**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

**FACULTY OF TECHNOLOGY AND ENGINEERING**

Department of Computer Engineering

**Subject Name**: Data Structures & Algorithm

**Semester**: IV

**Subject Code**: CE245

**Academic year**: 2019-20

**PRACTICAL LIST**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **AIM OF THE PRACTICAL** | **Date** | **Page No.** | **Remark** |
| **1.** | Write a program to store roll numbers of student in array who attended training program in random order. Write function for   1. Searching whether particular student attended training program or not using linear search and sentinel search. 2. Searching whether particular student attended training program or not using binary search and Fibonacci search. |  | 4-9 |  |
| **2.** | Mark purchased Books from books store of standard 1 to 7. He purchased 4 books for each standard(for std.1 books are 1.1,1.2,1.3,1.4 and for std. 2 books are 2.1,2.2,2.3,2.4 and so on..). When he reached home, he opens the bag and sees that all the books got mixed. So, how he will sort all the books, according to the standards and their preference in that particular standard. (ex. : preference in std. 1 is 1.1<1.2<1.3  2.1: **SELECTION SORT** that arranges in descending order  2.2: **INSERTION SORT** that arranges in ascending order |  | 10-12 |  |
| **3.** | Implement a menu driven program that performs following sorting algorithms  3.1 **QUICK SORT** that arranges in ascending order  3.2 **MERGE SORT** that arranges in descending order |  | 13-18 |  |
| **4.** | Perform following programs using Stack data structure:  4.1 Sometimes a program requires two stacks containing the same type of items. If the two stacks are stored in separate arrays. Then one stack might overflow while there was considerable unused space in the other. A neat way to avoid the problem is to put all the space in one array and let one stack grow from one end of the array and the other stack start at the other end and grow in opposite direction i.e., toward the first stack, in this way, if one stack turns out to be large and the other small, then they will still both fit, and there will be no overflow until all the space is actually used. Declare a new structure type Double stack that includes the array and the two indices top A and top B, and write functions Push A, Push B, Pop A and Pop B to handle the two stacks with in one Double Stack.  4.2 Implement Tower of Hanoi Problem using Recursion. |  | 19-24 |  |
| **5.** | We are developing software for a call centre. When a client calls, his/her call should be stored until there is a free service representative to pick the call. Calls should be processed in the same order they are received. Select appropriate data structure to build call centre software system. |  | 25-27 |  |
| **6.** | Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of Second, third and final year of department can be granted membership on request. Similarly, one may cancel the membership of club. First node is reserved for  president of club and last node is reserved for secretary of club. Write a program to maintain club member ‘s information using singly linked list. Store student PRN and Name. Write functions to   1. Add and delete the members as well as president or even secretary. 2. Compute total number of members of club 3. Display members 4. Display list in reverse order using recursion 5. Two linked lists exist for two divisions. Concatenate two lists. |  | 28-37 |  |
| **7.** | Write a program to implement Circular Queue with all operations. Check the queue contents and conditions with different combinations of insert and delete operations. Show the content of circular queue with front and rear pointer after each operation. Initially, the queue is empty. The size of the queue is 5. The sequence of operation given below:   * Insert 10, 50, 40, 80 * Delete * Insert 200, 70, 150 * Delete * Delete * Delete |  | 38-41 |  |
| **8.** | Implement the program Display Linked List in Reverse |  | 42-43 |  |
| **9.** | Implement a city database using a BST to store the database records. Each database record contains the name of the city (a string of arbitrary length) and the coordinates of the city expressed as integer x- and y-coordinates The BST should be organized by city name. Your database should allow records to be inserted, deleted by name or coordinate, and searched by name or coordinate. Another operation that should be supported is to print all records within a given distance of a specified point. Collect runningtime statistics for each operation. Which operations can be implemented reasonably efficiently (i.e., inΘ(logn)time in the average case) using a BST? Can the database system be made more efficient by using one or more additional BSTs to organize the records by location? |  | 44-50 |  |
| **10.** | Write a program that enters vertices, edges of a **Graph** and display sequence of vertices to traverse the graph in Depth First Search method. |  | 51-52 |  |
| **11.** | In an array of 20 elements, arrange 15 different values, which are generated randomly. Use **hash function** to generate key and linear probing to avoid collision. H(*x*) = (*x* mod 18) + 2. Write a program to input and display the final values of array. |  | 53-59 |  |

**PRACTICAL 1**

**AIM:**

Write a program to store roll numbers of student in array who attended training program in random order. Write function for

a) Searching whether particular student attended training program or not using linear search and sentinel search.

b) Searching whether particular student attended training program or not using binary search.

