NATIONAL INSTITUTE OF TECHNOLOGY

MANIPUR

(An Autonomous Institute under MHRD, Govt. of India)

Data Structure Lab – CS231

Academic Session Jul-Dec, 2021

(IIIrd Semester)

Submitted by:

Preceela Chanu Irengbam

20103028

Submitted to:

MS. Ksh Merina Devi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY MANIPUR

Imphal, Manipur-795001

(An Autonomous Institute under MHRD, Govt. of India)

INDEX

This page is intentionally left blank.

AIM : To implement Bubble sort using a C program.

CODE :

```
//A C program to implement Bubble sort.

#include<stdio.h>
void swap(int *a,int *b)                    //Swapping function
{
   int temp;
   temp=*a;
   *a=*b;
   *b=temp;
}
void bubbleSort(int A[],int n)              //Bubble sort function
{
   int i,j,temp;
   for(i=1;i<n;i++)
   {
      for(j=0;j<n-i;j++)
      {
         if(A[j]>A[j+1])
            swap(&A[j],&A[j+1]);    //Calling swap() by passing arguments, &A[j] and &A[j+1]
      }
   }
}
int main()                                  //Main function
{
   int A[10],i,n;
   printf("Enter the number of elements in the list: ");
   scanf("%d",&n);
   printf("\nEnter the elements: ");
   for(i=0;i<n;i++)
```

```
        scanf("%d",&A[i]);
    bubbleSort(A,n);                    //Calling bubbleSort() by passing values, array A and n
    printf("\nThe sorted list is");
    for(i=0;i<n;i++)
        printf(" %d ",A[i]);            //Printing sorted array A
    return 0;
}
```

OUTPUT :

```
Enter the number of elements in the list: 5

Enter the elements: 2 4 1 90 -10

The sorted list is -10  1  2  4  90
Process returned 0 (0x0)   execution time : 21.663 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Selection sort using a C program.

CODE :

```
// A c program to implement Selection sort.

#include<stdio.h>
void swap(int *a,int *b)                    //Swapping function
{
   int temp;
   temp=*a;
   *a=*b;
   *b=temp;
}
void selectionSort(int A[],int n)          //Selection sort function
{
   int i,j,min;
   for(i=0;i<n-1;i++)
   {
      min=i;
      for(j=i+1;j<n;j++)
      {
         if(A[j]<A[min])
                  min=j;
      }
      swap(&A[i],&A[min]);          //Calling swap() by passing arguments, &A[i] and &A[min]
   }
}
int main()                                  //Main function
{
   int A[10],i,n;
   printf("Enter the number of elements in the list: ");
   scanf("%d",&n);
```

```
    printf("\nEnter the elements: ");
  for(i=0;i<n;i++)
            scanf("%d",&A[i]);
    selectionSort(A,n);          //Calling selectionSort() function by passing values, array A and n
    printf("\nThe sorted list is");
    for(i=0;i<n;i++)
             printf(" %d ",A[i]);   //Printing the sorted array
    return 0;
}
```

OUTPUT :

```
Enter the number of elements in the list: 5

Enter the elements: 70 99 -91 -21 10

The sorted list is -91  -21  10  70  99
Process returned 0 (0x0)   execution time : 19.044 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Insertion sort using a C program.

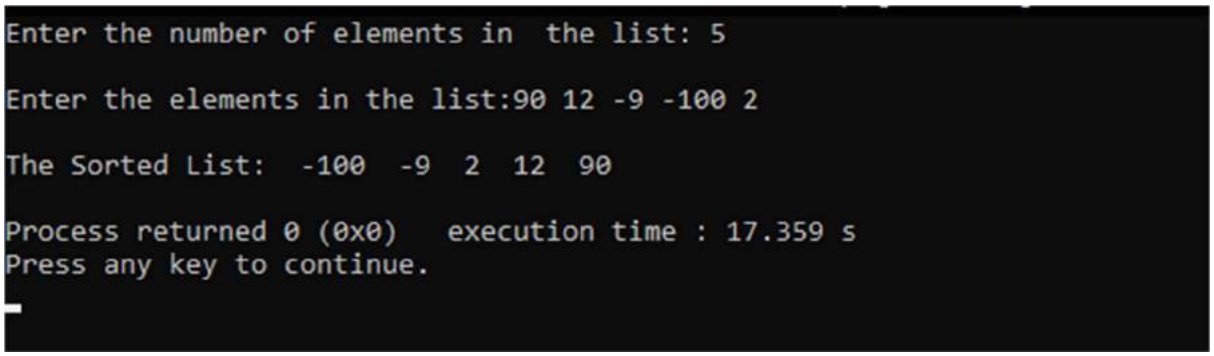
CODE :

```
//A C program to implement Insertion sort.

#include<stdio.h>
void insrtSort(int A[],int n)                    //Insertion sort function
{
   int i,hole,value;
   for(i=0;i<n;i++)
   {
      value=A[i];
      hole=i;
      while(hole>0&&A[hole-1]>value)
      {
         A[hole]=A[hole-1];
         hole--;
      }
      A[hole]=value;
   }
}
int main()                                       //Main function
{
   int A[10],n,i;
   printf("Enter the number of elements in  the list: ");
   scanf("%d",&n);
   printf("\nEnter the elements in the list:");
   for(i=0;i<n;i++)
         scanf("%d",&A[i]);
   insrtSort(A,n);                    //Calling insrtSort() by passing values, array A and n
   printf("\nThe Sorted List: ");
   for(i=0;i<n;i++)                   //printing sorted array
```

```
        printf(" %d ",A[i]);
    printf("\n");
    return 0;
}
```

OUTPUT :



RESULT : The implementation is successfully done.

AIM : To implement Merge sort using a C program.

CODE :

```
//A C program to implement Merge sort.

#include<stdio.h>
#include<stdlib.h>
void mergeSort(int A[],int n)            //Merge sort function
{
   int mid,i,*l,*r;
   if(n<2)
   return;
   mid=n/2;
   l=(int *)malloc(mid*sizeof(int));
   r=(int *)malloc((n-mid)*sizeof(int));
   for(i=0;i<mid;i++)
         l[i]=A[i];
   for(i=mid;i<n;i++)
         r[i-mid]=A[i];
   mergeSort(l,mid);                  //Recursively calling mergeSort()
   mergeSort(r,n-mid);
   merge(A,l,r,mid,n-mid);
   free(l);
   free(r);
}
void merge(int A[],int l[],int r[],int left,int right)      //Merge function to merge two arrays
{
   int i=0,j=0,k=0;
   while(i<left&&j<right)
      {
         if(l[i]<r[j])
         {
```

```
            A[k]=l[i];
          i++;
           }
           else
           {
              A[k]=r[j];
              j++;
           }
           k++;
       }
    while(i<left)
    {
       A[k]=l[i];
       i++;
       k++;
    }
    while(j<right)
    {
       A[k]=r[j];
       j++;
       k++;
    }
}
int main()
{
    int A[100],i,n;
    printf("Enter the number of elements in the list:");
    scanf("%d",&n);
    printf("\nEnter the elements:\n");
    for(i=0;i<n;i++)
          scanf("%d",&A[i]);
```

```
    mergeSort(A,n);                 //calling mergeSort()
    printf("\nThe elements after sorting by Merge sort are:\n");
    for(i=0;i<n;i++)                //printing sorted array
            printf(" %d ",A[i]);
    printf("\n");
}
```

OUTPUT :

```
Enter the number of elements in the list:7

Enter the elements:
10 -10 9 3 21 99 100

The elements after sorting by Merge sort are:
 -10  3  9  10  21  99  100

Process returned 0 (0x0)   execution time : 32.200 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Quick sort using a C program.

CODE :

```
//A C program to implement Quick sort.
#include<stdio.h>
#include<stdlib.h>
void swap(int *a,int *b)                    //Swap function
{
   int temp;
   temp=*a;
   *a=*b;
   *b=temp;
}
int partition(int A[],int start,int end)            //Partition function
{
   int p,i;
   int pivot=A[end];
   p=start;
   for(i=start;i<end;i++)
   {
      if(A[i]<=pivot)
      {
         swap(&A[i],&A[p]);
         p++;
      }
   }
   swap(&A[p],&A[end]);
   return p;
}
void quickSort(int A[],int start,int end)        //Quick sort function
{
```

```
    int p;
    if(start>=end)
        return;
    p=partition(A,start,end);          //Calling partition()
    quickSort(A,start,p-1);            //Recursively calling quickSort()
    quickSort(A,p+1,end);
}
int main()                             //Main function
{
    int A[100],i,n;
    printf("Enter the number of elements in the list:");
    scanf("%d",&n);
    printf("\nEnter the elements:");
    for(i=0;i<n;i++)
        scanf("%d",&A[i]);
    quickSort(A,0,n-1);                //Calling quickSort()
    printf("\nThe elements after sorting by Quick Sort are:\n");
    for(i=0;i<n;i++)                   //printing the sorted array
        printf(" %d ",A[i]);
    printf("\n");
}
```

OUTPUT :

