100
101 int isfull()                  // isfull function
102 {
103     if(top>=max)
104         return 1;
105     else
106         return 0;
107 }

```

## 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 Elements 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-

```

1  #include <stdio.h>
2  #include<conio.h>
3  #include <stdlib.h>
4
5  int main(){
6
7      int array[100],search_key,i,j,n,low,high,location,choice;
8      void linear_search(int search_key,int array[100],int n);
9      void binary_search(int search_key,int array[100],int n);
10
11     printf("ENTER THE SIZE OF THE ARRAY:");
12     scanf("%d",&n);
13     printf("ENTER THE ELEMENTS OF THE ARRAY:\n");
14     for(i=1;i<=n;i++){
15         scanf("%d",&array[i]);
16     }
17     printf("ENTER THE SEARCH KEY:");
18     scanf("%d",&search_key);
19
20     printf("\n");
21     printf("1.LINEAR SEARCH\n");
22     printf("2.BINARY SEARCH\n");
23     printf("\n");
24     printf("ENTER YOUR CHOICE:");
25     scanf("%d",&choice);
26
27     switch(choice){
28         case 1:
29             linear_search(search_key,array,n);
30             break;
31
32         case 2:
33             binary_search(search_key,array,n);
34             break;
35
36         default:
37             exit(0);
38     }
39     getch();
40     return 0;
41 }
42
43 void linear_search(int search_key,int array[100],int n)
44 {
45     int i,location;
46     for(i=1;i<=n;i++){
47         if(search_key == array[i])
48         {
49             location = i;
50         }
51     }
52     printf("The location of Search Key = %d is %d\n",search_key,location);
53     printf("\n");
54 }
55
56 void binary_search(int search_key,int array[100],int n)
57 {
58     int mid,i,low,high;
59     low = 1;
60     high = n;
61     mid = (low + high)/2;
62     i=1;
63     while(search_key != array[mid])
64     {
65         if(search_key <= array[mid]){
66             low = 1;
67             high = mid+1;
68             mid = (low+high)/2;
69         }
70         else
71         {
72             low = mid+1;
73             high = n;
74             mid = (low+high)/2;
75         }
76     }
77
78     printf("\n");
79     printf("location=%d\t",mid);
80     printf("Search_Key=%d Found!\n",search_key);
81     printf("\n");
82 }
83

```

## 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-



```

1  #include <stdio.h>
2  #include <stdlib.h>
3  struct node {
4      int info;
5      struct node* link;
6  };
7  struct node* start = NULL;
8
9  void createList()
10 {
11     if (start == NULL) {
12         int n;
13         printf("\nEnter the number of nodes: ");
14         scanf("%d", &n);
15         if (n != 0) {
16             int data;
17             struct node* newnode;
18             struct node* temp;
19             newnode = malloc(sizeof(struct node));
20             start = newnode;
21             temp = start;
22             printf("\nEnter number to"
23                 "\nbe inserted : ");
24             scanf("%d", &data);
25             start->info = data;
26
27             for (int i = 2; i <= n; i++) {
28                 newnode = malloc(sizeof(struct node));
29                 temp->link = newnode;
30                 printf("\nEnter number to"
31                     "\nbe inserted : ");
32                 scanf("%d", &data);
33                 newnode->info = data;
34                 temp = temp->link;
35             }
36             printf("\nThe list is created\n");
37         }
38         else
39             printf("\nThe list is already created\n");
40     }
41 }
42
43 void traverse()
44 {
45     struct node* temp;
46     if (start == NULL)
47         printf("\nlist is empty\n");
48
49     printf("\nEnter position and data :");
50     scanf("%d %d", &pos, &data);
51
52     temp = start;
53     newnode->info = data;
54     newnode->link = 0;
55     while (i < pos - 1) {
56         temp = temp->link;
57         i++;
58     }
59     newnode->link = temp->link;
60     temp->link = newnode;
61 }
62
63 void deleteFirst()
64 {
65     struct node* temp;
66     if (start == NULL)
67         printf("\nlist is empty\n");
68     else {
69         temp = start;
70         start = start->link;
71         free(temp);
72     }
73 }
74
75 void deleteEnd()
76 {
77     struct node *temp, *prevnode;
78     if (start == NULL)
79         printf("\nlist is Empty\n");
80     else {
81         temp = start;
82         while (temp->link != 0) {
83             prevnode = temp;
84             temp = temp->link;
85         }
86         free(temp);
87         prevnode->link = 0;
88     }
89 }
90
91 void deletePosition()
92 {
93     struct node *temp, *position;
94     int i = 1, pos;
95     if (start == NULL)
96         printf("\nlist is empty\n");

```

