1+16/19

Write a C Program to Search an Element using linear Search

```
#include<stdio.h>
main()
{
    int a[30],i,n,key,flag=0;
    Printf("Enter no of elements");
    Scanf("%d",&n);
    Printf("Enter %d elements",n);
    for(i=0;i<n;i++)
        Scanf("%d",&a[i]);
    Printf("Enter Key element");
    Scanf("%d",&key);
    for(i=0;i<n;i++)
    {
        if(a[i]==key)
        {
            Printf("element found at %d position",i);
            flag=1;
            break;
        }
    }
    if(flag==0)
        Printf("Element not found");
}
```

Output:-

Enter no of elements 5

Enter 5 elements 1 5 3 2 0

Enter Key element 4

Element not found.

Write a C-program to Search an element using Binary Search

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a[20], i, n, key, first, last, mid;
    Printf ("Enter n value");
    Scanf ("%d", &n);
    Printf ("Enter %d elements", n);
    For (i=0; i<n; i++)
        Scanf ("%d", &a[i]);
    Printf ("Enter key element");
    Scanf ("%d", &key);
    first = 0;
    last = n-1;
    while (first<=last)
    {
        mid = (first+last)/2;
        if (a(mid) < key)
            first = mid+1;
        else if (a[mid] == key)
        {
            Printf ("Element found at %d", mid);
            break;
        }
        else
            last = mid-1;
    }
    if (first > last)
        Printf ("Element Not found");
}
```

output:

Enter n value 5

Enter 5 Elements 01 0 2 4 3 4

Enter key element 4

Element found at 4

24/6/18

Write a C-Program to Implement Bubble Sorting.

```
#include<stdio.h>
main()
{
    int n,i,j,t,a[20];
    Printf("enter no of elements");
    Scanf("%d",&n);
    Printf(" enter %d elements",n);
    for(i=0;i<n;i++)
        Scanf("%d",&a[i]);
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(a[j]<a[j+1])
            {
                t=a[j];
                a[j]=a[j+1];
                a[j+1]=t;
            }
        }
    }
    Printf("Sorted order is");
    for(i=0;i<n;i++)
        Printf("%d",a[i]);
}
```

Output :-

Enter no of elements 5

Enter 5 elements 20 5 3 10 1

Sorted order is 1 3 5 10 20

Write a C- Program to Implement Selection Sorting

```
#include<Stdio.h>
main()
{
    int  a[20], n, i, key, min;
    Printf (" Enter no of elements");
    Scanf ("%d", &n);
    Printf(" Enter %d elements ", n);
    for (i=0; i<n ; i++)
       Scanf ("%d", &a[i]);
    for (i=0; i<n; i++)
    {
        min = i;
        for (j=i+1; j<n; j++)
        {
            if (a[j] < a[min])
            {
               min=j;
            }
        }
        t= a[i];
        a[i] = a[min];
        a[min] = t;
    }
    Printf (" Sorted order is");
    for (i=0; i<n; i++)
       Printf (" %d", a[i]);
}
```

output:

```
Enter no of elements 5
Enter 5 elements 6 5 3 1 2
Sorted order is 1 2 3 5 6
```

Write a C-Program to Implement Insertion Sorting

```
#include<Stdio.h>
main()
{
    int a[20], n, i, j, t;
    Printf(" Enter no of elements");
    Scanf (" %d", &n);
    Printf(" Enter %d elements", n);
    for(i=0; i<n; i++)
        Scanf(" %d", &a[i]);
    for (i=1; i<n; i++)
    {
        t= a[i];
        j = i-1;
        While (j>=0 && a[j]>t)
        {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1]= t;
    }
    Printf (" Sorted order is");
    for(i=0; i<n; i++)
        Printf (" %d", a[i]);
}
```

Output:- Enter no of elements 5

Enter 5 elements 1 4 2 5 3

Sorted order is 1 2 3 4 5

13/8/18

Write a C- Program to Implement merge Sort