```
Enter the number of elements in the list:7

Enter the elements:90 -10 -100 2 1 3 4

The elements after sorting by Quick Sort are:
 -100  -10  1  2  3  4  90

Process returned 0 (0x0)   execution time : 19.251 s
Press any key to continue.
```

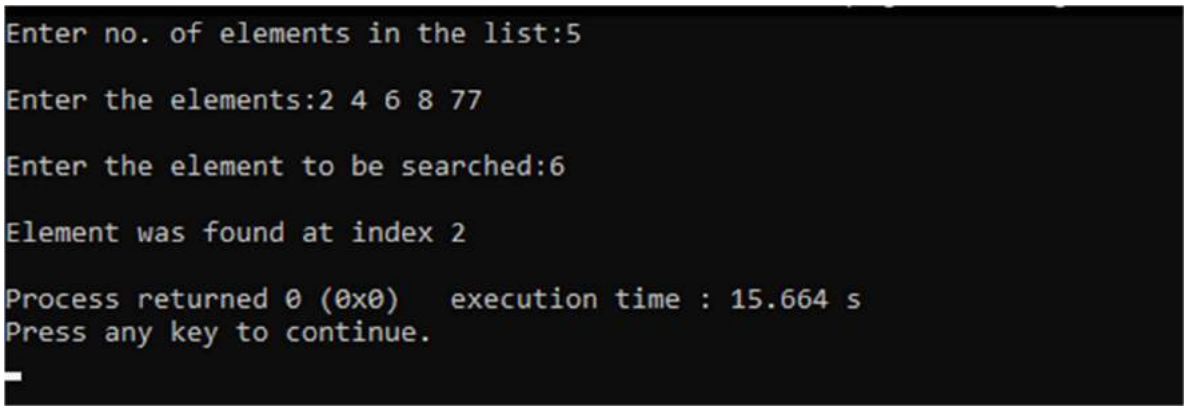
RESULT : The implementation is successfully done.

AIM : To implement Linear search algorithm using a C program.

CODE :

```
//A C program to implement Linear search algorithm.
#include<stdio.h>
#include<stdlib.h>
int linearsearch(int A[],int key,int n)            //Linear search function
{
  int i;
  for(i=0;i<=n;i++)
   {
      if(i==n&&A[i]!=key)
      {
         printf("\nKey not found");
         exit (0);
      }
      if(A[i]==key)
         return i;
   }
}
int main()                                         //Main function
{
   int A[10],i,n,key;
   printf("Enter no. of elements in the list:");
   scanf("%d",&n);
   printf("\nEnter the elements:");
   for(i=0;i<n;i++)
         scanf("%d",&A[i]);
   printf("\nEnter the element to be searched:");
   scanf("%d",&key);
   key=linearsearch(A,key,n);                      //passing values A and key to linearsearch()
   printf("\nElement was found at index %d\n",key);}
```

OUTPUT :



RESULT : The implementation is successfully done.

AIM : To implement Binary search algorithm using a C program.

CODE :

```
//A C program to implement Binary search algorithm.

#include<stdio.h>
#include<stdlib.h>
int binarySearch(int A[],int key,int left,int right)       //Binary search algorithm
{
   int i,mid;
   mid=left+(right-left)/2;
   if(left>right)
      {
         printf("\nElement is not in the list\n");
         exit (0);
      }
   else if(A[mid]==key)
         return mid;
   else if(A[mid]>key)
         binarySearch(A,key,left,mid-1);        //Recursively calling binarySearch()
   else
         binarySearch(A,key,mid+1,right);
}
void swap(int *a,int *b)                        //Swapping function
{
   int temp;
   temp=*a;
   *a=*b;
   *b=temp;
}
void bubbleSort(int A[],int n)                  //A sorting function to sort the array
{
```

```
    int i,j,temp;
    for(i=1;i<n;i++)
    {
        for(j=0;j<n-i;j++)
        {
            if(A[j]>A[j+1])
                    swap(&A[j],&A[j+1]);
        }
    }
}
int main()
{
    int A[10],i,n,key;
    printf("Enter no. of elements in the list:");
    scanf("%d",&n);
    printf("\nEnter the elements:");
    for(i=0;i<n;i++)
          scanf("%d",&A[i]);
    bubbleSort(A,n);                    //Entered elements are sorted first
    printf("\nThe elements in the list in sorted order:\n");
    for(i=0;i<n;i++)
          printf(" %d ",A[i]);
    printf("\nEnter the element to be searched:");
    scanf("%d",&key);                   //Taking the element to be search as key
    key=binarySearch(A,key,0,n-1);    //returning the position of key in sorted list to key
//printing the positon of the key in the sorted list
    printf("\nElement was found at index %d in the sorted list\n",key);
}
```

OUTPUT :

```
Enter no. of elements in the list:5

Enter the elements:23 17 3 5 2

The elements in the list in sorted order:
 2  3  5  17  23
Enter the element to be searched:17

Element was found at index 3 in the sorted list

Process returned 0 (0x0)   execution time : 17.932 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Linear singly linked list.

CODE :

```
//A C program to implement Linked list
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
//Structure of a node
struct node
{
   int info;
   struct node *next;
};
//Defining *list
struct node *list;
struct node *getnode()
{
   struct node *temp;
   temp= (struct node *)malloc(sizeof(struct node));
   return temp;
};
//A Function to print the values of the nodes in the linked list
void print()
{
   struct node *p;
   for(p=list;p!=NULL;p=p->next)
   {
      if(p->next!=NULL)
      {
         printf("%d-->",p->info);
      }
      else
```

```
        {
            printf("%d",p->info);
        }
    }
}
//A function to insert at the beginning of the linked list
void insert_beg(int x)
{
    struct node *temp;
    temp=getnode();
    temp->info=x;
    temp->next=list;
    list=temp;
    return;
}
//A function to insert after a position specified
void insert_after(struct node *p,int x)
{
    struct node *temp;
    if(p==NULL)
    {
        printf("\nVoid Insertion");
        exit(1);
    }
    temp=getnode();
    temp->info=x;
    temp->next=p->next;
    p->next=temp;
    return;
}
//A function to insert at the end of the linked list
```

```
void insert_end(int x)
{
   struct node *temp;
   if(list==NULL)
   {
      temp=getnode();
      temp->info=x;
      temp->next=NULL;
      list=temp;
   }
   else
   {
      temp=list;
      while(temp->next!=NULL)
      {
         temp=temp->next;
      }
      insert_after(temp,x);
   }
}
//A function to delete the beginning node the linked list
int del_beg()
{
   int x;
   struct node *temp;
   if(list==NULL)
   {
      printf("\nVoid deletion");
      exit(1);
   }
   temp=list;
```

```
    list=list->next;
    x=temp->info;
    free(temp);
    return x;
}
//A function to delete after a specified node in the linked list
int del_after(struct node *p)
{
    int x;
    struct node *temp;
    if(list==NULL)
    {
        printf("\nVoid deletion");
        exit(1);
    }
    temp=p->next;
    x=temp->info;
    p->next=temp->next;
    free(temp);
    return x;
}
//A function to delete the end node of the linked list
int del_end()
{
    int x;
    struct node *temp;
    if(list==NULL)
    {
        printf("\nVoid deletion");
        exit(1);
    }
```

```
    temp=list;
    while(temp->next->next!=NULL)
        temp=temp->next;
    x=del_after(temp);
    return x;
}
//Main function
main()
{
    int a,x,y,j;
    struct node *temp;
    list=NULL;
    printf("\n1.Insert beginning");
    printf("\n2.Insert end");
    printf("\n3.Delete beginning");
    printf("\n4.Delete end");
    printf("\n5.Insert after");
    printf("\n6.Delete after");
    printf("\n7.Print");
    printf("\n8.Exit");
    while(1)
    {
        printf("\n-----------X-----------");
        printf("\nEnter your choice:");
        scanf("%d",&a);
        switch(a)
        {
            case 1:
                {
                    printf("\nEnter the value:");
                    scanf("%d",&x);
```

```
        insert_beg(x);
        break;
    }
case 2:
    {
        printf("\nEnter the value:");
        scanf("%d",&x);
        insert_end(x);
        break;
    }
case 3:
    {
        del_beg();
        break;
    }
case 4:
    {
        del_end();
        break;
    }
case 7:
    {
        print();
        break;
    }
case 5:
    {
        printf("\nEnter the value you want to add:");
        scanf("%d",&x);
        printf("\nEnter the place in the list after which you want to add:");
        scanf("%d",&y);
```

```
        if(y==0)
            insert_beg(x);
        else
        {
            temp=list;
            for(j=2;j<=y;j++)
                temp=temp->next;
            insert_after(temp,x);
        }
        break;
    }
case 6:
    {
        printf("\nEnter the place in the list after which you want to delete:");
        scanf("%d",&y);
        if(y==0)
            del_beg();
        else
        {
            temp=list;
            for(j=2;j<y;j++)
                temp=temp->next;
            del_after(temp);
        }
        break;
    }
case 8:
    {
        exit(0);
    }
default:
```

```
            {
              printf("\nWrong choice");
              break;
            }
        }
    }
}
```

OUTPUT :

```
1.Insert beginning
2.Insert end
3.Delete beginning
4.Delete end
5.Insert after
6.Delete after
7.Print
8.Exit
-----------X-----------
Enter your choice:1

Enter the value:1

-----------X-----------
Enter your choice:1

Enter the value:2

-----------X-----------
Enter your choice:2

Enter the value:3

-----------X-----------
Enter your choice:5

Enter the value you want to add:4

Enter the place in the list after which you want to add:1

-----------X-----------
Enter your choice:7
2-->4-->1-->3
-----------X-----------
Enter your choice:3

-----------X-----------
Enter your choice:4

-----------X-----------
Enter your choice:7
4-->1
-----------X-----------
```

```
-----------X-----------
Enter your choice:7
2-->4-->1-->3
-----------X-----------
Enter your choice:3

-----------X-----------
Enter your choice:4

-----------X-----------
Enter your choice:7
4-->1
-----------X-----------
Enter your choice:6

Enter the place in the list after which you want to delete:1

-----------X-----------
Enter your choice:7
4
-----------X-----------
Enter your choice:8

Process returned 0 (0x0)   execution time : 67.792 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Circular singly linked list.

CODE :