```

97     else {
98         temp = start;
99         while (temp != NULL) {
100             printf("Data = %d\n", temp->info);
101             temp = temp->link;
102         }
103     }
104 }
105
106 void insertAtFront()
107 {
108     int data;
109     struct node* temp;
110     temp = malloc(sizeof(struct node));
111     printf("\nEnter number to"
112         "\nbe inserted : ");
113     scanf("%d", &data);
114     temp->info = data;
115
116     temp->link = start;
117     start = temp;
118 }
119
120 void insertAtEnd()
121 {
122     int data;
123     struct node *temp, *head;
124     temp = malloc(sizeof(struct node));
125
126     printf("\nEnter number to"
127         "\nbe inserted : ");
128     scanf("%d", &data);
129
130     temp->link = 0;
131     temp->info = data;
132     head = start;
133     while (head->link != NULL) {
134         head = head->link;
135     }
136     head->link = temp;
137 }
138
139 void insertAtPosition()
140 {
141     struct node *temp, *newnode;
142     int pos, data, i = 1;
143     newnode = malloc(sizeof(struct node));
144
145     else {
146         printf("\nEnter index : ");
147         scanf("%d", &pos);
148         position = malloc(sizeof(struct node));
149         temp = start;
150
151         while (i < pos - 1) {
152             temp = temp->link;
153             i++;
154         }
155
156         position = temp->link;
157         temp->link = position->link;
158
159         free(position);
160
161         newnode->link = position->link;
162         temp->link = newnode;
163     }
164 }
165
166 void maximum()
167 {
168     int a[10];
169     int i;
170     struct node* temp;
171
172     if (start == NULL)
173         printf("\nlist is empty\n");
174     else {
175         temp = start;
176         int max = temp->info;
177
178         while (temp != NULL) {
179             if (max < temp->info)
180                 max = temp->info;
181             temp = temp->link;
182         }
183         printf("\nMaximum number "
184             "is : %d ",
185             max);
186     }
187 }
188
189 void mean()
190 {
191     int a[10];
192     int i;
193     struct node* temp;

```

```

196     if (start == NULL)
197         printf("\nList is empty\n");
198     else {
199         temp = start;
200         int sum = 0, count = 0;
201         float m;
202
203         while (temp != NULL) {
204             sum = sum + temp->info;
205             temp = temp->link;
206             count++;
207         }
208         m = sum / count;
209         printf("\nMean is %f ", m);
210     }
211 }
212
213 void sort()
214 {
215     struct node* current = start;
216     struct node* index = NULL;
217     int temp;
218
219     if (start == NULL) {
220         return;
221     }
222
223     else {
224         while (current != NULL) {
225             index = current->link;
226
227             while (index != NULL) {
228                 if (current->info > index->info) {
229                     temp = current->info;
230                     current->info = index->info;
231                     index->info = temp;
232                 }
233                 index = index->link;
234             }
235             current = current->link;
236         }
237     }
238 }
239
240 void reverseLL()
241 {
242     printf("\t8 To find maximum among "
243         " the elements\n");
244     printf("\t9 To find mean of "
245         "the elements\n");
246     printf("\t10 To sort element\n");
247     printf("\t11 To reverse the "
248         "linked list\n");
249     printf("\t12 To exit\n");
250     printf("\nEnter Choice :\n");
251     scanf("%d", &choice);
252
253     switch (choice) {
254         case 1:
255             traverse();
256             break;
257         case 2:
258             insertAtFront();
259             break;
260         case 3:
261             insertAtEnd();
262             break;
263         case 4:
264             insertAtPosition();
265             break;
266         case 5:
267             deleteFirst();
268             break;
269         case 6:
270             deleteEnd();
271             break;
272         case 7:
273             deletePosition();
274             break;
275         case 8:
276             maximum();
277             break;
278         case 9:
279             mean();
280             break;
281         case 10:
282             sort();
283             break;
284         case 11:
285             reverseLL();
286             break;
287         case 12:
288             exit(1);
289             break;
290         default:
291             printf("Incorrect Choice\n");
292     }
293
294     return 0;
295 }

```

