1.Implement a Program in C for converting an Infix Expression to Postfix Expression

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

//Global Variables
#define SIZE 20
char stack[SIZE];
int top;

//Adds a character (like +, *, etc.) onto the stack.
void push(char symbol)
{
   stack[++top] = symbol;
}

//Removes and returns the top character from the stack.
char pop()
{
   return stack[top--];
}

int priority(char op)
{
   switch (op)
   {
      case '#': return 0;
      case '(': return 1;
      case '+':
      case '-': return 2;
      case '*':
      case '/': return 3;
   }
}
void infixtopostfix(char *infix, char *postfix)
{
   char symbol, brace;
   int i = 0, k = 0;
   push('#');
   while((symbol = infix[i++])!='\0')
   {
      if(isalnum(symbol))
         postfix[k++] = symbol;
      else if(symbol == '(')
         push(symbol);
      else if(symbol == ')')
      {
         while(stack[top] != '(')
            postfix[k++] = pop();
```

```
            brace = pop();
        }
        else
        {
            while (priority(stack[top]) >= priority(symbol))
                postfix[k++] = pop();// pop operators with higher/equal priority
            push(symbol);   // push the current operator
        }
    }
    while(stack[top] != '#')
        postfix[k++] = pop();
    postfix[k] = '\0';
}

void main()
{
    char infix[50],postfix[50];
    top=-1;
    printf("\nInput the Infix Expression: ");
    scanf("%s",infix);
    infixtopostfix(infix,postfix);
    printf("Postfix Expression: %s",postfix);
}
```

OUTPUT

Input the Infix Expression: 8+7*2
Postfix Expression: 872*+

2.Design, develop, and execute a program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are + (add), - (subtract), * (multiply) and / (divide).

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

#define MAX 20

float  stack[MAX];
int top;

void push(float result)
{
  stack[++top] = result;
}

float pop()
{
  return stack[top--];
}

float getResult(float op1, float op2, char symbol)
{
  float result;
  switch (symbol)
  {
    case '+': result = op1 + op2;
          break;
    case '-': result = op1 - op2;
          break;
    case '*': result = op1 * op2;
          break;
    case '/':   if(op2 != 0)
                  result = op1 / op2;
              else
              {
                printf("\nException: Dived by zero");
                exit(0);
              }
              break;
  }
  return (result);
}

float evalPost(char *postfix)
```

```
{
  int i=0;
  char symbol;
  float op1,op2,result;
  top = -1;
  while((symbol=postfix[i++]) != '\0')
  {
    if(isdigit(symbol))
    {
      result = symbol - '0';
      push(result);
    }
    else
    {
      op2 = pop();
      op1 = pop();
      result = getResult(op1,op2,symbol);
      push(result);
    }
  }
  return pop();
}

void main()
{
  char postfix[MAX];
  float result;
  printf("\nEnter a valid postfix expression: ");
  scanf("%s",&postfix);
  result = evalPost(postfix);
  printf("\nThe result is %f",result);
}
```

OUTPUT:

```
Enter a valid postfix expression: 45*9/8*
The result is 17.777779
```

3.Design, develop, and execute a program in C to simulate the working of a queue of integers using an array. Provide the following operations: a. Insert b. Delete c. Display

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3
int queue[MAX];
int front = -1,rear = -1;

void enqueue()
{
   int item;
   printf("Enter a value: ");
   scanf("%d",&item);
   if(rear == MAX-1)
   {
      printf("\n ** Overflow **\n");
      return;
   }
   if(front==-1)
      front++;
   queue[++rear] = item;
   //if(rear == MAX-1)
     // printf("\n** Full **\n");
}

void dequeue()
{
   int item;
   if(front==-1)
   {
      printf("\n** Underflow **\n");
      return;
   }
   item = queue[front];
   if(front==rear)
   {
      front = rear = -1;
      printf("**Empty**\n");
   }
   else
      front++;
   printf("\nDeleted elemet is %d \n",item);
}

void display()
{
```

```
    int i;
    if(front==-1)
    {
        printf("\n**Empty..**\n");
        return;
    }
    for(i=front; i<=rear; i++)
        printf("%d ",queue[i]);
}

void main()
{
    int choice,item;
    printf("\nQueue Implementation");
    printf("\n1.Insert.\n2.Delete.\n3.Display.\n4.Exit");
    while(1)
    {
        printf("\nEnter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: enqueue();
                    break;
            case 2: dequeue();
                    break;
            case 3: printf("\nThe Contents of the Queue are: \n");
                    display();
                    break;
            default: exit(0);
        }
    }
}
```

OUTPUT:

