INDEX

18	Write a program to implement Stack, using Linked List. Implement Push, Pop and display operations.	29-31	27/09/2023	
19	Write a program to implement Queue, using Linked List. Implement Insertion, deletion and display operations.	31-33	27/09/2023	
20	Write a program to count the number of times an item is present in a linked list.	34-35	27/09/2023	
21	Write a program to increment the data part of every node present in a linked list by 10. Display the data both before incrimination and after.	35-37	27/09/2023	
22	Write a program to implement Doubly Linked List, showing all the operations, like creation, display, insertion, deletion and searching.	38-42	04/10/2023	
23	Write program to create a Binary Search Tree and delete a node and display its contents using recursive preorder, postorder and inorder traversal	43	11/10/2023	

Q1. Write a program to search an element using Linear Search.

Program:

```c
#include <stdio.h>
#include<time.h>
int linearsearch(int ele,int n,int *arr){
  for(int i=0;i<n;i++){
    if(arr[i]==ele) return i;
  }
  return -1;
}
int main(void) {
  int n;
  printf("Enter the number of elements: ");
  scanf("%d",&n);
  int arr[n];
  printf("Enter the element: ");
  for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);
  }
  time_t start,end;
  start=time(NULL);
  int ele;
  printf("Enter the element to search: ");
  scanf("%d",&ele);
  int index=linearsearch(ele,n,arr);
  if(index==-1) printf("Element not found!!\n");
  else printf("Element occur at index %d\n",index);
  end=time(NULL);
  printf("Execution time for linear search is : %f 
seconds\n",difftime(end,start));
  return 0;
}
```

OUTPUT:

```
Enter the number of elements: 5

Enter the element: 1 4 2 6 5

Enter the element to search: 2

Element occur at index 2

Execution time for linear search is : 2.000000 seconds
```

Q2. Write a program to search an element using Binary Search.

Program:

```
#include <stdio.h>
#include<time.h>
int binarysearch(int ele,int n,int *arr){
  int s=0,e=n-1;
  int mid=s+(e-s)/2;
  while(s<=e){
    if(arr[mid]==ele) return mid;
    else if(arr[mid]>ele) e=mid-1;
    else if(arr[mid]<ele) s=mid+1;
    mid=s+(e-s)/2;
  }
  return -1;
}
int main(void) {
  int n;
  printf("Enter the number of elements: ");
  scanf("%d",&n);
  int arr[n];
  printf("Enter the element: ");
  for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);
  }
  time_t start,end;
  start=time(NULL);
  int ele,index;
  printf("Enter the element to search: ");
  scanf("%d",&ele);
  index=binarysearch(ele,n,arr);
  if(index==-1) printf("Element not found!!\n");
  else printf("Element occur at index %d\n",index);
  end=time(NULL);
  printf("Execution time for binary search is : %f
seconds",difftime(end,start));
  return 0;
}
```

OUTPUT:

```
Enter the number of elements: 5

Enter the element: 1 2 3 4 5

Enter the element to search: 4

Element occur at index 3

Execution time for binary search is : 2.000000 seconds
```

Q3. Write a program to sort the given array using Bubble Sort.

Program:

```c
#include <stdio.h>
int swap(int *a,int *b){
  int temp=*a;
  *a=*b;
  *b=temp;
}
int main(void) {
  int n;
  printf("Enter the size: ");
  scanf("%d",&n);
  int arr[n];
  printf("Enter the array: ");
  for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);
  }
  int j=n;
  while(j>=0){
    for(int i=0;i<j-1;i++){
      if(arr[i]>arr[i+1]) swap(&arr[i],&arr[i+1]);
    }
    j--;
  }
  printf("Sorted array is: ");
  for(int i=0;i<n;i++){
    printf("%d\t",arr[i]);
  }
  return 0;
}
```

Output:

```
Enter the size: 5

Enter the array: 5 2 7 3 1

Sorted array is: 1	2	3	5	7
```

Q8. Write a program to sort the given array using Selection Sort.
Program:

```c
#include <stdio.h>
void swap(int *a,int*b){
  int temp=*a;
  *a=*b;
  *b=temp;
}
int main(void) {
  int n;
  printf("Enter the number of elements: ");
  scanf("%d",&n);
  int arr[n];
  for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);
  }
  for(int i=0;i<n;i++){
    int min=i;
    for(int j=i;j<n;j++){
      if(arr[min]>arr[j]) min =j;
    }
    swap(&arr[min],&arr[i]);
  }
  for(int i=0;i<n;i++){
    printf("%d\t",arr[i]);
  }
  return 0;
}
```

Output:

```
Enter the number of elements: 5

5 2 7 3 1

1    2    3    5    7
```

Q5. Write a program to sort the given array using Insertion Sort.

Program:

```
#include<stdio.h>
void swap(int *a,int*b){
  int temp=*a;
  *a=*b;
  *b=temp;
}
int main(void) {
  int n;
  printf("Enter the number of elements: ");
  scanf("%d",&n);
  int arr[n];
  printf("Enter the elements: ");
  for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);
  }
  for(int i=1;i<n;i++){
    for(int j=i;j>0;j--){
      if(arr[j-1]>arr[j]) swap(&arr[j],&arr[j-1]);
    }
  }
  for(int i=0;i<n;i++){
    printf("%d\t",arr[i]);
  }
  return 0;
}
```

Output:

```
Enter the number of elements: 5

Enter the elements: 5 2 7 1 9

1    2    5    7    9
```

Q6. Write a program to insert an element inside a array at particular index.
Program:

```c
#include <stdio.h>
#define max 100
void insert(int *a,int pos,int item,int *n){
  if(max==*n)printf("Array is prefilled\n");
  else{
    for(int i=*n;i>pos-1;i--){
      a[i]=a[i-1];
    }
    a[pos-1]=item;
    *n=*n+1;
  }
}
int main(void) {
  int size;
  int n;
  printf("Enter the size of the array: ");
  scanf("%d",&n);
  int a[n],i;
  for(i=0;i<n;i++){
     scanf(" %d",&a[i]);
  }
  int item,pos;
  char choice;
  for(;;){
    printf("Want to continue(y/n): ");
    scanf(" %c",&choice);
    if (choice!='y') break;
    else{
      printf("Enter item to be inserted: ");
      scanf(" %d",&item);
      printf("Enter the postion: ");
      scanf(" %d",&pos);
      insert(a,pos,item,&n);
    }
  }
  printf("After Insertion: ");
  for(i=0;i<n;i++){
     printf("%d\t",a[i]);
  }
  return 0;
}
```

Output:

```
Enter the size of the array: 5
3 5 2 6 1
Want to continue(y/n): y
Enter item to be inserted: 8
Enter the position: 2
Want to continue(y/n): n
After Insertion: 3     8     5     2     6     1
```

Q7. Write a program to delete an element from the array.