```
#include<stdio.h>
#include<conio.h>
void mergesort (int [],int ,int);
void merge (int [],int, int, int);
void main()
{
    int a[10], n, i;
    Printf ("enter n value");
    Scanf ("%d", &n);
    Printf ("enter %d element ", n);
    for (i=0; i<n; i++)
        Scanf ("%d", &a[i]);
    Mergesort (a, 0, n-1);
    Printf ("After Sorting");
    for (i=0; i<n; i++)
        Printf ("%d", a[i]);
}
Void mergesort ( int a[20], int beg, int end)
{
    int mid;
    if (beg <end)
    {
        mid = (beg + end)/2;
        mergesort (a, beg, mid);
        mergesort (a, mid+1, end);
        merge (a, beg, mid, end);
    }
}
```

```
void merge (int a[], int beg, int mid, int end)
{
    int l=beg, R=mid+1, i=beg, t[20];
    while (l<=mid && R<=end)
    {
        if (a[l]<a[R])
        {
            t[i]=a[l];
            l++;
        }
        else
        {
            t[i]=a[R];
            R++;
        }
        i++;
    }
    while (l<=mid)
    {
        t[i]=a[l];
        l++;
        i++;
    }
    while (R<=end)
    {
        t[i]=a[R];
        R++;
        i++;
    }
    for (i=beg; i<end; i++)
    {
        a[i]=t[i];
    }
}
```

Output:

```
Enter n value 6
Enter 6 elements 0 1 5 3 6 4
After Sorting 0 1 3 4 5 6
```

Write a C-program to Implement Quick Sort.

```
#include <stdio.h>
#include <conio.h>
Void qs (int [], int, int);
int Part (int [], int, int);
Void main ()
{
    int a[20], n, i;
    ClrScr ();
    Printf ("Enter n value");
    Scanf ("%d", &n);
    Printf ("Enter %d Elements", n);
    for (i=0; i<n; i++)
        Scanf ("%d", &a[i]);
    qs (a, 0, n-1);
    for (i=0; i<n; i++)
        Printf ("%d", a[i]);
    getch ();
}
Void qs (int a[], int low, int high)
{
    int pi;
    if (low < high)
    {
        pi = Part (a, low, high);
        qs (a, low, pi-1);
        qs (a, pi+1, high);
    }
}
int Part (int a[], int low, int high)
{
    int p, pi = a[high], c, i = low, t=0;
    for (c=low; c<=high; c++)
    {
        if (a[c] <= pi)
        {
            t = a[i];
```

```
        a[c] = t;
        i++;
    }
  }
  t = a[i];
  a[i] = a[high];
  a[high] = t;
  return i;
}
```

Output:

Enter n value 7

Enter 7 elements 6 5 0 1 2 3 4

After Sorting 0 1 2 3 4 5 6.

15/7/19

* Write a C-program to perform hashing

```
#include<stdio.h>
#define n 10
int a[n];
void insert()
{
    int key,h,i,index;
    Printf("Enter Key element");
    Scanf("%d",&key);
    h = key%n;
    for (i=0;i<n;i++)
    {
       index = (h+i)%n;
       if (a[index]==0)
       {
          a[index] = key;
          Printf("Element Inserted");
          break;
       }
       if (i==n)
          Printf("Element not inserted");
}
void Search()
{
    int key, index, h,i;
    Printf("Enter Key element");
    Scanf("%d",&key);
    h = key%n;
    for (i=0; i<n; i++)
    {
       index = (h+i)%n;
       if (a[index]==key)
       {
          Printf("Element found at %d position", index);
          break;
       }
    }
    if (i==n)
    {
       Printf("Element not found");
    }
}
```

```
Void display()
{
   int i;
   For (i=0; i<n; i++)
      Printf ("%d \t", a[i]);
}
Void main()
{
   int opt;
   While (1)
   {
      PrintF ("\n Enter option from menu \n1. Insertion \n2. Display \n3. Search
        \n4. Exit");
      Scanf ("%d", &opt);
      Switch (opt)
      {
         Case 1: insert();
                 break;
         Case 2: display();
                 break;
         Case 3: Search();
                 break;
         Case 4: Exit (0);
      }
   }
}
```

Output :-

Enter option from menu
1. Insertion
2. Display
3. Search
4. Exit 1

Enter Key 10 20

Enter option from menu
1. Insertion
2. Display
3. Search
4. Exit 2

10 20

Enter option from menu
1. Insertion
2. Display
3. Search
4. Exit 3

Enter Key-element 40

Element not found

Enter option from menu
1. Insertion
2. Display
3. Search
4. Exit 4.

* Write a C- Program to Perform Single linked list Operation