```
//Circular Singly Linked List
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node
{
   int info;
   struct node *next;
};
struct node *list;
struct node *getnode()
{
   struct node *temp;
   temp= (struct node *)malloc(sizeof(struct node));
   return temp;
};
/*Function to Display */
void print()
{
   struct node *p,*q;
   p=list;
   q=list;
   if(list==NULL)
   {
      printf("\nEmpty list\n");
   }
   else
   {
```

```
    do
      {
          printf("%d -->",p->info);
          p=p->next;
      }while(p->next!=q->next);
   }
}
//Function to insert a node at the beginning
void insert_beg(int x)
{
   struct node *temp,*p;
   temp=getnode();
   temp->info=x;
   if(list==NULL)
   {
      list=temp;
      temp->next=list;
   }
   else
   {
      p=list;
      temp->next=list;
      while(p->next!=list)
         p=p->next;
      p->next=temp;
      list=temp;
   }
   return;
}
//Function to insert a node after a specific position
void insert_after(struct node *p,int x)
```

```
{
   struct node *temp;
   if(p==NULL)
   {
      printf("\nVoid Insertion");
      exit(1);
   }
   temp=getnode();
   temp->info=x;
   temp->next=p->next;
   p->next=temp;
   return;
}
//Function to insert at the end node
void insert_end(int x)
{
   struct node *temp,*p;
   temp=getnode();
   temp->info=x;
   if(list==NULL)
   {
      list=temp;
      temp->next=list;
   }
   else
   {
      p=list;
      temp->next=list;
      while(p->next!=list)
         p=p->next;
      p->next=temp;
```

```
    }
}
//Function to delete the beginning node
void del_beg()
{
    struct node *temp,*p;
    if(list==NULL)
    {
        printf("\nVoid deletion");
        exit(1);
    }
        temp=list;
        list=list->next;
        p=list;
        while(p->next!=temp)
            p=p->next;
        p->next=list;
        free(temp);
}
//Function to delete a node after a specific position
int del_after(struct node *p)
{
    int x;
    struct node *temp;
    if(list==NULL)
    {
        printf("\nVoid deletion");
        exit(1);
    }
    temp=p->next;
    x=temp->info;
```

```
    p->next=temp->next;
    free(temp);
    return x;
}
//Function to delete the end node
void del_end()
{
    if (list == NULL)
    {
        return;
    }

    if (list->next == list)
    {
        free(list);
        return;
    }

    struct node *new_tail;
    struct node *old_tail;

    new_tail = list;
    old_tail = list->next;

    while (old_tail->next != list)
    {
        new_tail = old_tail;
        old_tail = old_tail->next;
    }

    new_tail->next = list;
```

```
    free(old_tail);
    return;
}
//Main Function
main()
{
    int a,x,y,j;
    struct node *temp;
    char choice;
    list=NULL;
    printf("\n1.Insert beginning");
    printf("\n2.Insert end");
    printf("\n3.Insert after");
    printf("\n4.Delete beginning");
    printf("\n5.Delete end");
    printf("\n6.Delete after");
    printf("\n7.Print");
    printf("\n8.Exit");
    while(1)
    {
        printf("\n-----------X-----------");
        printf("\nEnter your choice:");
        scanf("%d",&a);
        switch(a)
        {
            case 1:
                {
                    printf("\nEnter the value:");
                    scanf("%d",&x);
                    insert_beg(x);
                    break;
```

```
    }
case 2:
    {
        printf("\nEnter the value:");
        scanf("%d",&x);
        insert_end(x);
        break;
    }
case 4:
    {
        del_beg();
        break;
    }
case 5:
    {
        del_end();
        break;
    }
case 7:
    {
        print();
        break;
    }
case 3:
    {
        printf("\nEnter the value you want to add:");
        scanf("%d",&x);
        printf("\nEnter the place in the list after which you want to add:");
        scanf("%d",&y);
        if(y==0)
            insert_beg(x);
```

```
        else
        {
           temp=list;
           for(j=2;j<=y;j++)
              temp=temp->next;
           insert_after(temp,x);
        }
        break;
    }
case 6:
    {
        printf("\nEnter the place in the list after which you want to delete:");
        scanf("%d",&y);
        if(y==0)
           del_beg();
        else
        {
           temp=list;
           for(j=2;j<y;j++)
              temp=temp->next;
           del_after(temp);
        }
        break;
    }
case 8:
    {
        exit(0);
    }
default:
    {
      printf("\nWrong choice");
```

```
            break;
          }
      }
   }
}
```

OUTPUT :

```
1.Insert beginning
2.Insert end
3.Insert after
4.Delete beginning
5.Delete end
6.Delete after
7.Print
8.Exit
-----------X------------
Enter your choice:1

Enter the value:1

-----------X------------
Enter your choice:2

Enter the value:2

-----------X------------
Enter your choice:3

Enter the value you want to add:3

Enter the place in the list after which you want to add:2

-----------X------------
Enter your choice:7
1 -->2 -->3 -->
```

```
-----------X-----------
Enter your choice:7
5 -->1 -->2 -->2 -->
-----------X-----------
Enter your choice:4

-----------X-----------
Enter your choice:7
1 -->2 -->2 -->
-----------X-----------
Enter your choice:5

-----------X-----------
Enter your choice:7
1 -->2 -->
-----------X-----------
Enter your choice:6

Enter the place in the list after which you want to delete:1

-----------X-----------
Enter your choice:7
1 -->
```

RESULT : The implementation is successfully done.

AIM : To implement Linear doubly linked list.

CODE :

```
// A C program to implement Linear doubly linked list
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *prev;
   struct node *next;
   int data;
};
//Pointer to structure node, head
struct node *head;
//Defining functions
void ins_beg();//To insert at the beginning
void ins_end();//To insert at the end
void ins_at();//To insert after a node
void del_beg();//To delete the beginning node
void del_end();//To delete the end node
void del_at();//To delete after a value
void print();//To print the contents of the list
void main ()//Main function
{
   int a;
   printf("\n 1.Insert beginning");
   printf("\n 2.Insert end");
   printf("\n 3.Insert after");
   printf("\n 4.Delete beginning");
   printf("\n 5.Delete end");
   printf("\n 6.Delete after");
```

```
printf("\n 7.Print");
printf("\n 8.Exit");
while(1)
{
   printf("\n-----------X-----------");
   printf("\n Enter your choice:");
   scanf("%d",&a);
   switch(a)
   {
      case 1:
         {
            ins_beg();
            break;
         }
      case 2:
         {
            ins_end();
            break;
         }
      case 3:
         {
            ins_at();
            break;
         }
      case 4:
         {
            del_beg();
            break;
         }
      case 5:
         {
```

```
                del_end();
                break;
            }
        case 6:
            {
                del_at();
                break;
            }
        case 7:
            {
                print();
                break;
            }
        case 8:
            {
                exit(0);
            }
        default:
            {
                printf(" Please enter valid choice.");
            }
      }
   }
}
void ins_beg()
{
  struct node *ptr;
  int item;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
```

```
    printf("\nOVERFLOW");
  }
  else
  {
   printf("\n Enter value: ");
   scanf("%d",&item);

  if(head==NULL)
  {
    ptr->next = NULL;
    ptr->prev=NULL;
    ptr->data=item;
    head=ptr;
  }
  else
  {
    ptr->data=item;
    ptr->prev=NULL;
    ptr->next = head;
    head->prev=ptr;
    head=ptr;
  }
  printf("\n Node inserted in the beginning.");
}
}
void ins_end()
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *) malloc(sizeof(struct node));
  if(ptr == NULL)
```

```
  {
     printf("\nOVERFLOW");
  }
  else
  {
     printf("\n Enter value:");
     scanf("%d",&item);
      ptr->data=item;
     if(head == NULL)
     {
        ptr->next = NULL;
        ptr->prev = NULL;
        head = ptr;
     }
     else
     {
       temp = head;
       while(temp->next!=NULL)
       {
          temp = temp->next;
       }
       temp->next = ptr;
       ptr ->prev=temp;
       ptr->next = NULL;
      }

   }
    printf("\n Node inserted at the end.");
}
void ins_at()
{
```

```
  struct node *ptr,*temp;
  int item,loc,i;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\n OVERFLOW");
  }
  else
  {
     temp=head;
     printf(" Enter a position within the position of the last node:");
     scanf("%d",&loc);
     for(i=0;i<loc;i++)
     {
        temp = temp->next;
        if(temp == NULL)
        {
           printf("\n Enter a valid position.");
           return;
        }
     }
     printf(" Enter value:");
     scanf("%d",&item);
     ptr->data = item;
     ptr->next = temp->next;
     ptr -> prev = temp;
     temp->next = ptr;
     temp->next->prev=ptr;
     printf("\n Node inserted after %d position.",loc);
  }
}
```

```
void del_beg()
{
   struct node *ptr;
   if(head == NULL)
   {
      printf("\n UNDERFLOW");
   }
   else if(head->next == NULL)
   {
      head = NULL;
      free(head);
      printf("\n Beginning node deleted.");
   }
   else
   {
      ptr = head;
      head = head -> next;
      head -> prev = NULL;
      free(ptr);
      printf("\n Beginning node deleted\n");
   }

}
void del_end()
{
   struct node *ptr;
   if(head == NULL)
   {
      printf("\n UNDERFLOW");
   }
   else if(head->next == NULL)
```