Program:

```c
#include <stdio.h>
int main() {
  int n;
  printf("Enter size of array: ");
  scanf("%d",&n);
  int a[n],i,index=-1;
  for(i=0;i<n;i++){
     scanf("%d",&a[i]);
  }
  int item;
  printf("enter item to be deleted: ");
  scanf("%d",&item);
  //Searching for index of item
  for(i=0;i<n;i++){
    if(a[i]==item){
     index=i;
    }
  }
  if (index==-1) printf("Element not present in the array\n");
  else{
    n--;
    for(i=index;i<n;i++){
      a[i]=a[i+1];
    }
  }
  printf("After deletion : ");
  for(i=0;i<n;i++){
    printf("%d\t",a[i]);
  }
  return 0;
}
```

Output:

```
Enter size of array: 5
4 5 2 3 1
enter item to be deleted: 2
After deletion : 4	5	3	1
```

Q8. Write a program to sort the given array using Quick Sort.

Program:

```
#include <stdio.h>
void swap(int *a,int *b){
  int temp=*a;
  *a=*b;
  *b=temp;}
int partion(int arr[],int s,int e){
  int count_smaller=0;
  s--;
  for(int i=s;i<e;i++){
    if(arr[i]<arr[e]) count_smaller++;}
  int pivot_index=s+count_smaller;
  swap(&arr[e],&arr[pivot_index]);
  int i=s,j=e;
  while(i<=pivot_index&&j>=pivot_index){
    if(arr[i]>=arr[pivot_index]&&arr[j]<=arr[pivot_index])
swap(&arr[i],&arr[j]);
    i++;
    j--;
  }
  return (pivot_index);
}
void quicksort(int arr[],int s,int e){
  if(s>=e) return;
  else{
    int p=partion(arr,s,e);
    quicksort(arr,s,p-1);
    quicksort(arr,p+1,e);
  }
}
int main(void) {
  int size;
  printf("Enter size: ");
  scanf("%d",&size);
  int arr[size];
  printf("Enter the elements: ");
  for(int i=0;i<size;i++){
    scanf("%d",&arr[i]);
  }
  quicksort(arr,0,size-1);
  printf("Printing the sorted array: ");
  for(int i=0;i<size;i++) printf("%d\t",arr[i]);
  return 0;
}
```

Output:

```
Enter size of array: 5
4 5 2 3 1
enter item to be deleted: 2
After deletion : 4     5     3     1
```

Q9. Write a program to sort the given array using merge Sort.

Program:

```
#include <stdio.h>
void merge(int arr[], int s, int mid, int e) {
    int n1 = mid - s + 1, n2 = e - mid;
    int L[n1], R[n2];
    for(int i = 0; i < n1; i++) L[i] = arr[s + i];
    for(int i = 0; i < n2; i++) R[i] = arr[mid + i + 1];
    int i = 0, j = 0, k = s;
    while(i < n1 && j < n2) {
        if(L[i] < R[j]) {
            arr[k] = L[i];
            i++;}
        else {
            arr[k] = R[j];
            j++;}
        k++;}
    while(i < n1) {
        arr[k] = L[i];
        i++;
        k++;}
    while(j < n2) {
        arr[k] = R[j];
        j++;
        k++;}
    return;}
void mergesort(int arr[], int s, int e) {
    if (s >= e) return;
    else {
        int mid = s + (e - s) / 2;
        mergesort(arr, s, mid);
        mergesort(arr, mid + 1, e);
        merge(arr, s, mid, e);}}
int main(void) {
    int size;
    printf("Enter the size: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements: ");
    for(int i = 0; i < size; i++) scanf("%d", &arr[i]);
    mergesort(arr, 0, size - 1);
    printf("After sorting: ");
    for(int i = 0; i < size; i++) printf("%d\t", arr[i]);
    return 0;
}
```

Output:

```
Enter the size: 5
Enter the elements: 3 1 5 2 4
After sorting: 1	2	3	4	5
```

Q10. Write a program to implement stack using Array.
Program:

```c
#include <stdio.h>
#include <stdbool.h>
#define N 7
int stack[N];
int top=-1;
bool isEmpty(){
  if(top==-1) return true;
  return false;
}
bool isFull(){
  if(top==N-1) return true;
  return false;
}
void push(){
  if(isFull()){
    printf("Stack overflow\n");
    return;
  }
  int ele;
  printf("Enter the element: ");
  scanf("%d",&ele);
  stack[++top]=ele;
}
void pop(){
  if(isEmpty()){
    printf("Stack underflow\n");
    return;
  }
  printf("Deleted element: %d\n",stack[top]);
  top--;
}
void display(){
  if(isEmpty()){
    printf("Stack is empty");
    return;
  }
  printf("The elements are: ");
  for(int i=0;i<=top;i++) printf("%d\t",stack[i]);
  printf("\n");
}
int main(void) {
  char c='y';
  int choice;
  printf("Enter 1 for insertion\n");
  printf("Enter 2 for deletion\n");
  printf("Enter 3 for Display\n");
  while(c=='y'){
    printf("\nEnter your choice: ");
    scanf(" %d",&choice);
```

```
        switch(choice){
          case 1:
            push();
            break;
          case 2:
            pop();
            break;
          case 3:
            display();
            break;
          default:
            printf("Invalid choice entered\n");
            break;
        }
        printf("Want to continue (y/n): ");
        scanf(" %c",&c);
      }
      printf("\nProgram ends!!!");
      return 0;
    }
```

Output:

```
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for Display

Enter your choice: 1
Enter the element: 3
Want to continue (y/n): y

Enter your choice: 1
Enter the element: 4
Want to continue (y/n): y

Enter your choice: 3
The elements are: 3     4
Want to continue (y/n): y

Enter your choice: 2
Deleted element: 4
Want to continue (y/n): y

Enter your choice: 2
Deleted element: 3
Want to continue (y/n): y

Enter your choice: 2
Stack underflow
Want to continue (y/n): n

Program ends!!!
```