**PROGRAM:**

**Linear Search:**

#include<stdio.h>

#include<conio.h>

void main()

{

int arr[10];

int i;

printf("Enter an array: ");

for(i=0;i<5;i++)

{

scanf("%d",&arr[i]);

}

printf("Array is: ");

for(i=0;i<5;i++)

{

printf("%d ",arr[i]);

}

int j;

printf("\n Enter Number to find: ");

scanf("%d",&j);

int count=0;

for( i=0;i<5;i++)

{

if(j==arr[i])

{

count++;

}

}

if(count==0)

{

printf("\nNumber not found");

}

else

{

printf("\nitem found");

}

}

**Sentinel Search:**

#include<stdio.h>

#include<conio.h>

void main()

{

int arr[5];

int i;

printf("Enter an array: ");

for(i=0;i<5;i++)

{

scanf("%d",&arr[i]);

}

for(i=0;i<5;i++)

{

printf("%d ",arr[i]);

}

int last=arr[4];

int j;

printf("\nEnter Number to find: ");

scanf("%d",&j);

arr[4]=j;

int count=0;

while(arr[i]=!arr[4])

{

i++;

}

if(i<4 || last==arr[4])

{

printf("Number found");

}

else

{

printf("Number not found");

}

}

**B)**

**Binary Search:**

#include<stdio.h>

#include<conio.h>

void main()

{

int arr[10];

int i;

printf("Enter an array");

for(i=0;i<10;i++)

{

scanf("%d",&arr[i]);

}

for(i=0;i<10;i++)

{

printf("%d",arr[i]);

}

int mid;

int lb=0;

int ub=9;

int num;

printf("\n Enter no. to find");

scanf("%d",&num);

while(lb<=ub)

{

mid=(lb+ub)/2;

if(arr[mid]==num)

{

printf("\n item matched");

break;

}

else if(num<arr[mid])

{

ub=mid-1;

}

else

{

lb=mid+1;

}

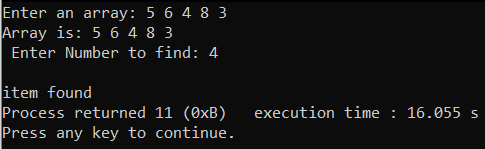
printf("\n item not matched");

}

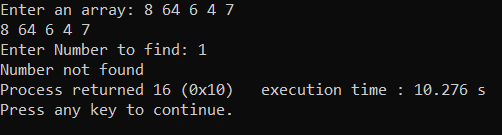
}

**OUTPUT:**

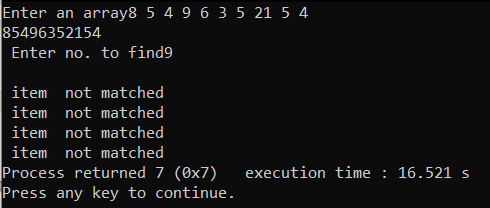
**Linear Search:**



**Sentinel Search:**



**Binary Search:**



**CONCLUSION:**

In this practical we learned about types of search

(1)linear search or sentinel search

(2)binary search.

**PRACTICAL 2**

**AIM:**

Mark purchased Books from books store of standard 1 to 7. He purchased 4 books for each standard(for std.1 books are 1.1,1.2,1.3,1.4 and for std. 2 books are 2.1,2.2,2.3,2.4 and so on..). When he reached home, he opens the bag and sees that all the books got mixed. So, how he will sort all the books, according to the standards and theirpreference in that particular standard. (ex. : preference in std. 1 is 1.1<1.2<1.3

2.1: SELECTION SORT that arranges in descending order

2.2: INSERTION SORT that arranges in ascending order

**PROGRAM:**

**Selection Sort:**

#include<stdio.h>

#include<conio.h>

void main()

{

double arr[]={1.1,1.2,1.3,1.4,2.1,2.2,2.3,2.4,3.1,3.2,3.3,3.4,4.1,4.2,4.3,4.4,5.1,5.2,5.3,5.4,6.1,6.2,6.3,6.4,7.1,7.2,7.3,7.4};

int i,j,k,min;

double temp;

for (i=0;i<27;i++)