```
    {
        head = NULL;
        free(head);
        printf("\n End node deleted.");
    }
    else
    {
        ptr = head;
        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\n End node deleted.");
    }
}
void del_at()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the data after which the node is to be deleted: ");
    scanf("%d", &val);
    ptr = head;
    while(ptr -> data != val)
    ptr = ptr -> next;
    if(ptr -> next == NULL)
    {
        printf("\n Void deletion.");
    }
    else if(ptr -> next -> next == NULL)
```

```
    {
        ptr ->next = NULL;
    }
    else
    {
        temp = ptr -> next;
        ptr -> next = temp -> next;
        temp -> next -> prev = ptr;
        free(temp);
        printf("\n Node after node containing %d deleted.",val);
    }
}
void print()
{
    struct node *ptr;
    printf("\n The linear doubly linked list:\n");
    ptr = head;
    if(head == NULL)
    {
        printf("\nVoid list.");
        return;
    }
    while(ptr != NULL)
    {
        printf("<--%d-->",ptr->data);
        ptr=ptr->next;
    }
}
```

OUTPUT :

```
 1.Insert beginning
 2.Insert end
 3.Insert after
 4.Delete beginning
 5.Delete end
 6.Delete after
 7.Print
 8.Exit
-----------X------------
 Enter your choice:1

 Enter value: 22

 Node inserted in the beginning.
-----------X------------
 Enter your choice:2

 Enter value:24

 Node inserted at the end.
-----------X------------
 Enter your choice:3
 Enter the position:1
 Enter value:23

 Node inserted after 1 position.
-----------X------------
 Enter your choice:7

 The linear doubly linked list:
<--22--><--24--><--23-->
-----------X------------
```

```
 Enter your choice:7

 The linear doubly linked list:
<--21--><--22--><--24--><--23-->
-----------X-----------
 Enter your choice:4

 Beginning node deleted

-----------X-----------
 Enter your choice:6

 Enter the data after which the node is to be deleted: 22

 Node after node containing 22 deleted.
-----------X-----------
 Enter your choice:5

 End node deleted.
-----------X-----------
 Enter your choice:7

 The linear doubly linked list:
<--22-->
-----------X-----------
 Enter your choice:8

Process returned 0 (0x0)   execution time : 106.318 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Circular doubly linked list.

CODE :

```
// A C program to implement Circular doubly linked list
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *prev;
   struct node *next;
   int data;
};
//Pointer to structure node, head
struct node *head;
//Defining functions
void ins_beg(); //To insert at the beginning
void ins_end(); //To insert at the end
void ins_at(); //To insert after a node
void del_beg(); //To delete the beginning node
void del_end(); //To delete the end node
void del_at(); //To delete after a value
void print(); //To print the contents of the list
void main () //Main function
{
   int a;
   printf("\n 1.Insert beginning");
   printf("\n 2.Insert end");
   printf("\n 3.Insert after");
   printf("\n 4.Delete beginning");
   printf("\n 5.Delete end");
   printf("\n 6.Delete after");
   printf("\n 7.Print");
```

```
printf("\n 8.Exit");
while(1)
{
   printf("\n-----------X-----------");
   printf("\n Enter your choice:");
   scanf("%d",&a);
   switch(a)
   {
      case 1:
         {
            ins_beg();
            break;
         }
      case 2:
         {
            ins_end();
            break;
         }
      case 3:
         {
            ins_at();
            break;
         }
      case 4:
         {
            del_beg();
            break;
         }
      case 5:
         {
            del_end();
```

```
                break;
            }
        case 6:
            {
                del_at();
                break;
            }
        case 7:
            {
                print();
                break;
            }
        case 8:
            {
                exit(0);
            }
        default:
            {
                printf(" Please enter valid choice.");
            }
      }
   }
}
void ins_beg()
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\nOVERFLOW");
```

```
  }
  else
  {
   printf("\nEnter value:");
   scanf("%d",&item);
   ptr->data=item;
  if(head==NULL)
  {
    head = ptr;
    ptr -> next = head;
    ptr -> prev = head;
  }
  else
  {
     temp = head;
   while(temp -> next != head)
   {
      temp = temp -> next;
   }
   temp -> next = ptr;
   ptr -> prev = temp;
   head -> prev = ptr;
   ptr -> next = head;
   head = ptr;
  }
  printf("\n Beginning node inserted.");
}

}
void ins_end()
{
```

```
struct node *ptr,*temp;
int item;
ptr = (struct node *) malloc(sizeof(struct node));
if(ptr == NULL)
{
   printf("\nOVERFLOW");
}
else
{
   printf("\nEnter value:");
   scanf("%d",&item);
    ptr->data=item;
   if(head == NULL)
   {
      head = ptr;
      ptr -> next = head;
      ptr -> prev = head;
   }
   else
   {
     temp = head;
     while(temp->next !=head)
     {
        temp = temp->next;
     }
     temp->next = ptr;
     ptr ->prev=temp;
     head -> prev = ptr;
  ptr -> next = head;
    }
}
```

```
    printf("\n Node inserted at the end.");
}
void ins_at()
{
  struct node *ptr,*temp;
  int item,loc,i;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\n OVERFLOW");
  }
  else
  {
     temp=head;
     printf(" Enter a position within the position of the last node:");
     scanf("%d",&loc);
     for(i=1;i<loc;i++)
     {
        temp = temp->next;
        if(temp == head)
        {
           printf("\n Enter a valid position.");
           return;
        }
     }
     printf(" Enter value:");
     scanf("%d",&item);
     ptr->data = item;
     ptr->next = temp->next;
     ptr -> prev = temp;
     temp->next = ptr;
```

```
      temp->next->prev=ptr;
      printf("\n Node inserted after %d position.",loc);
   }
}
void del_beg()
{
   struct node *temp;
   if(head == NULL)
   {
      printf("\n UNDERFLOW");
   }
   else if(head->next == head)
   {
      head = NULL;
      free(head);
      printf("\n Beginning node deleted.");
   }
   else
   {
      temp = head;
      while(temp -> next != head)
      {
         temp = temp -> next;
      }
      temp -> next = head -> next;
      head -> next -> prev = temp;
      free(head);
      head = temp -> next;
   }
}
```

```
void del_end()
{
   struct node *ptr;
   if(head == NULL)
   {
      printf("\n UNDERFLOW");
   }
   else if(head->next == head)
   {
      head = NULL;
      free(head);
      printf("\n End node deleted.");
   }
   else
   {
      ptr = head;
      if(ptr->next != head)
      {
         ptr = ptr -> next;
      }
      ptr -> prev -> next = head;
      head -> prev = ptr -> prev;
      free(ptr);
      printf("\n End node deleted.");
   }
}
void del_at()
{
   struct node *ptr, *temp;
   int val;
   printf("\n Enter the data after which the node is to be deleted: ");
```

```
    scanf("%d", &val);
    ptr = head;
    while(ptr -> data != val)
            ptr = ptr -> next;
    if(ptr -> next == NULL)
    {
        printf("\n Void deletion.");
    }
    else if(ptr -> next -> next == NULL)
    {
        ptr ->next = NULL;
    }
    else
    {
        temp = ptr -> next;
        ptr -> next = temp -> next;
        temp -> next -> prev = ptr;
        free(temp);
        printf("\n Node after node containing %d deleted.",val);
    }
}
void print()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nVoid list.");
        return;
    }
    while(ptr -> next != head)
```

```
    {
        printf("<--%d-->", ptr -> data);
        ptr = ptr -> next;
    }
    printf("<--%d-->", ptr -> data);
}
```

OUTPUT :

```
 1.Insert beginning
 2.Insert end
 3.Insert after
 4.Delete beginning
 5.Delete end
 6.Delete after
 7.Print
 8.Exit
-----------X------------
 Enter your choice:1

Enter value:7

 Beginning node inserted.
-----------X------------
 Enter your choice:2

Enter value:9

 Node inserted at the end.
-----------X------------
 Enter your choice:3
 Enter a position within the position of the last node:1
 Enter value:8

 Node inserted after 1 position.
-----------X------------
 Enter your choice:7
<--7--><--8--><--9-->
-----------X------------
```

```
 Enter your choice:7
<--6--><--7--><--8--><--9-->
-----------X------------
 Enter your choice:6

 Enter the data after which the node is to be deleted: 6

 Node after node containing 6 deleted.
-----------X------------
 Enter your choice:4

 Beginning node deleted.
-----------X------------
 Enter your choice:7
<--8--><--9-->
-----------X------------
 Enter your choice:5

 End node deleted.
-----------X------------
 Enter your choice:7
<--8-->
-----------X------------
 Enter your choice:8

Process returned 0 (0x0)   execution time : 83.091 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Stack using arrays.

CODE :

```
//A C program to implement Stack using Arrays
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
//Defining the size of the stack to be used, here 20
#define STACKSIZE 20
//Structure of the stack
/*Contains top element and items*/
struct stack
{
   int top;
   int item[STACKSIZE];
};
//Defining a global variable Stack S
struct stack S;
//Function to push elements in the stack
void push(int x)
{
   //Condition for Stack Overflow
   if(S.top==STACKSIZE-1)
   {
      printf("Stack overflow!");
      exit(1);
   }
   S.top++;
   S.item[S.top]=x;
   return;
}
```

```
//Function to pop the top element
void pop()
{
   //Condition for Stack Underflow
   if(S.top==-1)
   {
      printf("Stack underflow!");
      return;
   }
   S.top--;
}
//Function to display the elements in the Stack starting with top element on the left
void print()
{
   int i;
   if(S.top==-1)
   {
      printf("\nStack is empty.\n");
      return;
   }
   printf("\nThe elements in the stack:\n");
   for(i=S.top;i>=0;i--)
      printf("%d\t",S.item[i]);
}
//Main Function
main()
{
   char ch;
   int choice,x;
   S.top=-1;
   printf("\n1. Push");
```