Q11. Write a program to reverse a stack without recursion.
Program:

```c
#include <stdio.h>
#include <stdbool.h>
#define N 7
int stack[N];
int top=-1;
bool isEmpty(){
  if(top==-1) return true;
  return false;}
bool isFull(){
  if(top==N-1) return true;
  return false;}
void push(){
  if(isFull()){
    printf("Stack overflow\n");
    return;}
  int ele;
  printf("Enter the element: ");
  scanf("%d",&ele);
  stack[++top]=ele;}
void pop(){
  if(isEmpty()){
    printf("Stack underflow\n");
    return;}
  printf("Deleted element: %d\n",stack[top]);
  top--;}
void display(){
  if(isEmpty()){
    printf("Stack is empty");
    return;}
  printf("The elements are: ");
  for(int i=0;i<=top;i++) printf("%d\t",stack[i]);
  printf("\n");}
void reverse(){
  int temp[top+1];
  int i=top;
  for(int j=0;top>=0;j++) temp[j]=stack[top--];
  for(int j=0;j<=i;j++) stack[++top]=temp[j];
  return;}
int main(void) {
  char c='y';
  int choice;
  printf("Enter 1 for insertion\n");
  printf("Enter 2 for deletion\n");
  printf("Enter 3 for Display\n");
  printf("Enter 4 for Reverse\n");
  while(c=='y'){
    printf("\nEnter your choice: ");
    scanf(" %d",&choice);
    switch(choice){
      case 1:
        push();
        break;
      case 2:
        pop();
        break;
      case 3:
        display();
        break;
      case 4:
        reverse();
        break;
      default:
        printf("Invalid choice entered\n");
        break;
    }
    printf("Want to continue (y/n): ");
    scanf(" %c",&c);
  }
  printf("\nProgram ends!!!");
  return 0;}
```

Output:

```
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for Display
Enter 4 for Reverse

Enter your choice: 1
Enter the element: 4
Want to continue (y/n): y

Enter your choice: 1
Enter the element: 3
Want to continue (y/n): y

Enter your choice: 1
Enter the element: 5
Want to continue (y/n): y

Enter your choice: 3
The elements are: 4    3    5
Want to continue (y/n): y

Enter your choice: 4
Want to continue (y/n): y

Enter your choice: 3
The elements are: 5    3    4
Want to continue (y/n): n

Program ends!!!
```

Q12. Write a program to implement queue using array.
Program:

```c
#include <stdio.h>
#include <stdbool.h>
#define size 7
int front=-1,rear=-1;
int queue[size];
bool isFull(){
  if(rear==size-1) return true;
  return false;
}
bool isEmpty(){
  if (front==-1&&rear==-1) return true;
  return false;
}
void enqueue(){
  if(isFull()){
    printf("Queue overflow\n");
    return;
  }
  int ele;
  printf("Enter the element: ");
  scanf("%d",&ele);
  if(isEmpty()){
    front++;
    rear++;
    queue[front]=ele;
  }
  else queue[++rear]=ele;
  return;
}
void dequeue(){
  if(isEmpty()){
    printf("Queue underflow\n");
    return;
  }
  int ele;
  if(front==rear){
    ele=queue[rear];
    front=rear=-1;
  }
  else ele=queue[front++];
  printf("Deleted element is %d\n",ele);
  return;
}
void display(){
  if(isEmpty()){
    printf("Queue is empty\n");
    return;
  }
  else{
    printf("The elements are: ");
    for(int i=front;i<=rear;i++) printf("%d\t",queue[i]);
  }
  printf("\n");
  return;
}
int main(void) {
  printf("Enter 1 for Enqueue operation\n");
  printf("Enter 2 for Dequeue operation\n");
  printf("Enter 3 for Display operation\n");
  char c='y';
  int choice;
  while(c=='y'){
    printf("\nEnter the choice: ");
    scanf(" %d",&choice);
    switch(choice){
      case 1:
        enqueue();
        break;
      case 2:
        dequeue();
        break;
      case 3:
        display();
        break;
      default:
        printf("Invalid choice!!\n");
        break;
    }
    printf("Want to continue(y/n): ");
    scanf(" %c",&c);
  }
  printf("\nPrograms ends!!!");
  return 0;
}
```

Output:

```
Enter 1 for Enqueue operation
Enter 2 for Dequeue operation
Enter 3 for Display operation

Enter the choice: 3
Queue is empty
Want to continue(y/n): y

Enter the choice: 2
Queue underflow
Want to continue(y/n): y

Enter the choice: 1
Enter the element: 5
```

```
Want to continue(y/n): y

Enter the choice: 1
Enter the element: 8
Want to continue(y/n): y

Enter the choice: 3
The elements are: 5     8
Want to continue(y/n): y

Enter the choice: 2
Deleted element is 5
Want to continue(y/n): y

Enter the choice: 2
Deleted element is 8
Want to continue(y/n): y

Enter the choice: 2
Queue underflow
Want to continue(y/n): y

Enter the choice: 3
Queue is empty
Want to continue(y/n): n

Programs ends!!!
```

Lab : 13

Date: 13-Sep-2023

Objective: - Write a program to implement circular queue using array.

Equipment/Software used: A computer with a smooth internet connection.

Theory: A circular queue is a data structure that implements the queue abstract data type (ADT). Unlike a linear queue, a circular queue does not have a fixed front and rear end. Instead, the queue is implemented as a circular array, where the rear pointer points to the next element to be inserted, and the front pointer points to the next element to be removed.

To implement a circular queue using an array, we need to initialize two variables:

front: The index of the front element of the queue.

rear: The index of the rear element of the queue.

Initially, both front and rear should be set to -1, indicating that the queue is empty.

To enqueue an element into the queue, we increment the rear pointer by 1, and then insert the element into the array at the rear index. If the rear pointer reaches the end of the array, we wrap around to the beginning of the array.

To dequeue an element from the queue, we increment the front pointer by 1, and then return the element at the front index. If the front pointer reaches the rear pointer, it indicates that the queue is empty.

Program:

```c
#include <stdio.h>
#define n 5
int arr[n];
int front=-1,rear=-1;
void enqueue(int num){
  if (front==-1&&rear==-1){
    front++;
    rear++;
    arr[rear]=num;
  }
  else if((rear+1)%n==front) printf("Queue overflow\n");
  else{
    rear=(rear+1)%n;
    arr[rear]=num;
  }
}
void dequeue(){
  if(front==-1&&rear==-1) printf("Queue underflow\n");
  else if(front==rear){
    front=-1;
    rear=-1;
  }
  else front=(front+1)%n;
}
void display(){
  if(front==-1 && rear==-1) printf("Queue is empty");
  else{
    int i=front;
```

```
      while(i!=rear){
        printf("%d\t",arr[i]);
        i=(i+1)%n;
      }
      printf("%d\t",arr[i]);
    }
    printf("\n");
  }
  void peek(){
    if(front==-1&&rear==-1) printf("Queue is empty");
    else printf("%d",arr[front]);
    printf("\n");
  }
  int main(void) {
    printf("Welcome\n");
    char choice='y';
    int select,ele;
    printf("Enter 1 for enqueue\n");
    printf("Enter 2 for dequeue\n");
    printf("Enter 3 for Display\n");
    printf("Enter 4 for peek\n");
    while(choice=='y'){
      printf("\nEnter the choice: ");
      scanf("%d",&select);
      switch(select){
        case 1:
          printf("Enter the element: ");
          scanf("%d",&ele);
          enqueue(ele);
          break;
        case 2:
          dequeue();
          break;
        case 3:
          display();
          break;
        case 4:
          peek();
          break;
        default:
          printf("Wrong choice entered!!\n");
      }
      printf("Want to continue(y/n): ");
      scanf(" %c",&choice);
    }
    return 0;
  }
```