```
#include <stdio.h>
#include <stdlib.h>
Struct node
{
    int data;
    Struct node *link;
};
Struct node *root = NULL;
Void insert()
{
    Struct node *temp;
    temp = (Struct node *)malloc(sizeof(Struct node));
    Printf("Enter data");
    Scanf("%d", &temp->data);
    if (root == NULL)
        root = temp;
    Else
    {
        Struct node *p;
        P = root;
        While (P->link != NULL)
        {
            P = P->link;
        }
        P->link = temp;
    }
}
int counting();
{
    Struct node *P;
    int count = 0;
    P = root;
    if (root == NULL)
        Printf("list is Empty");
    Else
    {
        While (P != NULL)
        {
            Count++;
            P = P->link;
        }
    }
    return count;
}
```

```
Void insertafter()
{
   Struct node *P,*q;
   int loc, len, i=1;
   Printf("Enter location");
   Scanf("%d", &loc);
   len = Counting();
   if (loc>len)
      Printf("Invalid");
   Else
   {
      P= root;
      While (i<loc)
      {
         P= P->link;
         i++;
      }
   }
   temp = (Struct node *) malloc (Sizeof (Struct node));
   Printf("Enter data");
   Scanf("%d", &temp->data);
   temp -> link = P-> link;
   P-> link = temp;
}

Void display()
{
   Struct node *temp;
   temp = root;
   if (root == NULL)
      Printf("list is Empty");
   Else
   {
      While (temp != NULL)
      {
         Printf("%d", temp->data);
         temp = temp -> link;
      }
   }
}
```

```
Void delete()
{
    Struct node *temp;
    int loc, len;
    Printf("Enter location");
    Scanf("%d", &loc);
    len = Counting();
    if (len<loc)
      Printf(" Deletion is not possible");
    Else if (loc==1)
    {
        temp = root;
        root = temp->link;
        temp->link = NULL;
        free(temp);
    }
    Else
    {
        Struct node *p, *q;
        int i=1;
        P= root;
        while (i<loc-1)
        {
          P= P->link;
          i++;
        }
        q= P->link;
        P->link = q->link
        q->link = NULL;
        free(q);
    }
}
Void main()
{
    int opt, c;
    While (1)
    {
      Printf(" \n Enter Choice from the menu \n 1. insert \n 2. Count \n 3. insert
      after \n 4 : display \n 5. delete \n 6. Exit ");
      Scanf("%d", &opt);
```

```
        Case 1: insert();
                break;
        Case 2: c = Counting();
                Printf("Count = %d", c);
                break;
        Case 3: insertafter();
                break;
        Case 4: display();
                break;
        Case 5: delete();
                break;
        Case 6: exit(0);
        default : Printf("Invalid");
    }
  }
}
```

Output :-

Enter option from menu
1. Insert
2. Count
3. Insertafter
4. display
5. delete
6. Exit 1

Enter data 10 20

Enter option from menu
1. Insert
2. Count
3. Insert after
4. display
5. delete
6. Exit 2

Count = 2

Enter option from menu
1. Insert
2. Count
3. Insert after
4. display
5. delete
6. Exit 4

10 20

Enter option from menu
1. Insert
2. Count
3. Insert after
4. display
5. delete
6. Exit 6.

30/7/19

* Write a C-Program to perform double linked list Operation.

```
#include<Stdio.h>
Struct Node
{
    int data;
    Struct node *left;
    Struct node *right;
};
Struct Node *root = NULL;
Void insert()
{
    Struct Node *temp;
    temp = (Struct node *)malloc (Sizeof(Struct Node));
    Printf("Enter data");
    Scanf("%d",& temp->data);
    temp -> left = NULL;
    temp -> right = NULL;
    if (root == NULL)
        root=temp;
    Else
    {
        Struct Node *P;
        P = root;
        While (P-> right != NULL)
        {
            P = P-> right;
        }
        P-> right = temp;
        temp -> left =P;
    }
}
int Counting()
{
    Struct node *P;
    int Count=0;
    if (root == NULL)
        Printf(" list is Empty");
    Else
    {
        P=root;
```