{

min=i;

for(j=i+1;j<28;j++)

{

if(arr[min]>arr[j])

{

min=j;

}

}

temp=arr[i];

arr[i]=arr[min];

arr[min]=temp;

}

for(k=0;k<28;k++)

{

printf("%f ",arr[k]);

printf(" ");

}

}

**Insertion Sort:**

#include<stdio.h>

#include<conio.h>

void main()

{

double arr[]={1.1,1.2,1.3,1.4,2.1,2.2,2.3,2.4,3.1,3.2,3.3,3.4,4.1,4.2,4.3,4.4,5.1,5.2,5.3,5.4,6.1,6.2,6.3,6.4,7.1,7.2,7.3,7.4};

int i,j,k,min;

double temp;

for (i=1;i<28;i++)

{

temp=arr[i];

j=i-1;

while(j>=0 && arr[j]>temp)

{

arr[j+1]=arr[j];

j--;

}

arr[j+1]=temp;

}

for(i=0;i<28;i++)

{

printf("%f ",arr[i]);

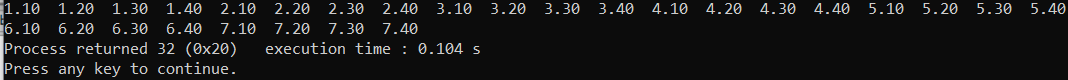
printf(" ");

}

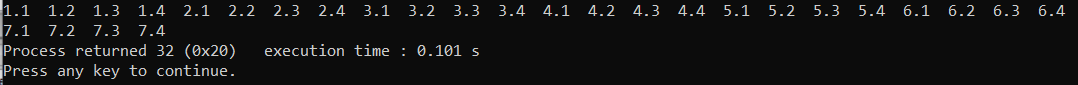
}

**OUTPUT:**

**Selection Sort:**



**Insertion Sort:**



**CONCLUSION:**

In this practical we learned about insertion sort and selection sort in ascending and descending.

**PRACTICAL 3**

**AIM:**

Implement a menu driven program that performs following sorting algorithms

3.1 QUICK SORT that arranges in ascending order

3.2 MERGE SORT that arranges in descending order

**PROGRAM:**

**Quick Sort:**

#include<stdio.h>

// A utility function to swap two elements

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition (int arr[], int low, int high)

{

int pivot = arr[high]; // pivot

int i = (low - 1); // Index of smaller element

for (int j = low; j <= high- 1; j++)

{

if (arr[j] < pivot)

{

i++; // increment index of smaller element

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index \*/

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

//printf("n"); }

// Driver program to test above functions

int main()

{

int arr[] = {10, 7, 8, 9, 1, 5};

int n = sizeof(arr)/sizeof(arr[0]);

quickSort(arr, 0, n-1);

printf("Sorted array: ");

printArray(arr, n);

return 0;

}

**Merge Sort:**

#include<stdlib.h>

#include<stdio.h>

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

void printArray(int A[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", A[i]);

printf("\n");

}

int main()

{

int arr[] = {12, 11, 13, 5, 6, 7};

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size)

mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

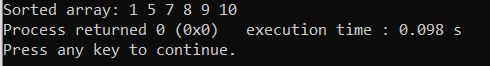
printArray(arr, arr\_size);

return 0;

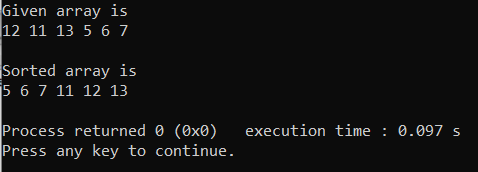
}

**OUTPUT:**

**Quick Sort:**



**Merge Sort:**



**CONCLUSION:**

In this practical we learned about marge sort and quick sort .

**PRACTICAL 4**

**AIM:**

Perform following programs using Stack data structure:

4.1 Sometimes a program requires two stack containing the same type of items. If the two stacks are stored in separate arrays. Then one stack might overflow while there was considerable unused space in the other. A neat way to avoid the problem is to put all the space in one array and let one stack grow from one end of the array and the other stack start at the other end and grow in opposite direction i.e., toward the first stack, in this way, if one stack turns out to be large and the other small, then they will still both fit, and there will be no overflow until all the space is actually used. Declare a new structure type Double stack that includes the array and the two indices top A and top B, and write functions Push A, Push B, Pop A and Pop B to handle the two stacks with in one Double Stack.