Output:

```
Welcome
Enter 1 for enqueue
Enter 2 for dequeue
Enter 3 for Display
Enter 4 for peek

Enter the choice: 3
Queue is empty
Want to continue(y/n): y

Enter the choice: 1
Enter the element: 1
Want to continue(y/n): y

Enter the choice: 1
Enter the element: 3
Want to continue(y/n): y

Enter the choice: 3
1      3
Want to continue(y/n): y

Enter the choice: 4
1
Want to continue(y/n): y

Enter the choice: 2
Want to continue(y/n): y

Enter the choice: 2
Want to continue(y/n): y

Enter the choice: 3
Queue is empty
Want to continue(y/n): n
```

LAB- 14

Date: 13-Sep-2023

Objective: - Given an array A with N-1 elements, with elements from 1 to N present into it. Find missing element.

Theory:- The sum of the first N natural numbers is N*(N+1)/2. We can use this formula to find the sum of all the elements in the array, and then subtract it from the sum of the first N natural numbers. The difference will be the missing element.

Program:

```
#include <stdio.h>
int main(void) {
  int n,sum=0;
  printf("Enter the size of array: ");
  scanf("%d",&n);
  int arr[n],actual_sum=n*(n+1)/2;
  printf("Enter the elements: ");
  for(int i=0;i<n-1;i++){
    scanf("%d",&arr[i]);
    sum=sum+arr[i];
  }
  printf("Missing element is: %d",actual_sum-sum);
  return 0;
}
```

Output:

```
Enter the size of array: 5
Enter the elements: 1 2 3 5
Missing element is: 4
```

LAB- 15

Date: 13-Sep-2023

Objective: -Given an array A of N elements, your task is to print all those indexes that have values greater than its left and right neighbors. In case of extreme indexes like 0, N-1 check with their single neighbors.

Theory:- To print all those indexes in an array A of N elements that have values greater than its left and right neighbors, we can use the following algorithm:

Iterate over the array from index 1 to index N - 2.

For each element, check if it is greater than its left and right neighbors.

If the element is greater than its left and right neighbors, then print its index.

Program:

```
#include <stdio.h>
int main(void) {
  int n;
  printf("Enter the size of array: ");
  scanf("%d",&n);
  int arr[n];
  printf("Enter the elements: ");
  for(int i=0;i<n-1;i++) scanf("%d",&arr[i]);
  printf("Required indices are: ");
  for(int i=0;i<n-1;i++){
    if(i==0){
      if(arr[i]>arr[i+1]) printf("%d\t",i);
      continue;
    }
    else if(i==n-1){
      if(arr[i]>arr[i-1]) printf("%d\t",i);
      continue;
    }
    else{
      if(arr[i]>arr[i+1]&&arr[i]>arr[i-1]) printf("%d\t",i);
    }
  }
  return 0;
}
```

Output:

```
Enter the size of array: 7
Enter the elements: 9 8 11 14 21 1 5
Required indices are: 0 4
```

LAB- 16

Date: 27-Sep-2023

Objective: -Given a unsorted integer array, find a triplet with a given sum in it.

Theory:- We are checking every combination of three elements with the help of three nested loop and checking it with given sum.

Program:

```
#include <stdio.h>
int main(void) {
  int n,sum;
  printf("Enter the size of array: ");
  scanf("%d",&n);
  int arr[n];
  printf("Enter the elements: ");
  for(int i=0;i<n;i++) scanf("%d",&arr[i]);
  printf("Enter the triplet sum: ");
  scanf(" %d",&sum);
  for(int i=0;i<n-2;i++){
    for(int j=i+1;j<n-1;j++){
      for(int k=j+1;k<n;k++){
        if(arr[i]+arr[j]+arr[k]==sum){
          printf("\nTriplet is: %d %d %d",arr[i],arr[j],arr[k]);
        }
      }
    }
  }
  return 0;
}
```

Output:

```
Enter the size of array: 6
Enter the elements: 45 1 8 10 6 4
Enter the triplet sum: 22

Triplet is: 8 10 4
```

LAB- 17

Date: 27-Sep-2023

Objective: -Write a program to implement Linear Linked List, showing all the operations, like creation, display, insertion, deletion and searching.

Theory:- A linear linked list is a data structure that consists of a sequence of nodes, where each node contains a data element and a pointer to the next node in the sequence. The last node in the list has a pointer to None to indicate the end of the list.

To implement a linear linked list, we need to define a Node class and a LinkedList class. The Node class should have two attributes: data and next. The LinkedList class should have one attribute: head, which points to the first node in the list.

Creation:- To create a new linked list, we can simply create an instance of the LinkedList class. The head attribute of the new linked list will be set to None to indicate that the list is empty.

Display:- To display the contents of a linked list, we can iterate over the list and print the data element of each node.