```
    while (P != NULL)
    {
        P = P -> link;
        count++;
    }
  }
  return count;
}
void display()
{
    struct node *temp;
    temp = root;
    if (root == NULL)
        Printf ("list is Empty");
    else
    {
        while (temp != NULL)
        {
            Printf("%d", temp -> data);
            temp = temp -> right;
        }
    }
}
Void insertafter()
{
    struct node *temp, *P;
    int loc, len, i = 1;
    Printf ("Enter location");
    Scanf ("%d", &loc);
    len = Counting();
    if (loc < len)
        Printf("Invalid");
    else
    {
        P = root;
        while (i < loc)
        {
            P = P -> right;
            i++;
        }
    }
    temp = (struct node *) malloc (sizeof (struct node));
```

```
    Printf("Enter data");
    Scanf("%d", &temp->data);
    temp->right = P->right;
    P->right->left = temp;
    temp->left = P;
    P->right = temp;
}
Void delete()
{
    Struct node *temp;
    int len, loc;
    Printf("Enter location");
    Scanf("%d", &loc);
    len = Counting();
    if (len > loc)
        Printf("Deletion is not possible");
    else if(loc == 1)
    {
        temp = root;
        root = temp->right;
        temp->link = NULL;
        temp->right = NULL;
        temp->right->left = NULL;
        free(temp);
    }
    else
    {
        Struct node *P, *Q;
        int i = 1;
        P = root;
        while (i<loc-1)
        {
            P = P->right;
            i++;
        }
        Q = P->right;
        q->right->left = P;
        P->right = q->right;
```

```
    q -> right = NULL;
    q -> left = NULL;
    free(q);
  }
}
void main()
{
    int c, ch;
    while(1)
    {
      Printf("\n Enter Choice \n1. insert \n2. Count \n3. Insert after
          \n4. delete \n5. display \n6. Exit ");
      Switch(ch)
      {
        Case 1: insert();
                break;
        Case 2: c = Counting();
                Printf("Count = %d", c);
                break;
        Case 3: InsertAfter();
                break;
        Case 4: delete();
                break;
        Case 5: display();
                break;
        Case 6: Exit(0);
      }
    }
}
```

Output :-

Enter Choice	Enter Choice	Enter Choice
1. Insert	1. Insert	1. Insert
2. Count	2. Count	2. Count
3. Insertafter	3. Insertafter	3. Insertafter
4. delete	4. delete	4. delete
5. display	5. display	5. display
6. Exit 1	6. Exit 2	6. Exit 5
Enter data 10 20 30	Count = 3	~~Enter data 25~~
		10 20 30

*Develop a C-program to perform addition of two polynomials using array.

```
#include<stdio.h>
Struct poly
{
    int coeff;
    int expn;
};
Struct poly P1[10], P2[10], P3[10];
void readpoly(Struct poly P[], int);
int addpoly(Struct poly P1[], Struct poly P2[], int, int);
void display(Struct poly P[], int);
Void main()
{
    int t1, t2, t3;
    Printf("\n Enter no of terms in polynomial 1");
    Scanf("%d", &t1);
    Read poly(P1, t1);
    Printf("\n Enter no of terms in polynomial 2");
    Scanf("%d", &t2);
    Read poly(P2, t2);
    Printf("\n polynomial 1 representation is");
    display(P1, t1);
    Printf("\n polynomial 2 representation is");
    display(P2, t2);
    t3 = addpoly(P1, P2, t1, t2);
    Printf("\n addition of two polynomials is");
    display(P3, t3);
}
Void readpoly(Struct poly P[], int t)
{
    int i;
    for(i=0; i<t; i++)
    {
        Printf("\n Enter Coefficient %d", i+1);
        Scanf("%d", &P[i].coeff);
        Printf("\n Enter Exponent %d", i+1);
        Scanf("%d", &P[i].expn);
    }
}
```

```
int addPoly (struct Poly P1[], struct Poly P2[], int t1, int t2)
{
    int i=0, j=0, k=0;
    while (i<t1 && j<t2)
    {
        if (P1[i].Expn > P2[j].Expn)
        {
            P3[k].Coeff = P1[i].Coeff;
            P3[k].Expn = P1[i].Expn;
            i++;
            k++;
        }
        else if (P1[i].Expn < P2[j].Expn)
        {
            P3[k].Coeff = P2[j].Coeff;
            P3[k].Expn = P2[j].Expn;
            j++;
            k++;
        }
        else
        {
            P3[k].Coeff = P1[i].Coeff + P2[j].Coeff;
            P3[k].Expn = P1[i].Expn;
            i++;
            j++;
            k++;
        }
    }
    while (i<t1)
    {
        P3[k].Coeff = P1[i].Coeff;
        P3[k].Expn = P1[i].Expn;
        i++; k++;
    }
    while (j<t2)
    {
        P3[k].Coeff = P2[j].Coeff;
        P3[k].Expn = P2[j].Expn;
        j++; k++;
    }
    return k;
}
```