```
Queue Implementation
1.Insert.
2.Delete.
3.Display.
4.Exit
Enter your choice: 1
Enter a value: 12

Enter your choice: 1
Enter a value: 14

Enter your choice: 3

The Contents of the Queue are:
12 14
```

Enter your choice: 2

Deleted elemet is 12

Enter your choice: 3

The Contents of the Queue are:
14
Enter your choice: 4

4.Write a C program to simulate the working of a singly linked list providing the following operations: a. Display & Insert b. Delete from the beginning/end c. Delete a given element

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

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

NODE *start = NULL;

NODE *getnode()
{
   NODE *temp;
   temp = (NODE *) malloc(sizeof(NODE));
   if(temp==NULL)
   {
      printf("\nNODE not created\n");
     // return;
      exit(0);
   }
   return temp;
}
//Insert front
void insertfront()
{
   NODE *newnode = getnode();
   printf("Enter a node data: ");
   scanf("%d",&newnode->data);
   newnode->link = start;
   start = newnode;
}
//Delete front
void deletefront()
{
   NODE *temp=start;
   if(start==NULL)
   {
      printf("\nList Empty\n");
      return;
   }
   start=start->link;
   printf("\nDeleted Node data is %d",temp->data);
   free(temp);
}
```

```
//Insert rear
void insertrear()
{
   NODE *newnode,*prev;
   newnode=getnode();
   printf("Enter a node data: ");
   scanf("%d",&newnode->data);
   newnode->link=NULL;
   if(start==NULL)
   {
      start=newnode;
      return;
   }
   prev=start;
   while(prev->link!=NULL)
   {
      prev=prev->link;
   }
   prev->link=newnode;
}
//Delete rear
void deleterear()
{
   NODE *prev,*cur;
   if(start==NULL)
   {
      printf("\nList Empty\n");
      return;
   }
   prev=cur=start;
   while(cur->link!=NULL)
   {
      prev=cur;
      cur=cur->link;
   }
   prev->link=NULL;
   printf("\nDeleted Node data is %d",cur->data);
   free(cur);
}
//Delete rear
void deleteelement()
{
   int item;
   NODE *prev, *cur;
   printf("Enter a data node to be delete: ");
   scanf("%d", &item);
   if (start == NULL)
   {
      printf("\nList Empty\n");
      return;
```

```
    }
    prev = NULL;
    cur = start;
    while (cur != NULL)
    {
        if (cur->data == item)
        {
            if (cur == start)
            {
                start = cur->link;
            }
            else
            {
                prev->link = cur->link;
            }
            printf("\nDeleted Node data is %d", cur->data);
            free(cur);
            return;
        }
        prev = cur;
        cur = cur->link;
    }
    printf("Element not found in the list.\n");
}
//Display
void display()
{
    NODE *temp=start;
    if(start==NULL)
    {
        printf("\nList Empty\n");
        return;
    }
    printf("\nLinked list elements are: ");
    while(temp!=NULL)
    {
        printf("%d ",temp->data);
        temp=temp->link;
    }
}

void main()
{
    int ch;
    printf("\nLinked List Implementation\n");
    printf("1.Insert Front\n2.Insert Rear\n3.Delete Front\n4.Delete Rear\n5.Delete
Given Element\n6.Display\n");
    while(1)
    {
        printf("\nEnter a choice: ");
```

```
        scanf("%d",&ch);
        switch(ch)
        {
           case 1: insertfront();
                   break;
           case 2: insertrear();
                   break;
           case 3: deletefront();
                   break;
           case 4: deleterear();
                   break;
           case 5: deleteelement();
                   break;
           case 6: display();
                   break;
        }
    }
}
```

OUTPUT:

```
Linked List Implementation
1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Delete Given Element
6.Display

Enter a choice: 1
Enter a node data: 10

Enter a choice: 2
Enter a node data: 30

Enter a choice: 1
Enter a node data: 5

Enter a choice: 6
Linked list elements are: 5 10 30

Enter a choice: 5
Enter a data node to be delete: 10
Deleted Node data is 10

Enter a choice: 6
Linked list elements are: 5 30

Enter a choice: 4
Deleted Node data is 30
```

Enter a choice: 6
Linked list elements are: 5

Enter a choice: 3
Deleted Node data is 5

Enter a choice: 6
List Empty

Enter a choice: 5
Enter a data node to be delete: 99
Element not found in the list.