Program:

```
#include <stdio.h>
#include<stdlib.h>
struct node{
  int data;
  struct node*next;
};
void InsertAtHead(struct node**head){
  int d;
  printf("Enter the data: ");
  scanf("%d",&d);
  struct node* temp;
  temp=(struct node*)(malloc(sizeof(struct node)));
  temp->data=d;
  if(*head==NULL){
    temp->next=NULL;
    *head=temp;
  }
  else{
    temp->next=(*head);
    *head=temp;
  }
  return;
}
void InsertAtTail(struct node**head){
  if(*head==NULL){
    InsertAtHead(head);
    return;
  }
  struct node* newnode;
  struct node* temp;
  newnode=(struct node*)(malloc(sizeof(struct node)));
  int element;
  printf("Enter the element: ");
  scanf("%d",&element);
  newnode->data=element;
```

```
    newnode->next=NULL;
    temp=*head;
    while(temp->next!=NULL){
      temp=temp->next;
    }
    temp->next=newnode;
    return;
  }
  void InsertAtPos(struct node **head){
    if(*head==NULL){
      InsertAtHead(head);
      return;
    }
    struct node *temp=*head;
    int pos,t=1,element;
    printf("Enter the position to enter: ");
    scanf("%d",&pos);
    while(temp->next!=NULL&&t<pos-1){
      temp=temp->next;
      t++;
    }
    printf("Enter the element: ");
    scanf("%d",&element);
    struct node* newnode;
    newnode=(struct node *)(malloc(sizeof(struct node)));
    newnode->data=element;
    newnode->next=temp->next;
    temp->next=newnode;
  }
  void deletion(struct node**head){
    if(*head==NULL){
      printf("List is empty\n");
      return;
    }
    int element;
    printf("Enter the element to delete: ");
    scanf("%d",&element);
    struct node*prev=NULL,*curr=*head;
    if(curr->data==element){//Handling first element
      *head=(*head)->next;
      free(curr);
      return;
    }
    while(curr!=NULL&&curr->data!=element){
      prev=curr;
      curr=curr->next;
    }
    if(curr==NULL){
      printf("Element not found!!!\n");
      return;
    }
    prev->next=curr->next;
    free(curr);
    return;
  }
  void display(struct node *head){
    if(head==NULL){
      printf("Linked list has no elements!!!\n");
      return;
    }
    printf("The elements are: ");
    while(head!=NULL){
      printf("%d\t",head->data);
      head=head->next;
    }
    printf("\n");
  }
  void search(struct node* head){
    int element,pos=1,check=0;
    printf("Enter the element to search: ");
    scanf("%d",&element);
    while(head!=NULL){
      if(head->data==element){
        printf("Element found at position %d\n",pos);
        check=1;
        return;
      }
      head=head->next;
      pos++;
    }
    if(check==0){
      printf("ELement not found!!\n");
      return;
    }
  }
  int main(void) {
    printf("Welcome\n");
    struct node *head=NULL;
    char ch='y';
    int choice;
    printf("Enter 1 for insertion at beginning\n");
    printf("Enter 2 for insertion at tail\n");
    printf("Enter 3 for insertion at pos\n");
    printf("Enter 4 for deletion\n");
    printf("Enter 5 for display\n");
    printf("Enter 6 for searching\n");
    while(ch=='y'){
      printf("Enter the choice: ");
      scanf("%d",&choice);
      switch(choice){
        case 1:
          InsertAtHead(&head);
          break;
        case 2:
```

```
        InsertAtTail(&head);
        break;
      case 3:
        InsertAtPos(&head);
        break;
      case 4:
        deletion(&head);
        break;
      case 5:
        display(head);
        break;
      case 6:
        search(head);
        break;
      default:
        printf("Enter choice entered!!\n");
    }
    printf("\nWant to continue(y/n): ");
    scanf(" %c",&ch);
  }
  printf("Program executed successfully!!!");
  return 0;
}
```

Output:

```
Welcome
Enter 1 for insertion at beginning
Enter 2 for insertion at tail
Enter 3 for insertion at pos
Enter 4 for deletion
Enter 5 for display
Enter 6 for searching
Enter the choice: 1
Enter the data: 1

Want to continue(y/n): y
Enter the choice: 2
Enter the element: 2

Want to continue(y/n): y
Enter the choice: 1
Enter the data: 0

Want to continue(y/n): y
Enter the choice: 5
The elements are: 0    1     2

Want to continue(y/n): y
Enter the choice: 3
Enter the position to enter: 2
Enter the element: 5
```

```
Want to continue(y/n): y
Enter the choice: 5
The elements are: 0    5      1      2

Want to continue(y/n): y
Enter the choice: 6
Enter the element to search: 5
Element found at position 2

Want to continue(y/n): y
Enter the choice: 6
Enter the element to search: 3
ELement not found!!

Want to continue(y/n): y
Enter the choice: 4
Enter the element to delete: 5

Want to continue(y/n): y
Enter the choice: 5
The elements are: 0    1      2

Want to continue(y/n): n
Program executed successfully!!!
```

LAB- 18

Date: 27-Sep-2023

Objective: -Write a program to implement stack using Linear Linked List, showing all the operations, like push,pop,display.

Theory:- Create a Node class to represent the nodes in the linked list. Each node should have two attributes: data and next. The data attribute will store the element of the stack, and the next attribute will point to the next node in the list.
Create a Stack class to represent the stack. The Stack class should have one attribute: head, which points to the top node in the stack.
Implement the following operations on the Stack class:
push(): This operation adds a new element to the top of the stack. To do this, we create a new node with the given element and set the next attribute of the new node to the current top node of the stack. We then set the head attribute of the stack to the new node.
pop(): This operation removes the top element from the stack and returns it. To do this, we store the data element of the top node in the stack. We then set the head attribute of the stack to the next node in the list. Finally, we return the stored data element.
display(): This operation prints the contents of the stack to the console. To do this, we iterate over the linked list and print the data element of each node.

Program:

```c
#include <stdio.h>
#include<stdlib.h>
struct node{
  int data;
  struct node* next;
};
struct node* insert(struct node** top){
  struct node* newnode;
  newnode=(struct node*)malloc(sizeof(struct node));
  int element;
  printf("Enter the element: ");
  scanf("%d",&element);
  newnode->data=element;
  newnode->next=*top;
  *top=newnode;
  return *top;
}
struct node* deletion(struct node** top){
  if (*top==NULL){
    printf("Stack underflow!!!\n");
    return *top;
  }
  struct node* newnode;
  newnode=*top;
  *top=(*top)->next;
  free(newnode);
  return *top;
}
void display(struct node* top){
  if(top==NULL){
    printf("Stack is empty!!!\n");
    return;
  }
  printf("Elements are: ");
  while(top!=NULL){
    printf("%d\t",top->data);
    top=top->next;
  }
  printf("\n");
  return;
}
int main(void) {
  struct node *top=NULL;
  char ch='y';
  int choice;
  printf("Enter 1 for insertion\n");
  printf("Enter 2 for deletion\n");
  printf("Enter 3 for traversal\n");
  while(ch=='y'){
    printf("Enter the choice: ");
    scanf("%d",&choice);
    switch(choice){
      case 1:
        insert(&top);
        break;
      case 2:
        deletion(&top);
        break;
      case 3:
        display(top);
        break;
      default:
        printf("Invalid choice entered!!!\n");
    }
    printf("\nWant to continue(y/n): ");
    scanf(" %c",&ch);
  }
  printf("Program executed succesfully!!!");
  return 0;
}
```

Output:

```
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for traversal
Enter the choice: 1
Enter the element: 3

Want to continue(y/n): y
Enter the choice: 1
Enter the element: 5

Want to continue(y/n): y
Enter the choice: 3
Elements are: 5 3
```

```
Want to continue(y/n): y
Enter the choice: 2

Want to continue(y/n): y
Enter the choice: 3
Elements are: 3

Want to continue(y/n): n
Program executed succesfully!!!
```

LAB- 19

Date: 27-Sep-2023

Objective: -Write a program to implement queue using Linear Linked List, showing all the operations, like push,pop,display.