```
printf("\n2. Pop");
printf("\n3. Print");
printf("\n4. Exit");
while(1)
{
   printf("\nEnter your choice:");
   scanf("%d",&choice);
   switch(choice)
   {
      case 1:
         {
            printf("\nEnter the value:");
            scanf("%d",&x);
            push(x);
            break;
         }
      case 2:
         {
            pop();
            break;
         }
      case 3:
         {
            print();
            break;
         }
      case 4:
         {
            exit(0);
         }
      default:
```

```
            {
              printf("\nWrong choice");
              break;
            }
        }
    }
}
```

OUTPUT :

```
1. Push
2. Pop
3. Print
4. Exit
Enter your choice:1

Enter the value:1

Enter your choice:1

Enter the value:2

Enter your choice:1

Enter the value:3

Enter your choice:3

The elements in the stack:
3       2       1
Enter your choice:2

Enter your choice:3

The elements in the stack:
2       1
Enter your choice:2

Enter your choice:2

Enter your choice:2
Stack underflow!
Enter your choice:4

Process returned 0 (0x0)   execution time : 42.161 s
Press any key to continue.
```

RESULT : The implementation is succesfully done.

AIM : To implement Stack using linked list.

CODE :

```
//A C Program to implement Stack Using Linked List
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
//Structure of a node in the linked list(here we are using linear linked list)
struct node
{
   int info;
   struct node *next;
};
//Defining a pointer top to always point at top element of the stack
struct node *top;
struct node *getnode()
{
   struct node *temp;
   temp= (struct node *)malloc(sizeof(struct node));
   return temp;
};
//A function to print the elements in the stack starting from top element in the left
void print()
{
   struct node *p;
   if(top==NULL)
   {
      printf("\nEmpty Stack.");
      return;
   }
   for(p=top;p!=NULL;p=p->next)
```

```
    {
        if(p->next!=NULL)
        {
            printf("%d-->",p->info);
        }
        else
        {
            printf("%d",p->info);
        }
    }
}
//A function to push elements in the stack and update the top pointer
void push(int x)
{
    struct node *temp;
    temp=getnode();
    temp->info=x;
    temp->next=top;
    top=temp;
    return;
}
//A function to pop element in the stack and update the top pointer
int pop()
{
    int x;
    struct node *temp;
    if(top==NULL)
    {
        printf("\nStack Underflow");
        return;
    }
```

```
    temp=top;
    top=top->next;
    x=temp->info;
    free(temp);
    return x;
}
//Main Function
main()
{
    int a,x;
    struct node *temp;
    char choice;
    top=NULL;
    printf("\n1. Push");
    printf("\n2. Pop");
    printf("\n3. Print");
    printf("\n4. Exit");
    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                {
                    printf("\nEnter the value:");
                    scanf("%d",&x);
                    push(x);
                    break;
                }
            case 2:
```

```
            {
                pop();
                break;
            }
        case 3:
            {
                print();
                break;
            }
        case 4:
            {
                exit(0);
            }
        default:
            {
              printf("\nWrong choice");
              break;
            }
      }
   }
}
```

OUTPUT :

```
1. Push
2. Pop
3. Print
4. Exit
Enter your choice:1

Enter the value:2

Enter your choice:1

Enter the value:3

Enter your choice:1

Enter the value:4

Enter your choice:3
4-->3-->2
Enter your choice:2

Enter your choice:2

Enter your choice:3
2
Enter your choice:2

Enter your choice:4

Process returned 0 (0x0)   execution time : 49.040 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement postfix evaluation.

CODE :

```
//A C program to implement postfix evaluation
//Here, this program can evaluate operands from 0 to 9 only
//For example, it cannot identify 12 and will identify it as operands 1 and 2
#include<conio.h>
#include<stdio.h>
#include<math.h>
#define MAXCOL 50
//A function to return operand(operator)operand
// In this program, we can perform only +,-,*,/ and ^.
//All the operators are classified as illegal operation.
double oper(char symb,double op1,double op2)
{
   switch(symb)
   {
      case '+':{return(op1+op2);}
      case '-':{return(op1-op2);}
      case '*':{return(op1*op2);}
      case '/':{return(op1/op2);}
      case '^':{return(pow(op1,op2));}
      default:{printf("\nIllegal Operation");break;}
   }
}
//Main Function
void main()
{
   char exp[MAXCOL];
   double opndstack[MAXCOL];
   char ch;int i,top=0;
```

```
    double opnd1,opnd2,value;
    printf("\nEnter the postfix expression:");
    gets(exp);
    for(i=0; (ch=exp[i])!='\0' ; i++)
    {
       if(ch== '+' || ch=='-' || ch=='*' || ch=='^' || ch=='/')
       {
          opnd2=opndstack[--top];
          opnd1=opndstack[--top];
          value=oper(ch,opnd1,opnd2);
          opndstack[top]=value;
       }
       else
       {
          opndstack[top]=ch-48;
       }
       top++;
    }
    printf("\nResult is %.2f",opndstack[0]);
    getch();
}
```

OUTPUT :

```
Enter the postfix expression:62^22/+45*+9-

Result is 48.00
```

RESULT : The implementation is successfully done.

AIM : To implement infix to postfix conversion.

CODE :

```
//Infix to Postfix
#include<stdio.h>
#include <string.h>
//Global defining character expressions to be used in all scope
char postfix[50];
char infix[50];
//Golbal defining operator stack
char opstack[50];
int i, j, top = 0;
//A Function that checks priority of operators
int lesspriority(char op, char op_at_stack)
{
   int k;
   int pv1; // priority value of op
   int pv2; // priority value of op_at_stack
   char operators[] = {'+', '-', '*', '/', '%', '^', '(' };
   int priority_value[] = {0,0,1,1,2,3,4};
   if( op_at_stack == '(' )
      return 0;
   for(k = 0; k < 6; k ++)
   {
      if(op == operators[k])
         pv1 = priority_value[k];
   }
   for(k = 0; k < 6; k ++)
   {
      if(op_at_stack == operators[k])
         pv2 = priority_value[k];
   }
```

```
    if(pv1 < pv2)
        return 1;
    else
        return 0;
}
//A Function to push operators into stack
void push(char op)
{
    if(top == 0)
    {
        opstack[top] = op;
        top++;
    }
    else
    {
        if(op != '(' )
        {
            while(lesspriority(op, opstack[top-1]) == 1 && top > 0)
            {
                postfix[j] = opstack[--top];
                j++;
            }
        }
        opstack[top] = op;
        top++;
    }
}
//A Function to pop the stack until top element is (
void pop()
{
    while(opstack[--top] != '(' )
```

```
    {
        postfix[j] = opstack[top];
        j++;
    }
}
//Main Funtion
void main()
{
    char ch;
    printf("\n Enter Infix Expression : ");
    gets(infix);
    while ((ch=infix[i++]) != '\0')
        {
            switch(ch)
            {
                case ' ' : break;
                case '(' :
                case '+' :
                case '-' :
                case '*' :
                case '/' :
                case '^' :
                case '%' :push(ch); /* check priority and push */
                break;
                case ')' :pop();break;
                default :postfix[j] = ch;
                j++;
            }
        }
    while(top >= 0)
    {
```

```
        postfix[j] = opstack[--top];
        j++;
    }
    postfix[j] = '\0';
    printf("\n Infix Expression : %s ", infix);
    printf("\n Postfix Expression : %s ", postfix);
    getch();
}
```

OUTPUT :

```
Enter Infix Expression : A+B^C*D/E

Infix Expression : A+B^C*D/E
Postfix Expression : ABC^DE/*+
```

RESULT : The implementation is successfully done.

AIM : To implement postfix to infix conversion.

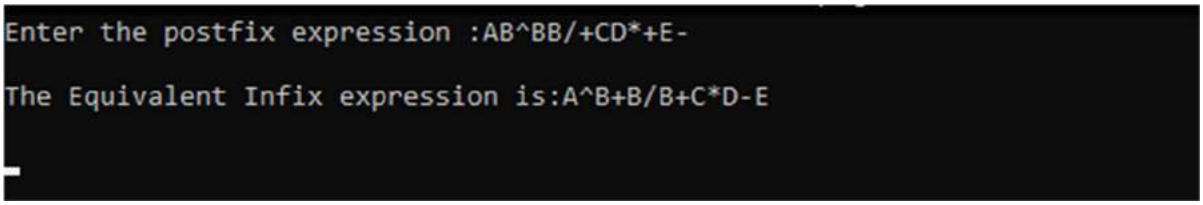
CODE :

```
//Postfix To Infix Conversion
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int top = 10;
//Declaring a node to be used in the program sampling doubly linear linked list
struct node
{
        char ch;
        struct node *next;
        struct node *prev;
};
typedef struct node node;
node *stack[11];
//push() pushes str in stack after checking overflow condition
//it pushes string instead of character
void push(node *str)
{
        if (top <= 0)
      printf("Stack Overflow!! ");
        else
        {
                stack[top] = str;
                top--;
        }
}
//pop() pops out str in the stack last entered
node *pop()
```

```
{
        node *exp;
        if (top >= 10)
                printf("Stack Underflow!! ");
        else
                exp = stack[++top];
        return exp;
}
//convert() checks for operand and operator.
//if operand, pushes it to stack
//if operator, pops out last two operands in stack
//and pushes str(operand operator operand) to stack.
//In short, it is the function that converts
//the postfix expression to infix expression.
void convert(char exp[])
{
        node *op1,  *op2;
        node *temp;
        int i;
        for (i=0;exp[i]!='\0';i++)
        if (exp[i] >= 'a'&& exp[i] <= 'z'|| exp[i] >= 'A' && exp[i] <= 'Z')
        {
                temp = (node*)malloc(sizeof(node));
                temp->ch = exp[i];
                temp->next = NULL;
                temp->prev = NULL;
                push(temp);
        }
        else if (exp[i] == '+' || exp[i] == '-' || exp[i] == '*' || exp[i] == '/' || exp[i] == '^')
        {
                op1 = pop();
```

```
                op2 = pop();
                temp = (node*)malloc(sizeof(node));
                temp->ch = exp[i];
                temp->next = op1;
                temp->prev = op2;
                push(temp);
        }
}
//display() prints converted infix expression.
void display(node *temp)
{
        if (temp != NULL)
        {
                display(temp->prev);
                printf("%c", temp->ch);
                display(temp->next);
        }
}
//Main function.
void main()
{
        char exp[50];
        printf("Enter the postfix expression :");
        scanf("%s", exp);
        convert(exp);
        printf("\nThe Equivalent Infix expression is:");
        display(pop());
        printf("\n\n");
        getch();
}
```

OUTPUT :



RESULT : The implementation is successfully done.