```

245 {
246     struct node *t1, *t2, *temp;
247     t1 = t2 = NULL;
248
249     if (start == NULL)
250         printf("List is empty\n");
251     else {
252         while (start != NULL) {
253             t2 = start->link;
254             start->link = t1;
255             t1 = start;
256             start = t2;
257         }
258         start = t1;
259
260         temp = start;
261
262         printf("Reversed linked "
263             "list is : ");
264
265         while (temp != NULL) {
266             printf("%d ", temp->info);
267             temp = temp->link;
268         }
269     }
270 }
271
272 int main()
273 {
274     int choice;
275     while (1) {
276         printf("\n\t1 To see list\n");
277         printf("\t2 For insertion at "
278             " starting\n");
279         printf("\t3 For insertion at "
280             " end\n");
281         printf("\t4 For insertion at "
282             "any position\n");
283         printf("\t5 For deletion of "
284             "first element\n");
285         printf("\t6 For deletion of "
286             "last element\n");
287         printf("\t7 For deletion of "
288             "element at any position\n");
289         printf("\t8 To find maximum among "
290             "the elements\n");
291         printf("\t9 To find mean of "
292             "the elements\n");
293         printf("\t10 To sort element\n");
294         printf("\t11 To reverse the "
295             "linked list\n");
296         printf("\t12 To exit\n");
297         printf("\nEnter Choice :\n");
298         scanf("%d", &choice);
299
300         switch (choice) {
301             case 1:
302                 traverse();
303                 break;
304             case 2:
305                 insertAtFront();
306                 break;
307             case 3:
308                 insertAtEnd();
309                 break;
310             case 4:
311                 insertAtPosition();
312                 break;
313             case 5:
314                 deleteFirst();
315                 break;
316             case 6:
317                 deleteEnd();
318                 break;
319             case 7:
320                 deletePosition();
321                 break;
322             case 8:
323                 maximum();
324                 break;
325             case 9:
326                 mean();
327                 break;
328             case 10:
329                 sort();
330                 break;
331             case 11:
332                 reverseLL();
333                 break;
334             case 12:
335                 exit(1);
336                 break;
337             default:
338                 printf("Incorrect Choice\n");
339         }
340     }
341
342     return 0;
343 }

```

## 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-

```
1  #include <stdio.h>
2  #include <malloc.h>
3  struct node
4  {
5      struct node *left;
6      int data;
7      struct node *right;
8  };
9
10 void main()
11 {
12     void insert(struct node **,int);
13     void inorder(struct node *);
14     void postorder(struct node *);
15     void preorder(struct node *);
16
17     struct node *ptr = NULL;
18     int no,i,num;
19     int data;
20     char ch;
21     do
22     {
23         printf("\nSelect one of the operations::");
24         printf("\n1. To insert a new node in the Binary Tree");
25         printf("\n2. To display the nodes of the Binary Tree(via Preorder Traversal).");
26         printf("\n3. To display the nodes of the Binary Tree(via Inorder Traversal).");
27         printf("\n4. To display the nodes of the Binary Tree(via Postorder Traversal).\n");
28
29         int choice;
30         scanf("%d",&choice);
31         switch (choice)
32         {
33             case 1 :
34                 printf("\nEnter the value to be inserted\n");
35                 scanf("%d",&data);
36                 insert(&ptr,data);
37                 break;
38             case 2 :
```