Theory:- A queue is a data structure that follows the First In First Out (FIFO) principle. This means that the first element added to the queue is the first element to be removed. Queues can be implemented using a variety of data structures, including arrays, linked lists, and dynamic arrays.

To implement a queue using a linear linked list, we can use the following steps:
Create a Node class to represent the nodes in the linked list. Each node should have two attributes: data and next. The data attribute will store the element of the queue, and the next attribute will point to the next node in the list.
Create a Queue class to represent the queue. The Queue class should have two attributes: front and rear. The front attribute will point to the first node in the queue, and the rear attribute will point to the last node in the queue.
Implement the following operations on the Queue class:
enqueue(): This operation adds a new element to the back of the queue. To do this, we create a new node with the given element and set the next attribute of the new node to None. We then set the rear attribute of the queue to the new node. If the front attribute of the queue is None, we also set the front attribute to the new node.
dequeue(): This operation removes the element at the front of the queue and returns it. To do this, we store the data element of the first node in the queue. We then set the front attribute of the queue to the next node in the list. If the front attribute of the queue becomes None, we also set the rear attribute to None.
display(): This operation prints the contents of the queue to the console. To do this, we iterate over the linked list and print the data element of each node.

Program:

```c
#include <stdio.h>
#include<stdlib.h>
struct node{
  int data;
  struct node *next;
};
struct node *front=NULL,*rear=NULL;
void enqueue(){
  struct node* temp=(struct node*)malloc(sizeof(struct node));
  printf("Enter the data: ");
```

```
    scanf("%d",&temp->data);
    temp->next=NULL;
    if(front==NULL&&rear==NULL){
      rear=front=temp;
    }
    else{
      rear->next=temp;
      rear=temp;
    }
  }
  void dequeue(){
    struct node* temp=front;
    if(front==NULL &&rear==NULL){
      printf("Queue underflow\n");
      return;
    }
    else if(front==rear){
      front=rear=NULL;
    }
    else{
      front=front->next;
    }
    free(temp);
  }
  void display(){
    if(front==NULL && rear==NULL) printf("Queue is empty");
    else{
      struct node* temp=front;
      while(temp!=NULL){
        printf("%d\t",temp->data);
        temp=temp->next;
      }
    }
    printf("\n");
  }
  void peek(){
    if (front==NULL&& rear==NULL) printf("Queue is empty\n");
    else printf("%d\n",front->data);
  }
  int main(void) {
    char choice='y';
    printf("Enter 1 for enqueue\n");
    printf("Enter 2 for dequeue\n");
    printf("Enter 3 for display\n");
    printf("Enter 4 for peek\n");
    while(choice=='y'){
      int select;
      printf("\nEnter the choice: ");
      scanf("%d",&select);
      switch(select){
        case 1:
          enqueue();
          break;
        case 2:
          dequeue();
          break;
        case 3:
          display();
          break;
        case 4:
          peek();
          break;
        default:
          printf("Wrong choice entered\n");
      }
      printf("Want to continue(y/n): ");
      scanf(" %c",&choice);
    }
    printf("Sucessfully executed\n");
    return 0;
}
```

Output:

```
Enter 1 for enqueue
Enter 2 for dequeue
Enter 3 for display
Enter 4 for peek

Enter the choice: 1
Enter the data: 5
Want to continue(y/n): y

Enter the choice: 1
Enter the data: 3
Want to continue(y/n): y

Enter the choice: 3
5     3
Want to continue(y/n): y

Enter the choice: 4
5
Want to continue(y/n): y

Enter the choice: 2
Want to continue(y/n): y

Enter the choice: 3
3
Want to continue(y/n): n
Sucessfully executed
```

LAB- 20

Date: 27-Sep-2023

Objective: -Write a program to count number of times an item is present in the linked list.

Theory:- To count the number of times an item is present in a linked list, we can use the following algorithm:

Start at the head of the linked list.

Iterate over the linked list, comparing each node's data to the given item.

If the node's data matches the given item, increment a counter.

Repeat steps 2 and 3 until the end of the linked list is reached.

Return the value of the counter.

Program:

```
#include <stdio.h>
#include<stdlib.h>
struct node{
  int data;
  struct node*next;
};
void InsertAtHead(struct node**head){
  int d;
  printf("Enter the data: ");
  scanf("%d",&d);
  struct node* temp;
  temp=(struct node*)(malloc(sizeof(struct node)));
  temp->data=d;
  if(*head==NULL){
    temp->next=NULL;
    *head=temp;
  }
  else{
    temp->next=(*head);
    *head=temp;
  }
  return;
}
void count(struct node* head){
  int count=0,item;
  printf("Enter the item to search: ");
  scanf("%d",&item);
  while(head!=NULL){
    if(head->data==item)count++;
    head=head->next;
  }
  printf("Item %d occur %d times\n",item,count);
}
int main(void) {
  printf("Welcome\n");
  struct node *head=NULL;
  char ch='y';
  while(ch=='y'){
    InsertAtHead(&head);
    printf("\nWant to continue(y/n): ");
    scanf(" %c",&ch);
  }
  count(head);
  printf("Program executed successfully!!!");
  return 0;
}
```
Output:

```
Welcome
Enter the data: 3

Want to continue(y/n): y
Enter the data: 3

Want to continue(y/n): y
Enter the data: 5

Want to continue(y/n): y
Enter the data: 7

Want to continue(y/n): n
Enter the item to search: 3
Item 3 occur 2 times
Program executed successfully!!!
```

LAB- 21

Date: 27-Sep-2023

Objective: -Write a program to increment the data part of every node present in the linked list by 10. Display the data before and after the incrementation.

Theory: To increment the data part of every node present in the linked list by 10, we can use the following algorithm:

1. Start at the head of the linked list.
2. Iterate over the linked list, incrementing the data of each node by 10.
3. Repeat step 2 until the end of the linked list is reached.