```
* Void display(Struct poly P[], int t)
{
    int i;
    for (i=0; i<t ; i++)
    {
        Printf ("%d x^%d ", P[i].Coeff, P[i].Expr);
        if (i<t-1)
            Printf ("+");
    }
}
```

Output :-

Enter no of terms in polynomial 1 3
Enter Coefficient 1 7
Enter Exponent 1 4
Enter Coefficient 2 5
Enter Exponent 2 2
Enter Coefficient 3 3
Enter Exponent 3 1

Enter no of terms in polynomial 2 3
Enter Coefficient 1 5
Enter Exponent 1 3
Enter Coefficient 2 3
Enter Exponent 2 1
Enter Coefficient 3 -8
Enter Exponent 3 0

Polynomial 1 representation is $7x^4 + 5x^2 + 3x^1$

Polynomial 2 representation is $5x^3 + 3x^1 + -8x^0$

Addition of two polynomials is

$7x^4 + 5x^3 + 5x^2 + 6x^1 - 8x^0$

3/7/19

Write a c-program to represent Stack using arrays.

```
#include<Stdio.h>
#define MAX 10
int Stacks[MAX], TOP=-1;
Void Push();
Void Pop();
Void Peek();
Void display();
Void main()
{
    int OPt;
    while(1)
    {
        Printf("Enter option \n1. Push \n2. Pop \n3. Peek \n4. display
            \n5. Exit ");
        Scanf("%d",&OPt);
        Switch(OPt)
        {
            Case 1: Push();
                    break;
            Case 2: PoP();
                    break;
            Case 3: Peek();
                    break;
            Case 4: display();
                    break;
            Case 5: Exit(0);
            default: Printf("\n Over flow");
        }
    }
}
Void Push()
{
    int val;
    if(TOP == MAX-1)
        Printf("Over flow");
    Else
    {
        Printf("Enter Element");
```

```
        scanf("%d", &val);
        TOP = TOP++;
        Stack[TOP] = val;
        Printf("Element inserted");
    }
}
Void pop()
{
    if (TOP == -1)
        Printf("underflow");
    else
    {
        Printf("The element deleted is %d", Stack[TOP]);
        TOP = TOP-1;
    }
}
Void peek()
{
    if (TOP == -1)
        Printf("underflow");
    else
        Printf("Top most Element is: %d", Stack[TOP]);
}
Void display()
{
    int i;
    if (TOP == -1)
        Printf("No Element found");
    else
    {
        for (i = TOP; i >= 0; i--)
            Printf("%d \t", Stack[i]);
    }
}
```

output :-

Enter Choice
1. Push
2. Pop
3. Peek
4. display
5. Exit 1

Enter Element 10 20 30

Enter Choice
1. Push
2. Pop
3. Peek
4. display
5. Exit 4

30 20 10

Enter Choice
1. Push
2. Pop
3. Peek
4. display
5. Exit 3

Top most Element is 30.

Enter Choice
1. Push
2. Pop
3. Peek
4. display
5. Exit 2

The Element deleted is 30

Enter Choice
1. Push
2. Pop
3. Peek
4. display
5. Exit 4

20 10

Enter Choice
1. Push
2. Pop
3. Peek
4. display
5. Exit 5.

5/8/19

* Write a C-Program to represent Queue using Single linked list.