```

39 |         printf("\nPreorder Traversal of the Binary Tree::\n");
40 |         preorder(ptr);
41 |         break;
42 |     case 3 :
43 |         printf("\nInorder Traversal of the Binary Tree::\n");
44 |         inorder(ptr);
45 |         break;
46 |     case 4 :
47 |         printf("\nPostorder Traversal of the Binary Tree::\n");
48 |         postorder(ptr);
49 |         break;
50 |     default :
51 |         printf("Wrong Entry\n");
52 |         break;
53 |     }
54 |
55 |     printf("\nDo you want to continue (Type y or n)\n");
56 |     scanf(" %c",&ch);
57 | } while (ch == 'Y' || ch == 'y');
58 |
59 | }
60 |
61 | void insert(struct node **p,int num)
62 | {
63 |     if((*p)==NULL)
64 |     {
65 |         printf("Leaf node created.");
66 |         (*p)=malloc(sizeof(struct node));
67 |         (*p)->left = NULL;
68 |         (*p)->right = NULL;
69 |         (*p)->data = num;
70 |         return;
71 |     }
72 |     else
73 |     {
74 |         if(num==(*p)->data)
75 |         {
76 |             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

```
1  #include<stdio.h>
2  int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;
3
4  void bfs(int v) {
5      for(i = 1; i <= n; i++)
6          if(a[v][i] && !visited[i])
7              q[++r] = i;
8      if(f <= r) {
9          visited[q[f]] = 1;
10         bfs(q[f++]);
11     }
12 }
13
14 void main() {
15     int v;
16     printf("\n Enter the number of vertices:");
17     scanf("%d", &n);
18
19     for(i=1; i <= n; i++) {
20         q[i] = 0;
21         visited[i] = 0;
22     }
23
24     printf("\n Enter graph data in matrix form:\n");
25     for(i=1; i<=n; i++) {
26         for(j=1; j<=n; j++) {
27             scanf("%d", &a[i][j]);
28         }
29     }
30
31     printf("\n Enter the starting vertex:");
32     scanf("%d", &v);
33     bfs(v);
34     printf("\n The node which are reachable are:\n");
35
36     for(i=1; i <= n; i++) {
37         if(visited[i])
38             printf("%d\t", i);
39         else {
40             printf("\n Bfs is not possible. Not all nodes are reachable");
41             break;
42         }
43     }
44 }
```

## //graph traversal using dfs

```
1 #include<stdio.h>
2
3 int a[20][20],reach[20],n;
4 void dfs(int v) {
5     int i;
6     reach[v]=1;
7     for (i=1;i<=n;i++)
8         if(a[v][i] && !reach[i]) {
9             printf("\n %d->%d",v,i);
10            dfs(i);
11        }
12 }
13 int main() {
14     int i,j,count=0;
15     printf("\n Enter number of vertices:");
16     scanf("%d",&n);
17     for (i=1;i<=n;i++) {
18         reach[i]=0;
19         for (j=1;j<=n;j++)
20             a[i][j]=0;
21     }
22     printf("\n Enter the adjacency matrix:\n");
23     for (i=1;i<=n;i++)
24         for (j=1;j<=n;j++)
25             scanf("%d",&a[i][j]);
26     dfs(1);
27     printf("\n");
28     for (i=1;i<=n;i++) {
29         if(reach[i])
30             count++;
31     }
32     if(count==n)
33         printf("\n Graph is connected"); else
34         printf("\n Graph is not connected");
35     return 0;
36 }
```

## 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-

```
1  #include<stdio.h>
2  void print(int a[], int n) //function to print array elements
3  {
4      int i;
5      for(i = 0; i < n; i++)
6      {
7          printf("%d ",a[i]);
8      }
9  }
10 void bubble(int a[], int n) // function to implement bubble sort
11 {
12     int i, j, temp;
13     for(i = 0; i < n; i++)
14     {
15         for(j = i+1; j < n; j++)
16         {
17             if(a[j] < a[i])
18             {
19                 temp = a[i];
20                 a[i] = a[j];
21                 a[j] = temp;
22             }
23         }
24     }
25 }
26 int main ()
27 {
28     int i, j,temp;
29     int a[5] = { 10, 35, 32, 13, 26};
30     int n = sizeof(a)/sizeof(a[0]);
31     printf("Before sorting array elements are - \n");
32     print(a, n);
33     bubble(a, n);
34     printf("\nAfter sorting array elements are - \n");
35     print(a, n);
36     return 0;
37 }
```