AIM : To implement Queue using arrays.

CODE :

```
//A Program for implementing Queue using arrays
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 20
//Structure of a queue
struct queue
{
   int item[MAXSIZE];
   int front,rear;
};
//Declaring a queue called q
struct queue q;
//A function print elements in the queue
void print()
{
   int i;
   //Checks if the queue is empty and if not, only then performs printing
   if(q.front == -1 && q.rear == -1)
   {
      printf("\nQueue is empty.\n");
      return;
   }
   printf("\nThe elements in the queue starting from front are:\n");
   for(i = q.front; i <= q.rear; i++)
      printf("%d\t",q.item[i]);
}
//A function enqueue(add) elements in the queue
void enqueue(int x)
```

```
{
   //Checks if the queue is FULL
   //and if not, only then adds elements to the queue
   if(q.rear == MAXSIZE - 1)
   {
      printf("\nQueue OVERFLOW\n");
      return;
   }
   else
   {
      if(q.front = -1)
         q.front = 0;
      q.rear ++;
   }
   q.item[q.rear] = x;
}
//A function to dequeue(delete) elements from the queue
int dequeue()
{
   int x;
   x=q.item[q.front];
   //Checks of the queue is EMPTY
   //and if not, checks if the queue has only one element
   //if yes, dequeue and updates the value of front and rear to -1
   //or if no, performs dequeue
   if(q.rear == -1 && q.front == -1)
   {
      printf("\nQueue UNDERFLOW\n");
      return;
   }
   else if(q.rear == q.front)
```

```
 {
      q.rear = -1;
      q.front = -1;
   }
   else
   {
      q.front++;
   }
   return x;
}
//Main function
main()
{
   char ch;
   int choice,x;
   q.front = -1;
   q.rear = -1;
   printf("\nChoose:\n");
   printf("\n1. Enqueue");
   printf("\n2. Dequeue");
   printf("\n3. Print");
   printf("\n4. Exit");
   while(1)
   {
      printf("\n-----------X-----------");
      printf("\nEnter your choice:");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
            {
```

```
            printf("\nEnter the value:");
            scanf("%d",&x);
            enqueue(x);
            break;
          }
        case 2:
          {
            dequeue();
            break;
          }
        case 3:
          {
            print();
            break;
          }
        case 4:
          {
            exit(0);
          }
        default:
          {
           printf("\nWrong choice");
            break;
          }
      }
  }
}
```

OUTPUT :

```
Choose:

1. Enqueue
2. Dequeue
3. Print
4. Exit
-----------X-----------
Enter your choice:1

Enter the value:1

-----------X-----------
Enter your choice:1

Enter the value:2

-----------X-----------
Enter your choice:1

Enter the value:3

-----------X-----------
Enter your choice:3

The elements in the queue starting from front are:
1       2       3
-----------X-----------
Enter your choice:2

-----------X-----------
Enter your choice:3

The elements in the queue starting from front are:
2       3
-----------X-----------
Enter your choice:4

Process returned 0 (0x0)   execution time : 34.625 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Circular Queue using circular array.

CODE :

```
//A Program for implementing Queue using array
#include<stdio.h>
#include<stdlib.h>
//Maximum size of the circular queue is 20
#define N 20
//Structure of a queue
struct queue
{
   int item[N];
   int front,rear;
};
//Declaring a queue called q
struct queue q;
//A function print elements in the queue
void print()
{
   int i;
   //Checks if the queue is empty and if not, only then performs printing
   if(q.front == -1 && q.rear == -1)
   {
      printf("\nQueue is empty.\n");
      return;
   }
   printf("\nThe elements in the queue starting from front are:\n");
   for(i = q.front; i <= q.rear; i++)
      printf("%d\t",q.item[i]);
}
//A function enqueue(add) elements in the queue
```

```
void enqueue(int x)
{
   //Checks if the queue is FULL
   //Condition for queue is FULL in circular queue is (q.rear+1)%N == q.front
   if((q.rear+1)%N == q.front)
   {
      printf("\nQueue OVERFLOW\n");
      return;
   }
   //and if not, checks if queue is empty
   //Condition if queue is empty is q.front=q.rear=-1
   else if(q.front == -1 && q.rear == -1)
   {
      q.front = 0;
      q.rear = 0;
      q.item[q.rear] = x;
   }
   //and if not, performs update of queue with q.rear=(q.rear+1)%N
   else
   {
      q.rear=(q.rear+1)%N;
      q.item[q.rear]=x;
   }
}
//A function to dequeue(delete) elements from the queue
int dequeue()
{
   int x;
   //Checks of the queue is EMPTY
   //and if not, checks if the queue has only one element
   //if yes, dequeue and updates the value of front and rear to -1
```

```
    //or if no, performs dequeue with q.front=(q.front+1)%N
    if(q.rear == -1 && q.front == -1)
    {
        printf("\nQueue UNDERFLOW\n");
        return;
    }
    else if(q.rear == q.front)
    {
        x=q.item[q.front];
        q.rear = -1;
        q.front = -1;
    }
    else
    {
        x=q.item[q.front];
        q.front=(q.front+1)%N;
    }
    //returning the dequeued element just in case
    return x;
}
//Main function
main()
{
    char ch;
    int choice,x;
    q.front = -1;
    q.rear = -1;
    printf("\nChoose:\n");
    printf("\n1. Enqueue");
    printf("\n2. Dequeue");
    printf("\n3. Print");
```

```
printf("\n4. Exit");
while(1)
{
   printf("\n-----------X-----------");
   printf("\nEnter your choice:");
   scanf("%d",&choice);
   switch(choice)
   {
      case 1:
         {
            printf("\nEnter the value:");
            scanf("%d",&x);
            enqueue(x);
            break;
         }
      case 2:
         {
            dequeue();
            break;
         }
      case 3:
         {
            print();
            break;
         }
      case 4:
         {
            exit(0);
         }
      default:
         {
```

```
            printf("\nWrong choice");
            break;
        }
    }
  }
}
```

OUTPUT :

```
Choose:

1. Enqueue
2. Dequeue
3. Print
4. Exit
-----------X------------
Enter your choice:1

Enter the value:1

-----------X------------
Enter your choice:1

Enter the value:5

-----------X------------
Enter your choice:1

Enter the value:7

-----------X------------
Enter your choice:1

Enter the value:9

-----------X------------
Enter your choice:3

The elements in the queue starting from front are:
1       5       7       9
-----------X------------
Enter your choice:2

-----------X------------
Enter your choice:3

The elements in the queue starting from front are:
5       7       9
-----------X------------
Enter your choice:4

Process returned 0 (0x0)   execution time : 32.757 s
```

RESULT : The implementation is successfully done.

AIM : To implement Queue using linked list.

CODE :

```
//A program to implement queue using linked list
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
//Structure of a node in the queue
struct node
{
   int info;
   struct node *next;
};
//Declaring pointer to node front and rear to always point
//at the front node and rear node in the queue
struct node *front;
struct node *rear;
//Declaring a pointer getnode() to allocate dynamic memory
struct node *getnode()
{
   struct node *temp;
   temp= (struct node *)malloc(sizeof(struct node));
   return temp;
};
//A function to print elements in the queue
void print()
{
   struct node *p;
   //Queue implemented by linked list has only UNDERFLOW condition
   //Here, in this it is checking this condition with front == NULL && rear == NULL
   if(front == NULL && rear == NULL)
```

```
    {
        printf("\nEmpty Queue.");
        return;
    }
    //If not, prints the elements in the queue
    printf("\nThe elements in queue starting from the front are:\n");
    for(p=front;p!=NULL;p=p->next)
    {
        if(p->next!=NULL)
        {
            printf("%d\t",p->info);
        }
        else
        {
            printf("%d",p->info);
        }
    }
}
//A function to enqueue nodes in the queue
void enqueue()
{
    struct node *p;
    p = getnode();
    int x;
    printf("\nEnter the Value: ");
    scanf("%d",&x);
    if(front == NULL)
    {
        p -> info = x;
        front = p;
        rear = p;
```

```
        rear -> next = NULL;
        front -> next = NULL;
    }
    else
    {
        p -> info = x;
        rear -> next = p;
        rear = p;
        rear -> next = NULL;
    }
}
//A function to dequeue nodes in the queue
void dequeue()
{
    struct node *p;
    if(front == NULL && rear == NULL)
    {
        printf("\nQueue UNDERFLOW\n");
        return;
    }
    else
    {
        p = front;
        front = front -> next;
        free(p);
    }
}
//Main Function
main()
{
    char ch;
```