4.2 Implement Tower of Hanoi Problem using Recursion.

**PROGRAM:  
4.1**

#include<iostream>

#include<stdlib.h>

using namespace std;

class twoStacks

{

int \*arr;

int size;

int top1, top2;

public:

twoStacks(int n) // constructor

{

size = n;

arr = new int[n];

top1 = -1;

top2 = size;

}

void push1(int x)

{

if (top1 < top2 - 1)

{

top1++;

arr[top1] = x;

}

else

{

cout << "Stack Overflow";

exit(1);

}

}

void push2(int x)

{

if (top1 < top2 - 1)

{

top2--;

arr[top2] = x;

}

else

{

cout << "Stack Overflow";

exit(1);

}

}

int pop1()

{

if (top1 >= 0 )

{

int x = arr[top1];

top1--;

return x;

}

else

{

cout << "Stack UnderFlow";

exit(1);

}

}

int pop2()

{

if (top2 < size)

{

int x = arr[top2];

top2++;

return x;

}

else

{

cout << "Stack UnderFlow";

exit(1);

}

}

};

int main()

{

twoStacks ts(5);

ts.push1(5);

ts.push2(10);

ts.push2(15);

ts.push1(11);

ts.push2(7);

cout << "Popped element from stack1 is " << ts.pop1();

ts.push2(40);

cout << "\nPopped element from stack2 is " << ts.pop2();

return 0;

}

**4.2:**

#include<stdio.h>

#include<conio.h>

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

if (n == 1)

{

printf("\n Move disk 1 from rod %c to rod %c", from\_rod, to\_rod);

return;

}

towerOfHanoi(n-1, from\_rod, aux\_rod, to\_rod);

printf("\n Move disk %d from rod %c to rod %c", n, from\_rod, to\_rod);

towerOfHanoi(n-1, aux\_rod, to\_rod, from\_rod);

}

int main()

{

int n = 4; // Number of disks

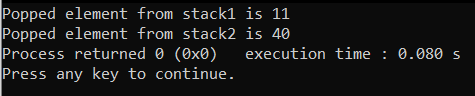
towerOfHanoi(n, 'A', 'C', 'B');

return 0;

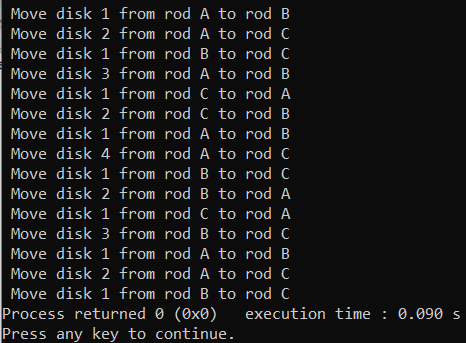
}

**OUTPUT:**

**4.1**



**4.2**



**CONCLUSION:**

In this practical we learned about stack and tower of Hanoi using recursion.

**PRACTICAL 5**

**AIM:**

We are developing software for a call center. When a client calls, his/her call should be

stored until there is a free service representative to pick the call. Calls should be

processed in the same order they are received. Select appropriate data structure to build

call center software system.

**PROGRAM:**

#include<iostream>

using namespace std;

int main()

{

int arr1[10];

static int max1=10,rear=-1,front1=-1,ele,num,i;

//cout<<sizeof(arr1)/sizeof(int);

cout<<"The total size of the array is 10!!";

while(1)

{

cout<<"\nEnter (1 to INSERT, 0 to DELETE , 9 to EXIT ) :";

cin>>num;

cout<<"======================================================";

if(num==1)

{

if(rear>max1)

{

cout<<"\nThe QUEUE IS FULL!";

cout<<"\n=======================================================";

}

else

{

rear=rear+1;

cout<<"\nEnter the Element you want to enter:";

cin>>ele;

arr1[rear]=ele;

front1++;

}

}

else if(num==0)

{

if(front1>=0)

{

ele=arr1[front1];

arr1[front1]=arr1[front1 + 1];

cout<<"\nDELETED item is:"<<ele;

cout<<"\n======================================================";

}

else

{

cout<<"\nTHERE IS NO ELEMENT TO DELETE!!";

cout<<"\n======================================================";

}

}

else if(num==9)

{

break;

}

else

{

for (i=front1+1;i<rear;i++)