5.Write a C program to Implement the following searching techniques a. Linear Search b. Binary Search.

```
#include <stdio.h>

int linearSearch(int *a, int n, int key)
{
   int i = 0;
   while (i < n)
   {
      if (a[i] == key)
         return i + 1;
      i++;
   }
   return -1;
}

int binarySearch(int a[], int left, int right, int key)
{
   if (left > right)
      return -1;
   int mid = (left + right) / 2;
   if (a[mid] == key)
      return mid + 1;
   else if (a[mid] > key)
      binarySearch(a, left, mid - 1, key);
   else
      binarySearch(a, mid + 1, right, key);
}

void main()
{
   int choice, arr[10], n, key, index;
   printf(" Enter the size of the array : ");
   scanf("%d", &n);
   printf(" Select from Menu : \n 1. Linear Search. \n 2. Binary Search. ");
   printf("\n Enter Your Choice : ");
   scanf("%d", &choice);
   if (choice == 1)
      printf("\n Enter the array Elements: \n");
   else
      printf("\n Enter sorted elements for Binary Search : ");
   for (int i = 0; i < n; i++)
      scanf("%d", &arr[i]);
   printf("\n Enter the Key Element: ");
   scanf("%d", &key);
   switch (choice)
   {
   case 1:
```

```
        index = linearSearch(arr, n, key);
        break;
    case 2:
        index = binarySearch(arr, 0, n - 1, key);
        break;
    default:
        printf(" ***** Wrong Choice Entered ******: \n");
        break;
    }
    if (index == -1)
        printf("\n Element Not found. ");
    else
        printf("\n Element found at location %d \n", index);
}
```

OUTPUT:

```
Enter the size of the array : 5
 Select from Menu :
 1. Linear Search.
 2. Binary Search.
 Enter Your Choice : 1

 Enter the array Elements:
4 5 2 1 8

 Enter the Key Element: 1

 Element found at location 4


Enter the size of the array : 5
 Select from Menu :
 1. Linear Search.
 2. Binary Search.
 Enter Your Choice : 2

 Enter sorted elements for Binary Search : 1 2 3 4 5

 Enter the Key Element: 3

 Element found at location 3
```

6.Write a C program to implement the following sorting algorithms using user defined functions: a. Bubble sort (Ascending order) b. Selection sort (Descending order).

```
#include <stdio.h>

void swap(int *x, int *y)
{
   int temp = *x;
   *x = *y;
   *y = temp;
}

void bubbleSort(int *arr, int n)
{
   int i, j;
   for (i = 0; i < n - 1; i++) // Runs n-1 passes
   {
      for (j = 0; j < (n - 1) - i; j++) // bubble smallest element to left
         if (arr[j + 1] < arr[j])
            swap(&arr[j], &arr[j + 1]);
   }
   printf("END bubble Sort");
}

void selectionSort(int *arr, int n)
{
   int i, j, min;
   for (i = 0; i < n - 1; i++) // Runs n-1 passes
   {
      min = i;
      for (j = i + 1; j < n; j++) // finds min element
      {
         if (arr[j] > arr[min])
            min = j;
      }
      swap(&arr[i], &arr[min]); // swap min with leftmost element
   }
}

void main()
{
   int choice, arr[10], n, index, i;
   printf("Enter the size of the array : \n");
   scanf("%d", &n);
   printf("Enter array Elements : \n");
   for (i = 0; i < n; i++)
      scanf("%d", &arr[i]);
   printf("Select from Menu : \n 1. Bubble Sort. \n 2. Selection Sort. \n");
```

```
    printf("Enter Your Choice : ");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
        bubbleSort(arr, n);
        break;
    case 2:
        selectionSort(arr, n);
        break;
    default:
        printf("***** Wrong Key Entered ******: \n");
    }
    printf("\nArray Elements after Sorting: \n");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
}
```