Program:

```
#include <stdio.h>
#include<stdlib.h>
struct node{
  int data;
  struct node*next;
};
void InsertAtHead(struct node**head){
  int d;
  printf("Enter the data: ");
  scanf("%d",&d);
  struct node* temp;
  temp=(struct node*)(malloc(sizeof(struct node)));
  temp->data=d;
  if(*head==NULL){
    temp->next=NULL;
    *head=temp;
  }
  else{
    temp->next=(*head);
    *head=temp;
  }
  return;
}

void display(struct node *head){
  if(head==NULL){
    printf("Linked list has no elements!!!\n");
    return;
  }
  printf("The elements are: ");
  while(head!=NULL){
    printf("%d\t",head->data);
    head=head->next;
  }
  printf("\n");
}
void increment(struct node *head){
  while(head!=NULL){
    head->data=head->data+10;
    head=head->next;
  }
  return;
}
int main(void) {
  printf("Welcome\n");
  struct node *head=NULL;
  char ch='y';
  int choice;
  printf("Enter 1 for insertion at beginning\n");
  printf("Enter 2 for increment each value\n");
  printf("Enter 3 for display\n");
  while(ch=='y'){
    printf("Enter the choice: ");
    scanf("%d",&choice);
    switch(choice){
      case 1:
        InsertAtHead(&head);
        break;
      case 2:
        increment(head);
        break;
      case 3:
        display(head);
        break;
      default:
        printf("Wrong choice entered!!\n");
    }
    printf("\nWant to continue(y/n): ");
    scanf(" %c",&ch);
  }
  printf("Program executed successfully!!!");
  return 0;
}
```

Output:

```
Welcome
Enter 1 for insertion at beginning
Enter 2 for increment each value
Enter 3 for display
Enter the choice: 1
Enter the data: 4

Want to continue(y/n): y
Enter the choice: 1
Enter the data: 7

Want to continue(y/n): y
Enter the choice: 1
Enter the data: 5

Want to continue(y/n): y
Enter the choice: 3
The elements are: 5     7       4

Want to continue(y/n): y
Enter the choice: 2

Want to continue(y/n): y
Enter the choice: 3
The elements are: 15    17      14

Want to continue(y/n): n
Program executed successfully!!!
```

LAB- 22

Date: 27-Sep-2023

Objective: -Write a program to implement Doubly Linked List, showing all the operations, like creation, display, insertion, deletion and searching.

Theory:-

A doubly linked list is a data structure that consists of a collection of nodes, where each node contains a data element and two pointers, one that points to the next node and another that points to the previous node. This allows for traversal of the list in both directions.

Theory to implement a Doubly linked list

The data field will store the data element of the node. The prev and next fields will store pointers to the previous and next nodes in the list, respectively.

Once we have defined the struct for the node, we can create a doubly linked list by allocating memory for the head node and setting its prev and next pointers to NULL.

Creation: To insert a new node after a given node, we can allocate memory for the new node and set its prev pointer to the given node and its next pointer to the next node after the given node. We then update the next pointer of the given node to be the new node and the prev pointer of the next node after the given node to be the new node

Display: To display the contents of a doubly linked list, we can start from the header node and traverse the list until we reach the last node (which points to NULL). For each node, we print the data element.

Deletion To delete a node from the list, we can check if the node is the head node or the tail node. If the node is the head node, we simply update the head node to be the next node in the list. If the node is the tail node, we update the prev pointer of the previous node in the list to be NULL. If the node is neither the head node nor the tail node, we update the next pointer of the previous node in the list to be the next pointer of the node to be deleted and the prev pointer of the next node in the list to be the prev pointer of the node to be deleted.

Searching: To search for an element in a doubly linked list, we can use the following steps:

1. Start from the header node and traverse the list until we find the element or reach the end of the list.
2. If we find the element, return its location. Otherwise, return -1.

Program:

```
#include <stdio.h>
#include<stdlib.h>
struct node{
  int data;
  struct node*next,*prev;
};
void InsertAtHead(struct node**head){
  int d;
  printf("Enter the data: ");
  scanf("%d",&d);
  struct node* temp;
  temp=(struct node*)(malloc(sizeof(struct node)));
  temp->data=d;
  temp->prev=NULL;
  if(*head==NULL){
    temp->next=NULL;
    *head=temp;
  }
  else{
    temp->next=(*head);
    *head=temp;
  }
  return;
}
void InsertAtTail(struct node**head){
  if(*head==NULL){
    InsertAtHead(head);
    return;
  }
  struct node* newnode;
  struct node* temp;
  newnode=(struct node*)(malloc(sizeof(struct node)));
  int element;
  printf("Enter the element: ");
  scanf("%d",&element);
  newnode->data=element;
  newnode->next=NULL;
  temp=*head;
  while(temp->next!=NULL){
    temp=temp->next;
  }
  temp->next=newnode;
  newnode->prev=temp;
  return;
}
void deletion(struct node**head){
  if(*head==NULL){
    printf("List is empty\n");
    return;
  }
  int element;
  printf("Enter the element to delete: ");
  scanf("%d",&element);
  struct node*prev=NULL,*curr=*head;
  if(curr->data==element){//Handling first element
    *head=(*head)->next;
    free(curr);
    return;
  }
  while(curr!=NULL&&curr->data!=element){
    prev=curr;
    curr=curr->next;
  }
  if(curr==NULL){
    printf("Element not found!!!\n");
    return;
  }
  prev->next=curr->next;
  curr->next->prev=prev;
  free(curr);
  return;
}
void display(struct node *head){
  if(head==NULL){
    printf("Linked list has no elements!!!\n");
    return;
  }
  printf("The elements are: ");
  while(head!=NULL){
    printf("%d\t",head->data);
    head=head->next;
  }
  printf("\n");
}
void search(struct node* head){
  int element,pos=1,check=0;
  printf("Enter the element to search: ");
  scanf("%d",&element);
  while(head!=NULL){
    if(head->data==element){
      printf("Element found at position %d\n",pos);
      check=1;
      return;
    }
    head=head->next;
    pos++;
  }
  if(check==0){
    printf("ELement not found!!\n");
    return;
  }
}
int main(void) {
  printf("Welcome\n");
  struct node *head=NULL;
  char ch='y';
  int choice;
  printf("Enter 1 for insertion at beginning\n");
  printf("Enter 2 for insertion at tail\n");
  printf("Enter 3 for deletion\n");
  printf("Enter 4 for display\n");
  printf("Enter 5 for searching\n");
  while(ch=='y'){
    printf("Enter the choice: ");
    scanf("%d",&choice);
    switch(choice){
      case 1:
        InsertAtHead(&head);
        break;
      case 2:
        InsertAtTail(&head);
        break;
      case 3:
        deletion(&head);
        break;
      case 4:
        display(head);
        break;
      case 5:
        search(head);
        break;
      default:
        printf("Enter choice entered!!\n");
    }
    printf("\nWant to continue(y/n): ");
    scanf(" %c",&ch);
  }
  printf("Program executed successfully!!!");
  return 0;
}
```