### 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-

```
1  #include <stdio.h>
2
3  void selection(int arr[], int n)
4  {
5      int i, j, small;
6
7      for (i = 0; i < n-1; i++)    // One by one move boundary of unsorted subarray
8      {
9          small = i; //minimum element in unsorted array
10         for (j = i+1; j < n; j++)
11             if (arr[j] < arr[small])
12                 small = j;
13         // Swap the minimum element with the first element
14         int temp = arr[small];
15         arr[small] = arr[i];
16         arr[i] = temp;
17     }
18 }
19
20 void printArr(int a[], int n) /* function to print the array */
21 {
22     int i;
23     for (i = 0; i < n; i++)
24         printf("%d ", a[i]);
25 }
26
27 int main()
28 {
29     int a[] = { 12, 31, 25, 8, 32, 17 };
30     int n = sizeof(a) / sizeof(a[0]);
31     printf("Before sorting array elements are - \n");
32     printArr(a, n);
33     selection(a, n);
34     printf("\nAfter sorting array elements are - \n");
35     printArr(a, n);
36     return 0;
37 }
```

### 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-**

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<ctype.h>
4  #include<string.h>
5
6  #define SIZE 100
7
8  char stack[SIZE];
9  int top = -1;
10
11 void push(char item){
12     if(top >= SIZE-1){
13         printf("\nStack Overflow.");
14     }
15     else{
16         top = top+1;
17         stack[top] = item;
18     }
19 }
20
21 char pop(){
22     char item ;
23
24     if(top <0){
25         printf("stack under flow: invalid infix expression");
26         getchar();
27         exit(1);
28     }
29     else{
30         item = stack[top];
31         top = top-1;
32         return(item);
33     }
34 }
35
36 int is_operator(char symbol){
37     if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
38     {
39         return 1;
40     }
41     else{
42         return 0;
43     }
44 }
45
46 int precedence(char symbol){
47     if(symbol == '^'){
48         return(3);
49     }
50     else if(symbol == '*' || symbol == '/')
51     {
52         return(2);
53     }
54     else if(symbol == '+' || symbol == '-')
55     {
56         return(1);
57     }
58     else{
59         return(0);
60     }
61 }
62
63 void InfixToPostfix(char infix_exp[], char postfix_exp[])
64 {
65     int i, j;
66     char item;
67     char x;
68

```



```

69     push('(');
70     strcat(infix_exp, "(");
71
72     i=0;
73     j=0;
74     item=infix_exp[i];
75
76     while(item != '\0')
77     {
78         if(item == '('){
79             push(item);
80         }
81         else if( isdigit(item) || isalpha(item))
82         {
83             postfix_exp[j] = item;
84             j++;
85         }
86         else if(is_operator(item) == 1)
87         {
88             x=pop();
89             while(is_operator(x) == 1 && precedence(x)>= precedence(item))
90             {
91                 postfix_exp[j] = x;
92                 j++;
93                 x = pop();
94             }
95             push(x);
96
97             push(item);
98         }
99         else if(item == ')')
100        {
101            x = pop();
102            while(x != '(')
103            {
104                postfix_exp[j] = x;
105                j++;
106                x = pop();
107            }
108        }
109        else{
110            printf("\nInvalid infix Expression.\n");
111            getchar();
112            exit(1);
113        }
114        i++;
115        item = infix_exp[i];
116    }
117    if(top>0){
118        printf("\nInvalid infix Expression.\n");
119        getchar();
120        exit(1);
121    }
122    if(top>0){
123        printf("\nInvalid infix Expression.\n");
124        getchar();
125        exit(1);
126    }
127    postfix_exp[j] = '\0';
128 }
129
130 int main(){
131     char infix[SIZE], postfix[SIZE];
132     printf("\nEnter Infix expression : ");
133     gets(infix);
134
135     InfixToPostfix(infix,postfix);
136     printf("Postfix Expression: ");
137     puts(postfix);
138
139     return 0;
140 }

```

## Output-

```

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

```