OUTPUT:

```
Enter the size of the array :
5
Enter array Elements :
4 5 1 2 3
Select from Menu :
 1. Bubble Sort.
 2. Selection Sort.
Enter Your Choice : 1
END bubble Sort
Array Elements after Sorting:
1 2 3 4 5

Enter the size of the array :
5
Enter array Elements :
4 5 1 2 3
Select from Menu :
 1. Bubble Sort.
 2. Selection Sort.
Enter Your Choice : 2

Array Elements after Sorting:
5 4 3 2 1
```

7.Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm (C programming)

```
#include <stdio.h>
#define INF 999

int main() {
   int n, i, j, u, v, a, b;
   int ne = 1, min_cost = 0;
   int cost[20][20], parent[20] = {0};

   printf("Enter the number of vertices: ");
   scanf("%d", &n);

   printf("Enter the cost matrix:\n");
   for (i = 1; i <= n; i++)
      for (j = 1; j <= n; j++)
         scanf("%d", &cost[i][j]);

   for (i = 1; i <= n; i++)
      parent[i] = 0;

   printf("\nThe edges of the Minimum Spanning Tree are:\n");
   while (ne < n) {
      int min = INF;
      for (i = 1; i <= n; i++) {
         for (j = 1; j <= n; j++) {
            if (cost[i][j] < min) {
               min = cost[i][j];
               a = u = i;
               b = v = j;
            }
         }
      }

      while (parent[u]) u = parent[u];
      while (parent[v]) v = parent[v];

      if (u != v) {
         printf("Edge %d:\t(%d -> %d) = %d\n", ne++, a, b, min);
         min_cost += min;
         parent[v] = u;
      }

      cost[a][b] = cost[b][a] = INF;

      // Safety check: if no valid edge is found
      if (min == INF) {
         printf("Graph is disconnected. Spanning tree not possible.\n");
```

```
            return 1;
        }
    }

    printf("\nMinimum cost = %d\n", min_cost);
    return 0;
}
```

OUTPUT

```
Enter the number of vertices: 4
Enter the cost matrix:
0 1 3 0
1 0 1 6
3 1 0 2
0 6 2 0

The edges of the Minimum Spanning Tree are:
Edge 1:         (1 -> 4) = 0
Edge 2:         (1 -> 2) = 1
Edge 3:         (2 -> 3) = 1

Minimum cost = 2
```

8.From a given vertex in a weighted connected graph, find shortest paths to other vertices Using Dijkstra's algorithm (C programming)

```
#include <stdio.h>
#define INF 999

void dijkstra(int c[10][10], int n, int s, int d[10]) {
   int v[10], min, u, i, j;
   for (i = 1; i <= n; i++) {
      d[i] = c[s][i];
      v[i] = 0;
   }

   v[s] = 1;

   for (i = 1; i < n; i++) {
      min = INF;
      u = -1;
      for (j = 1; j <= n; j++) {
         if (v[j] == 0 && d[j] < min) {
            min = d[j];
            u = j;
         }
      }

      if (u == -1) // No more reachable nodes
         break;

      v[u] = 1;

      for (j = 1; j <= n; j++) {
         if (v[j] == 0 && (d[u] + c[u][j]) < d[j]) {
            d[j] = d[u] + c[u][j];
         }
      }
   }
}

int main() {
   int c[10][10], d[10], i, j, s, n;

   printf("Enter number of vertices: ");
   scanf("%d", &n);

   printf("Enter the cost matrix (0 if no edge):\n");
   for (i = 1; i <= n; i++) {
```

```
        for (j = 1; j <= n; j++) {
            scanf("%d", &c[i][j]);
            if (i != j && c[i][j] == 0)
                c[i][j] = INF;
        }
    }

    printf("Enter the source node (1 to %d): ", n);
    scanf("%d", &s);

    dijkstra(c, n, s, d);

    printf("\nShortest distances from node %d:\n", s);
    for (i = 1; i <= n; i++) {
        if (d[i] == INF)
            printf("Node %d is unreachable\n", i);
        else
            printf("To node %d: %d\n", i, d[i]);
    }

    return 0;
}
```

OUTPUT:

```
Enter number of vertices: 4
Enter the cost matrix (0 if no edge):
0 5 0 10
5 0 3 0
0 3 0 1
10 0 1 0
Enter the source node (1 to 4): 1

Shortest distances from node 1:
To node 1: 0
To node 2: 5
To node 3: 8
To node 4: 9
```