Output:

```
Welcome
Enter 1 for insertion at beginning
Enter 2 for insertion at tail
Enter 3 for deletion
Enter 4 for display
Enter 5 for searching
Enter the choice: 1
Enter the data: 4

Want to continue(y/n): y
Enter the choice: 1
Enter the data: 2

Want to continue(y/n): y
Enter the choice: 2
Enter the element: 7

Want to continue(y/n): y
Enter the choice: 4
The elements are: 2     4     7

Want to continue(y/n): y
Enter the choice: 5
Enter the element to search: 4
Element found at position 2

Want to continue(y/n): y
Enter the choice: 3
Enter the element to delete: 4
```

```
Want to continue(y/n): y
Enter the choice: 4
The elements are: 2     7

Want to continue(y/n): n
Program executed successfully!!!
```

CRITERIA	**TOTAL MARKS**	**MARKS OBTAINED**	**COMMENTS**
CONCEPT (A)	2		
IMPLEMENTATION (B)	2		
PERFORMANCE (C)	2		
TOTAL	6 (TO BE SCALED DOWN TO 1.5)		

LAB- 23

Date: 14-Oct-2023

Objective: -Write program to create a Binary Search Tree and delete a node and display its contents using recursive preorder, postorder and inorder traversal.

Theory:-

A binary search tree (BST) is a data structure that efficiently stores a sorted set of elements. It is represented as a tree, where each node has a value and two pointers, one to its left child and one to its right child. The BST property states that all elements in the left subtree of a node are smaller than the node's value, and all elements in the right subtree of a node are greater than the node's value.

Insertion

To insert a new node into a binary search tree, we first compare the new node's value to the value of the root node. If the new node's value is less than the root node's value, we recursively insert the new node into the left subtree. If the new node's value is greater than the root node's value, we recursively insert the new node into the right subtree

Deletion

To delete a node from a binary search tree, we first need to find the node we want to delete. Once we have found the node, we need to consider three cases:

1. If the node has no children, we can simply delete it and set the pointer to the node to null.
2. If the node has one or two children, we need to replace the node with another node.
3. If the node has one child, we can simply replace the node with its child. If the node has two children, we can replace the node with its smallest right child or its largest left child.

Traversal

1. Preorder traversal of a binary search tree visits the root node first, then the left subtree, and then the right subtree. To recursively traverse a binary search tree in preorder
2. Postorder traversal of a binary search tree visits the left subtree first, then the right subtree, and then the root node
3. Inorder traversal of a binary search tree visits the left subtree first, then the root node, and then the right subtree.

Program:

```c
#include <stdio.h>
#include<stdlib.h>
struct node{
  int data;
  struct node* left,*right;
};
struct node* createnode(int val){
  struct node* root=(struct node*)malloc(sizeof(struct node));
  root->data=val;
  root->left=NULL;
  root->right=NULL;
  return root;
}
void inorder(struct node *root){
  if(root!=NULL){
    inorder(root->left);
    printf("%d\t",root->data);
    inorder(root->right);
  }
}
void preorder(struct node *root){
  if(root!=NULL){
    printf("%d\t",root->data);
    preorder(root->left);
    preorder(root->right);
  }
}
void postorder(struct node *root){
  if(root!=NULL){
    postorder(root->left);
    postorder(root->right);
    printf("%d\t",root->data);
  }
}
void insert(struct node**root,int ele){
  if((*root)==NULL){
    (*root)=createnode(ele);
    return;
  }
  if((*root)->data<ele) insert(&((*root)->right),ele);;
  if((*root)->data>ele) insert(&((*root)->left),ele);
}
struct node* nextinorder(struct node**root){
  struct node*curr=*root;
  while(curr&&curr->left!=NULL)
    curr=curr->left;
  return curr;
}
void delete(struct node**root,int ele){
  if((*root)==NULL) printf("\nElement not found!!\n");
  else if((*root)->data<ele) delete(&((*root)->right),ele);
  else if((*root)->data>ele) delete(&((*root)->left),ele);
  else if((*root)->left&&(*root)->right){
    struct node* next=nextinorder(&((*root)->right));
    (*root)->data=next->data;
    delete(&((*root)->right),next->data);
  }
  else{
    struct node**temp=root;
    if((*root)->left==NULL&&(*root)->right==NULL) (*root)=NULL;
    else if((*root)->left!=NULL) (*root)=(*root)->left;
    else if((*root)->right!=NULL) (*root)=(*root)->right;
    free(*temp);}
}
int main(void) {
  printf("Welcome\n");
  char ch='y';
  int ele;
  struct node *root=NULL;
  while(ch=='y'){
    printf("Enter 1 for insertion\n");
    printf("Enter 2 for deletion\n");
    printf("Enter 3 for inorder traversal\n");
    printf("Enter 4 for preorder traversal\n");
```

```c
      printf("Enter 5 for postorderorder traversal\n");
      int choice;
      printf("Enter the choice: ");
      scanf("%d",&choice);
      switch(choice){
        case 1:
          printf("Enter element to insert: ");
          scanf("%d",&ele);
          insert(&root,ele);
          break;
        case 2:
          printf("Enter element to delete: ");
          scanf("%d",&ele);
          delete(&root,ele);
          break;
        case 3:
          inorder(root);
          break;
        case 4:
          preorder(root);
          break;
        case 5:
          postorder(root);
          break;
        default:
          printf("Invalid choice entered!!!\n");
      }
      printf("\nDo you want to continue(y/n): ");
      scanf(" %c",&ch);}
    printf("Program Executed successfully!!");
    return 0;}
```

Output:

```
Welcome
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter 5 for postorderorder traversal
Enter the choice: 1
Enter element to insert: 4

Do you want to continue(y/n): y
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter 5 for postorderorder traversal
Enter the choice: 1
Enter element to insert: 2

Do you want to continue(y/n): y
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter 5 for postorderorder traversal
Enter the choice: 1
Enter element to insert: 8

Do you want to continue(y/n): y
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter 5 for postorderorder traversal
Enter the choice: 3
2       4       8
Do you want to continue(y/n): y
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter 5 for postorderorder traversal
Enter the choice: 4
4       2       8
Do you want to continue(y/n): y
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter 5 for postorderorder traversal
Enter the choice: 5
2       8       4
Do you want to continue(y/n): y
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter 5 for postorderorder traversal
Enter the choice: 2
Enter element to delete: 4

Do you want to continue(y/n): y
Enter 1 for insertion
Enter 2 for deletion
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter 5 for postorderorder traversal
Enter the choice: 3
2       8
Do you want to continue(y/n): n
Program Executed successfully!!
```

CRITERIA	**TOTAL MARKS**	**MARKS OBTAINED**	**COMMENTS**
CONCEPT (A)	2		
IMPLEMENTATION (B)	2		
PERFORMANCE (C)	2		
TOTAL	6 (TO BE SCALED DOWN TO 1.5)		