```
#include<stdio.h>
#include<stdlib.h>
Struct node
{
    int data;
    Struct node *link;
};
Struct node *front = NULL, *rear = NULL;
Void insert()
{
    Struct node *temp;
    temp = (Struct node *) malloc (sizeof (Struct node));
    Printf("Enter data");
    Scanf ("%d", &temp->data);
    temp->link = NULL;
    if (rear = NULL || front == NULL)
       front = rear = temp;
    Else
    {
       rear->link = temp;
       rear = temp;
    }
}
Void delete()
{
    Struct node *P;
    if (front == NULL)
       Printf(" Queue is Empty");
    Else
    {
       Printf(" Deleted Element is %d";
       P= front;
       front = P->link;
       P->link = NULL;
       free (P);
    }
}
```

```
void display()
{
    Struct Node *P;
    P= Front;
    if (Front == NULL)
        Printf("Queue is Empty");
    Else
    {
        While (P!= NULL)
        {
            Printf ("%d", P->data);
            P= P-> link;
        }
    }
}
Void main()
{
    int OPt;
    While (1)
    {
        Printf(" \n Enter Choice \n 1. Insert \n 2. delete \n 3. display \n 4. Exit");
        Scanf ("%d", &OPt);
        Switch (OPt)
        {
            Case 1: Insert();
                    break;
            Case 2: delete();
                    break;
            Case 3: display();
                    break;
            Case 4: Exit (0);
        }
    }
}
```

* Write a C-Program to perform operations on circular queue using array

```
#include <stdio.h>
#define Size 5
int front = -1, rear = -1, q[size];
void insert()
{
    int val;
    Printf("Enter value");
    Scanf("%d", &val);
    if (front == rear+1 || Size == rear+1 && front == 0)
        Printf("Queue is Full");
    if (front == -1 && rear == -1)
    {
        front = rear = 0;
        q[rear] = val;
    }
    Else if (rear == Size-1)
    {
        rear = 0;
        q[rear] = val;
    }
    Else
    {
        rear+1 = 1;
        q[rear] = val;
    }
}
Void delete()
{
    if (front == -1 && rear == -1)
        Printf("Queue is Empty");
    if (front == Size-1)
    {
        Printf("Deleted Element is %d", q[front]);
        front = 0;
    }
    Else if (front == rear)
    {
        Printf("Deleted Element is %d", q[front]);
        front = rear = -1;
    }
```

```
    else
    {
        Printf ("Deleted Element is %d", q[front]);
        front += 1;
    }
}
void display()
{
    int i;
    if (front == -1 && rear == -1)
        Printf ("Queue is Empty");
    if (front <= rear)
    {
        for (i= front; i<= rear; i++)
            Printf ("%d", q[i]);
    }
    else
    {
        for (i = 0; i <= rear; i++)
            Printf ("%d", q[i]);
        for (i= front; i< size; i++)
            Printf ("%d", q[i]);
    }
}
void main()
{
    int opt;
    while (1)
    {
        Printf ("\nEnter Choice \n1.Insert \n2.delete \n3.display \n4.Exit");
        Scanf ("%d", &opt);
        Switch (opt)
        {
            case 1: Insert();
                    break;
            case 2: delete();
                    break;
            case 3: display();
                    break;
            case 4: Exit(0);
        }
    }
}
```

* Write a C-Program to Convert Infix Expression to Post Fix Expression

```
#include<stdio.h>
#include<ctype.h>
#define max 20
int top=-1;
void push(char ch)
{
    if(top==max-1)
        Printf("Stack is full");
    else
    {
        top+=1;
        Stack[top]=ch;
    }
}
Char pop()
{
    if(top==-1)
        Printf("Stack is Empty");
    else
        return Stack[top--];
}
int priority(Char ch)
{
    if(ch=='(')
        return 0;
    else if(ch=='+'||ch=='-')
        return 1;
    else if(ch=='*'||ch=='/'||ch=='%')
        return 2;
}
```

```
Void main()
{
    Char infix[20], *P, x;
    Printf("Enter Infix Expression");
    Scanf("%s", infix);
    P = &infix[0];
    While (*P != '\0')
    {
        if (*P == '(')
            Push(*P);
        Else if (isalnum(*P))
            Printf("%c", *P);
        Else if (*P == ')')
        {
            While ((x = Pop()) != '(')
            {
                Printf("%c", x);
            }
        }
        Else
        {
            While (Priority(*P) <= Priority(Stack[TOP]))
            {
                Printf("%c", Pop());
            }
            Push(*P);
        }
        P++;
    }
    While (TOP != -1)
    {
        Printf("%c", Pop());
    }
}
```

Output :-

Enter Infix Expression (A+B) * (C-D)

AB+ CD - *

Write a program to perform operations on Binary Search tree