```
int choice;
front = NULL;
rear = NULL;
printf("\nChoose:\n");
printf("\n1. Enqueue");
printf("\n2. Dequeue");
printf("\n3. Print");
printf("\n4. Exit");
while(1)
{
   printf("\n-----------X-----------");
   printf("\nEnter your choice:");
   scanf("%d",&choice);
   switch(choice)
   {
      case 1:
         {
            enqueue();
            break;
         }
      case 2:
         {
            dequeue();
            break;
         }
      case 3:
         {
            print();
            break;
         }
      case 4:
```

```
            {
               exit(0);
            }
         default:
            {
              printf("\nWrong choice");
              break;
            }
      }
   }
}
```

OUTPUT :

```
Choose:

1. Enqueue
2. Dequeue
3. Print
4. Exit
-----------X------------
Enter your choice:1

Enter the Value: 100

-----------X------------
Enter your choice:1

Enter the Value: 200

-----------X------------
Enter your choice:1

Enter the Value: 300

-----------X------------
Enter your choice:3

The elements in queue starting from the front are:
100     200     300
-----------X------------
Enter your choice:2

-----------X------------
Enter your choice:3

The elements in queue starting from the front are:
200     300
-----------X------------
Enter your choice:4

Process returned 0 (0x0)   execution time : 28.306 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Binary Search Tree and perform insertion and deletion.

CODE :

```
//C Program to implement a Binary Search Tree to perform insertion and deletion
//and printing in in-order traversal of the tree
#include <stdio.h>
#include <stdlib.h>
//Declaring a node structure to represent the node in the BST
struct btnode
{
   int value;
   struct btnode *l;
   struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;

//Above, defining nodes root to always point at the root node of the BST
//a temporary node temp initialise to NULL
//nodes t1, t2 to represent left sub-tree and right sub-tree to be later used in the program

//Below, declaring functions to be used in the program

void delete1();
//To delete a node
void insert();
//To insert a node in the tree
void delete();
//To check whether node to be deleted is in left sub-tree or right sub-tree
void create();
//To create a node
void search(struct btnode *t);
//To search the appropriate position to insert a new node
```

```
void search1(struct btnode *t,int data);
//To search left sub-tree for the appropriate position to insert a new node
int smallest(struct btnode *t);
// To find the smallest element in the right sub tree
int largest(struct btnode *t);
// To find the largest element in the left sub tree
void inorder(struct btnode *t);
// A function to print in-order traversal of the BST

int flag = 1;

void main()
{
   int ch;

   printf("\n\n\n\n Choose");
   printf("\n1. Insert a node to BST");
   printf("\n2. Delete a node from the BST");
   printf("\n3. Print in in-order:");
   printf("\n4. Exit\n");
   while(1)
   {
      printf("\n--------------X--------------");
      printf("\nEnter your choice : ");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         insert();
         break;
      case 2:
```

```
            delete();
            break;
        case 3:
            inorder(root);
            break;
        case 4:
            exit(0);
            break;
        default :
            printf("Enter valid choice");
            break;
        }
    }
}

void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
```

```
    temp->value = data;
    temp->l = temp->r = NULL;
}

void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))      //Checking to insert at right
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
    else if ((temp->value < t->value) && (t->l != NULL))    //Checking to insert at left
        search(t->l);
    else if ((temp->value < t->value) && (t->l == NULL))
        t->l = temp;
}

void delete()
{
    int data;

    if (root == NULL)
    {
        printf("No elements in a tree to delete");
        return;
    }
    printf("Enter the data to be deleted : ");
    scanf("%d", &data);
    t1 = root;
    t2 = root;
    search1(root, data);
}
```

```
void search1(struct btnode *t, int data)
{
   if ((data>t->value))
   {
      t1 = t;
      search1(t->r, data);
   }
   else if ((data < t->value))
   {
      t1 = t;
      search1(t->l, data);
   }
   else if ((data==t->value))
   {
      delete1(t);
   }
}

void delete1(struct btnode *t)
{
   int k;

   /* To delete leaf node */
   if ((t->l == NULL) && (t->r == NULL))
   {
      if (t1->l == t)
      {
         t1->l = NULL;
      }
      else
      {
```

```
    t1->r = NULL;
   }
   t = NULL;
   free(t);
   return;
}

/* To delete node having one left hand child */
else if ((t->r == NULL))
{
   if (t1 == t)
   {
      root = t->l;
      t1 = root;
   }
   else if (t1->l == t)
   {
      t1->l = t->l;

   }
   else
   {
      t1->r = t->l;
   }
   t = NULL;
   free(t);
   return;
}

/* To delete node having right hand child */
else if (t->l == NULL)
```

```
{
   if (t1 == t)
   {
      root = t->r;
      t1 = root;
   }
   else if (t1->r == t)
      t1->r = t->r;
   else
      t1->l = t->r;
   t == NULL;
   free(t);
   return;
}
/* To delete node having two child */
  else if ((t->l != NULL) && (t->r != NULL))
  {
   t2 = root;
   if (t->r != NULL)
   {
      k = smallest(t->r);
      flag = 1;
   }
   else
   {
      k =largest(t->l);
      flag = 2;
   }
   search1(root, k);
   t->value = k;
```

```
    }
}
int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
    {
        t2 = t;
        return(smallest(t->l));
    }
    else
        return (t->value);
}
int largest(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}
void inorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("NULL");
        return;
    }
    if (t->l != NULL)
```

```
    inorder(t->l);
  printf("%d \t", t->value);
  if (t->r != NULL)
     inorder(t->r);
}
```

OUTPUT :

```
 Choose
1. Insert a node to BST
2. Delete a node from the BST
3. Print in in-order:
4. Exit

--------------X--------------
Enter your choice : 1
Enter data of node to be inserted : 10

--------------X--------------
Enter your choice : 1
Enter data of node to be inserted : 8

--------------X--------------
Enter your choice : 1
Enter data of node to be inserted : 12

--------------X--------------
Enter your choice : 1
Enter data of node to be inserted : 3

--------------X--------------
Enter your choice : 1
Enter data of node to be inserted : 16

--------------X--------------
Enter your choice : 3
3       8       10      12      16
--------------X--------------
Enter your choice : 2
Enter the data to be deleted : 8

--------------X--------------
Enter your choice : 3
3       10      12      16
--------------X--------------
Enter your choice : 4

Process returned 0 (0x0)   execution time : 45.478 s
Press any key to continue.
```

RESULT : The implementation is successfully done.

AIM : To implement Binary Search Tree and perform printing in in-order, pre-order and post-order traversal form.

CODE :

```
//C Program to implement a Binary Search Tree
//to print it in  Pre-Order,In-Order and Post-Order traversal form

#include <stdio.h>
#include <stdlib.h>
//Declaring a node structure to represent the node in the BST
struct btnode
{
   int value;
   struct btnode *l;
   struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;

//Above, defining nodes root to always point at the root node of the BST
//a temporary node temp initialise to NULL
//nodes t1, t2 to represent left sub-tree and right sub-tree to be later used in the program

//Below, declaring functions to be used in the program

void delete1();
//To delete a node
void insert();
//To insert a node in the tree
void delete();
//To check whether node to be deleted is in left sub-tree or right sub-tree
void create();
//To create a node
```

```
void search(struct btnode *t);
//To search the appropriate position to insert a new node
void search1(struct btnode *t,int data);
//To search left sub-tree for the appropriate position to insert a new node
int smallest(struct btnode *t);
// To find the smallest element in the right sub tree
int largest(struct btnode *t);
// To find the largest element in the left sub tree
void inorder(struct btnode *t);
// A function to print in-order traversal of the BST
void preorder(struct btnode *t);
// A function to print in pre-order traversal of the BST
void postorder(struct btnode *t);
// A function to print in post-order traversal of the BST

int flag = 1;

void main()
{
   int ch;

   printf("\n   Choose");
   printf("\n1. Insert a node to BST");
   printf("\n2. Delete a node from the BST");
   printf("\n3. Print in Pre-Order traversal form");
   printf("\n4. Print in In-Order traversal form");
   printf("\n5. Print in Post-Order traversal form");
   printf("\n6. Exit\n");
   while(1)
   {
      printf("\n----------------X----------------");
```

```
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            insert();
            break;
        case 2:
            delete();
            break;
        case 3:
            preorder(root);
            break;
        case 4:
            inorder(root);
            break;
        case 5:
            postorder(root);
            break;
        case 6:
            exit(0);
            break;
        default :
            printf("Enter valid choice");
            break;
        }
    }
}

void insert()
{
```

```
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}

void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))      //Checking to insert at right
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
    else if ((temp->value < t->value) && (t->l != NULL))    //Checking to insert at left
        search(t->l);
    else if ((temp->value < t->value) && (t->l == NULL))
        t->l = temp;
}
void delete()
{
```

```
    int data;

    if (root == NULL)
    {
        printf("No elements in a tree to delete");
        return;
    }
    printf("Enter the data to be deleted : ");
    scanf("%d", &data);
    t1 = root;
    t2 = root;
    search1(root, data);
}

void search1(struct btnode *t, int data)
{
    if ((data>t->value))
    {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value))
    {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value))
    {
        delete1(t);
    }
}
```

```
void delete1(struct btnode *t)
{
   int k;

   /* To delete leaf node */
   if ((t->l == NULL) && (t->r == NULL))
   {
      if (t1->l == t)
      {
         t1->l = NULL;
      }
      else
      {
         t1->r = NULL;
      }
      t = NULL;
      free(t);
      return;
   }

   /* To delete node having one left hand child */
   else if ((t->r == NULL))
   {
      if (t1 == t)
      {
         root = t->l;
         t1 = root;
      }
      else if (t1->l == t)
      {
         t1->l = t->l;
```

```
    }
    else
    {
        t1->r = t->l;
    }
    t = NULL;
    free(t);
    return;
}

/* To delete node having right hand child */
else if (t->l == NULL)
{
    if (t1 == t)
    {
        root = t->r;
        t1 = root;
    }
    else if (t1->r == t)
        t1->r = t->r;
    else
        t1->l = t->r;
    t == NULL;
    free(t);
    return;
}

/* To delete node having two child */
else if ((t->l != NULL) && (t->r != NULL))
{
    t2 = root;
```

```
        if (t->r != NULL)
        {
            k = smallest(t->r);
            flag = 1;
        }
        else
        {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }

}

int smallest(struct btnode *t)
{
    t2 = t;
    if (t->l != NULL)
    {
        t2 = t;
        return(smallest(t->l));
    }
    else
        return (t->value);
}

int largest(struct btnode *t)
{
    if (t->r != NULL)
```

```
    {
        t2 = t;
        return(largest(t->r));
    }
    else
        return(t->value);
}

void inorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("NULL");
        return;
    }
    if (t->l != NULL)
        inorder(t->l);
    printf("%d \t", t->value);
    if (t->r != NULL)
        inorder(t->r);
}
void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d \t ", t->value);
    if (t->l != NULL)
        preorder(t->l);
```

```
    if (t->r != NULL)
        preorder(t->r);
}

void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d \t ", t->value);
}
```