```
#include<stdio.h>
#include<stdlib.h>
Struct node
{
    int data;
    Struct node *left,*right;
};
Void inorder(Struct node *root)
{
    if(root != NULL)
    {
        Inorder (root->left);
        Printf("%d", root->data);
        inorder (root->right);
    }
}
Void Preorder (Struct node *root)
{
    if(root != NULL)
    {
        Printf(" %d", root->data);
        Pre order (root -> left);
        Pre order (root ->right);
    }
}
Void Post order (Struct node * root)
{
    if (root != NULL)
    {
        Postorder (root->left);
        Post order (root->right);
        Print f ("%d", root->data);
    }
}
```

```
Struct node * insert (Struct node *root, int key)
{
    Struct node *temp = (Struct node *) malloc (sizeof (Struct node));
    temp -> data = key;
    temp -> left = temp -> right = NULL;
    if (root == NULL)
        root = temp;
    else
    {
        Struct node *curr, *parent;
        curr = *root;
        while (curr != NULL)
        {
            Parent = curr;
            if (temp -> data < curr -> data)
                curr = curr -> left;
            else if (temp -> data > curr -> data)
        }       curr = curr -> right;
    }
    if (temp -> data < parent -> data)
        parent -> left = temp;
    else
        parent -> right = temp;
}
return root;
}

Void Search (Struct node *root, int key)
{
    if (root == NULL)
        Printf ("Element not found");
    elseif (key < root -> data)
        Search (root -> left, key);
    else if (key > root -> data)
        Search (root -> right, key);
    else
        Printf (" Key found %d", key);
}
```

```
Struct node *min(Struct node *root)
{
    Struct node *Current = root;
    while(Current && Current->left != NULL)
        Current = Current->left;
    return Current;
}
Struct node *delete(Struct node *root, int key)
{
    if (root == NULL)
        return root;
    if (key < root->data)
        root->left = delete(root->left, key);
    else if (key > root->data)
        root->right = delete(root->right, key);
    else
    {
        if (root->left == NULL)
        {
            Struct node *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            Struct node *temp = root->left;
            free(root);
            return temp;
        }
        else
        {
            Struct node *temp = min(root->right);
            root->data = temp->data;
            root->right = delete(root->right, temp->data);
        }
    }
    return root;
}
```

```
void main()
{
    Struct node *root = NULL;
    int key, ch;
    while(1)
    {
      Printf("\n Enter Choice - - - ");
      Scanf("%d", &ch);
      Switch(ch)
      {
          case 1:
            Printf("Enter data");
            Scanf("%d", &key);
            root = insert(root, key);
            break;
          Case 2:
            Printf("Inorder Traversal of given tree \n");
            inorder(root);
            break;
          Case 3:
            Printf("Postorder Traversal of given tree \n");
            Postorder(root);
            break;
          Case 4:
            Printf("Pre order Traversal of given tree \n");
            Pre order(root);
            break;
          Case 5:
            Printf("Enter key");
            Scanf("%d", &key);
            Printf("Deleted node is %d", key);
            root = delete(root, key);
            if(root == NULL)
              Printf("Tree is Empty");
            else
            { Printf("Inorder of modified tree \n");
```

```
        inorder(root);
      }
      break;
    case 4: exit(0);
    }
  }
}
```

Output:

```
#include<stdio.h>

typedef struct node
{
        int data;
        struct node *left,*right;
        int ht;
}node;

node *insert(node *,int);
node *Delete(node *,int);
void preorder(node *);
void inorder(node *);
void search(node *,int);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int BF(node *);

int main()
{
        node *root=NULL;
        int x,n,i,op;

        do
        {
                printf("\n1)Create:");
                printf("\n2)Insert:");
                printf("\n3)Delete:");
                printf("\n4)Print:");
                printf("\n5)Search:");
                printf("\n6)Quit:");
                printf("\n\nEnter Your Choice:");
                scanf("%d",&op);

                switch(op)
                {
                        case 1: printf("\nEnter no. of elements:");
                                scanf("%d",&n);
                                printf("\nEnter tree data:");
                                root=NULL;
                                for(i=0;i<n;i++)
                                {
                                        scanf("%d",&x);
                                        root=insert(root,x);
                                }
                                break;

                        case 2: printf("\nEnter a data:");
                                scanf("%d",&x);
                                root=insert(root,x);
                                break;

                        case 3: printf("\nEnter a data:");
                                scanf("%d",&x);
                                root=Delete(root,x);
                                break;