OUTPUT :

```
   Choose
1. Insert a node to BST
2. Delete a node from the BST
3. Print in Pre-Order traversal form
4. Print in In-Order traversal form
5. Print in Post-Order traversal form
6. Exit

----------------X----------------
Enter your choice : 1
Enter data of node to be inserted : 20

----------------X----------------
Enter your choice : 1
Enter data of node to be inserted : 9

----------------X----------------
Enter your choice : 1
Enter data of node to be inserted : 24

----------------X----------------
Enter your choice : 1
Enter data of node to be inserted : 29

----------------X----------------
Enter your choice : 1
Enter data of node to be inserted : 3

----------------X----------------
Enter your choice : 1
Enter data of node to be inserted : 11

----------------X----------------
Enter your choice : 3
20       9       3       11      24      29
----------------X----------------
Enter your choice : 4
3       9       11      20      24      29
----------------X----------------
Enter your choice : 5
3        11      9       29      24      20
----------------X----------------
```

RESULT : The implementation is successfully done.

AIM : To implement checking of whether a Binary Search Tree(BST) is an AVL tree or not.

CODE :

```
//C Program to check a given Binary Tree is an AVL Tree or not

#include <stdio.h>
#include <stdlib.h>
//Declaring a node structure to represent the node in the BST
struct btnode
{
   int value;
   struct btnode *l;
   struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;

//Above, defining nodes root to always point at the root node of the BST
//a temporary node temp initialise to NULL
//nodes t1, t2 to represent left sub-tree and right sub-tree to be later used in the program

//Below, declaring functions to be used in the program

void delete1();
//To delete a node
void insert();
//To insert a node in the tree
void delete();
//To check whether node to be deleted is in left sub-tree or right sub-tree
void create();
//To create a node
void search(struct btnode *t);
//To search the appropriate position to insert a new node
```

```
void search1(struct btnode *t,int data);
//To search left sub-tree for the appropriate position to insert a new node
int smallest(struct btnode *t);
// To find the smallest element in the right sub tree
int largest(struct btnode *t);
// To find the largest element in the left sub tree
void inorder(struct btnode *t);
// A function to print in-order traversal of the BST
void preorder(struct btnode *t);
// A function to print in pre-order traversal of the BST
void postorder(struct btnode *t);
// A function to print in post-order traversal of the BST
int isAVL(struct btnode *t);
// The function to check whether the tree is AVL or not
int height(struct btnode *t);
// The function that return the height of a node

int flag = 1;

void main()
{
   int ch;

   printf("\n   Choose");
   printf("\n1. Insert a node to BST");
   printf("\n2. Delete a node from the BST");
   printf("\n3. Print in Pre-Order traversal form");
   printf("\n4. Print in In-Order traversal form");
   printf("\n5. Print in Post-Order traversal form");
   printf("\n6. Checking AVL tree or not");
   printf("\n7. Exit\n");
```

```
while(1)
  {
     printf("\nEnter your choice : ");
     scanf("%d", &ch);
     switch (ch)
     {
     case 1:
        insert();
        break;
     case 2:
        delete();
        break;
     case 3:
        preorder(root);
        break;
     case 4:
        inorder(root);
        break;
     case 5:
        postorder(root);
        break;
     case 6:
        if(isAVL(root))
        {
           printf("\nThe given tree is an AVL tree.");
        }
        else
        {
           printf("\nThe given tree is not an AVL tree.");
        }
        break;
```

```
        case 7:
            exit(0);
            break;
        default :
            printf("Enter valid choice");
            break;
        }
    }
}

void insert()
{
    create();
    if (root == NULL)
        root = temp;
    else
        search(root);
}

void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}
```

```
void search(struct btnode *t)
{
   if ((temp->value > t->value) && (t->r != NULL))      //Checking to insert at right
      search(t->r);
   else if ((temp->value > t->value) && (t->r == NULL))
      t->r = temp;
   else if ((temp->value < t->value) && (t->l != NULL))    //Checking to insert at left
      search(t->l);
   else if ((temp->value < t->value) && (t->l == NULL))
      t->l = temp;
}

void delete()
{
   int data;

   if (root == NULL)
   {
      printf("No elements in a tree to delete");
      return;
   }
   printf("Enter the data to be deleted : ");
   scanf("%d", &data);
   t1 = root;
   t2 = root;
   search1(root, data);
}

void search1(struct btnode *t, int data)
{
   if ((data>t->value))
```

```
    {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value))
    {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value))
    {
        delete1(t);
    }
}

void delete1(struct btnode *t)
{
    int k;

    /* To delete leaf node */
    if ((t->l == NULL) && (t->r == NULL))
    {
        if (t1->l == t)
        {
            t1->l = NULL;
        }
        else
        {
            t1->r = NULL;
        }
        t = NULL;
```

```
    free(t);
    return;
}

/* To delete node having one left hand child */
else if ((t->r == NULL))
{
    if (t1 == t)
    {
        root = t->l;
        t1 = root;
    }
    else if (t1->l == t)
    {
        t1->l = t->l;

    }
    else
    {
        t1->r = t->l;
    }
    t = NULL;
    free(t);
    return;
}

/* To delete node having right hand child */
else if (t->l == NULL)
{
    if (t1 == t)
    {
```

```
            root = t->r;
            t1 = root;
        }
        else if (t1->r == t)
            t1->r = t->r;
        else
            t1->l = t->r;
        t == NULL;
        free(t);
        return;
    }

    /* To delete node having two child */
    else if ((t->l != NULL) && (t->r != NULL))
    {
        t2 = root;
        if (t->r != NULL)
        {
            k = smallest(t->r);
            flag = 1;
        }
        else
        {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }

}
```

```
int smallest(struct btnode *t)
{
   t2 = t;
   if (t->l != NULL)
   {
      t2 = t;
      return(smallest(t->l));
   }
   else
      return (t->value);
}

int largest(struct btnode *t)
{
   if (t->r != NULL)
   {
      t2 = t;
      return(largest(t->r));
   }
   else
      return(t->value);
}

void inorder(struct btnode *t)
{
   if (root == NULL)
   {
      printf("NULL");
      return;
   }
   if (t->l != NULL)
```

```
        inorder(t->l);
    printf("%d \t", t->value);
    if (t->r != NULL)
        inorder(t->r);
}
void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d \t ", t->value);
    if (t->l != NULL)
        preorder(t->l);
    if (t->r != NULL)
        preorder(t->r);
}

void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d \t ", t->value);}
```

```
int isAVL(struct btnode *t)
{
	int h_l, h_r, diff;
	if(t == NULL)
		return 1;
	h_l = height(t->l);
	h_r = height(t->r);
	diff = h_l>h_r ? h_l-h_r : h_r-h_l;
	if( diff<=1 &&  isAVL(t->l) && isAVL(t->r) )
		return 1;
	return 0;
}

int height(struct btnode *t)
{
	int h_left, h_right;

	if (t == NULL)
		return 0;

	h_left =  height(t->l);
	h_right = height(t->r);

	if (h_left > h_right)
		return 1 + h_left;
	else
		return 1 + h_right;
}
```

OUTPUT :

```
   Choose
1. Insert a node to BST
2. Delete a node from the BST
3. Print in Pre-Order traversal form
4. Print in In-Order traversal form
5. Print in Post-Order traversal form
6. Checking AVL tree or not
7. Exit

Enter your choice : 1
Enter data of node to be inserted : 19

Enter your choice : 1
Enter data of node to be inserted : 12

Enter your choice : 1
Enter data of node to be inserted : 10

Enter your choice : 6

The given tree is not an AVL tree.
Enter your choice : 1
Enter data of node to be inserted : 21

Enter your choice : 6

The given tree is an AVL tree.
Enter your choice : 7

Process returned 0 (0x0)   execution time : 42.706 s
Press any key to continue.
```

RESULT : The implementation is successfully done.