                        case 4: printf("\nPreorder sequence:\n");
                                preorder(root);
```

```
                printf("\n\nInorder sequence:\n");
                inorder(root);
                printf("\n");
                break;
                                case 5:printf("Enter key");
                                   scanf("%d",&x);
                                   search(root,x);
                                   break;
        }
    }while(op!=6);

    return 0;
}

node * insert(node *T,int x)
{
    if(T==NULL)
    {
        T=(node*)malloc(sizeof(node));
        T->data=x;
        T->left=NULL;
        T->right=NULL;
    }
    else
        if(x > T->data)         // insert in right subtree
        {
            T->right=insert(T->right,x);
            if(BF(T)==-2)
                if(x>T->right->data)
                    T=RR(T);
                else
                    T=RL(T);
        }
        else
            if(x<T->data)
            {
                T->left=insert(T->left,x);
                if(BF(T)==2)
                    if(x < T->left->data)
                        T=LL(T);
                    else
                        T=LR(T);
            }

        T->ht=height(T);

        return(T);
}

node * Delete(node *T,int x)
{
    node *p;

    if(T==NULL)
    {
        return NULL;
    }
    else
        if(x > T->data)         // insert in right subtree
        {
            T->right=Delete(T->right,x);
            if(BF(T)==2)
                if(BF(T->left)>=0)
                    T=LL(T);
```

```
                        else
                                T=LR(T);
                }
                else
                        if(x<T->data)
                        {
                                T->left=Delete(T->left,x);
                                if(BF(T)==-2)   //Rebalance during windup
                                        if(BF(T->right)<=0)
                                                T=RR(T);
                                        else
                                                T=RL(T);
                        }
                        else
                        {
                                //data to be deleted is found
                                if(T->right!=NULL)
                                {       //delete its inorder succesor
                                        p=T->right;

                                        while(p->left!= NULL)
                                                p=p->left;

                                        T->data=p->data;
                                        T->right=Delete(T->right,p->data);

                                        if(BF(T)==2)//Rebalance during windup
                                                if(BF(T->left)>=0)
                                                        T=LL(T);
                                                else
                                                        T=LR(T);\
                                }
                                else
                                        return(T->left);
                        }
        T->ht=height(T);
        return(T);
}

int height(node *T)
{
        int lh,rh;
        if(T==NULL)
                return(0);

        if(T->left==NULL)
                lh=0;
        else
                lh=1+T->left->ht;

        if(T->right==NULL)
                rh=0;
        else
                rh=1+T->right->ht;

        if(lh>rh)
                return(lh);

        return(rh);
}

node * rotateright(node *x)
{
        node *y;
```

```
        y=x->left;
        x->left=y->right;
        y->right=x;
        x->ht=height(x);
        y->ht=height(y);
        return(y);
}

node * rotateleft(node *x)
{
        node *y;
        y=x->right;
        x->right=y->left;
        y->left=x;
        x->ht=height(x);
        y->ht=height(y);

        return(y);
}

node * RR(node *T)
{
        T=rotateleft(T);
        return(T);
}

node * LL(node *T)
{
        T=rotateright(T);
        return(T);
}

node * LR(node *T)
{
        T->left=rotateleft(T->left);
        T=rotateright(T);

        return(T);
}

node * RL(node *T)
{
        T->right=rotateright(T->right);
        T=rotateleft(T);
        return(T);
}

int BF(node *T)
{
        int lh,rh;
        if(T==NULL)
                return(0);

        if(T->left==NULL)
                lh=0;
        else
                lh=1+T->left->ht;

        if(T->right==NULL)
                rh=0;
        else
                rh=1+T->right->ht;

        return(lh-rh);
```

```
}

void preorder(node *T)
{
        if(T!=NULL)
        {
                printf("%d(Bf=%d)",T->data,BF(T));
                preorder(T->left);
                preorder(T->right);
        }
}

void inorder(node *T)
{
        if(T!=NULL)
        {
                inorder(T->left);
                printf("%d(Bf=%d)",T->data,BF(T));
                inorder(T->right);
        }
}
void search(node *T,int key)
{
if(T==NULL)
printf("Value not found");
else if(key<T->data)
search(T->left,key);
else if(key>T->data)
search(T->right,key);
else
{
if(key==T->data)
printf("key found%d",key);
}
}
```