{

cout<<"\n";

cout<<" "<<arr1[i];

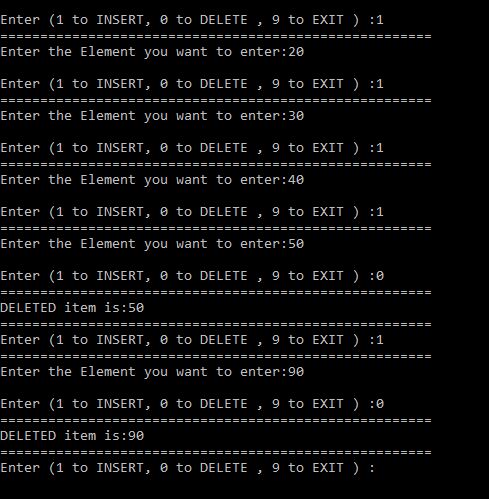
}

}

}

}

**OUTPUT:**

****

**CONCLUSION:**

In this practical we came to know that in queue it works on the principle of ’FIRST IN FIRST OUT’

**PRACTICAL 6**

**AIM:**

Department of Computer Engineering has student's club named 'Pinnacle Club'.

Students of Second, third and final year of department can be granted membership

on request. Similarly one may cancel the membership of club. First node is reserved

for president of club and last node is reserved for secretary of club. Write C++

program to maintain club member‘s information using singly linked list. Store

student PRN and Name. Write functions to

a) Add and delete the members as well as president or even secretary.

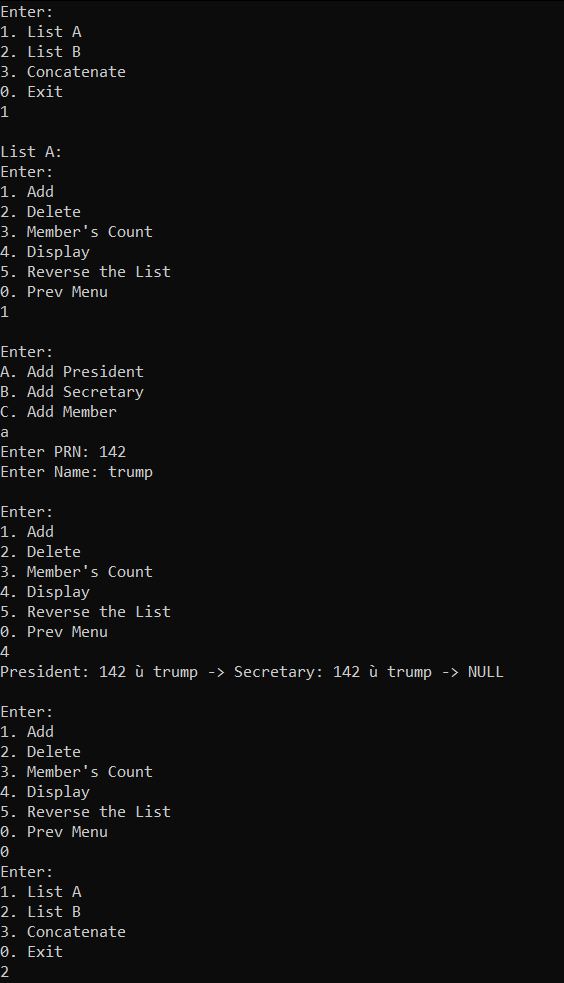
b) Compute total number of members of club

c) Display members

d) Display list in reverse order using recursion

e) Two linked lists exists for two divisions. Concatenate two lists.

|  |
| --- |
| **PROGRAM CODE:**  #include <iostream>  #include <string.h>  using namespace std;  //Node  struct node {  int prn;  string name;  struct node \*next;  };  //Linked List  class list {  node \*head, \*temp;  public:  list() {  head = NULL;  }  node \*create(int val, string n);  void insertEnd();  void insertBeg();  void deleteAt(int i);  void insertAt(int i);  void display();  int count();  void reverse();  void rev(node \*t);  node\* readAt(int i);  void concatenate(list A,list B);  void op();  };  //Create  node\* list::create(int val, string n) {  temp = new(struct node);  if (temp == NULL) {  cout<<"Memory Allocation Failed!"<<endl;  return 0;  } else {  temp -> prn = val;  temp -> name = n;  temp -> next = NULL;  return temp;  }  }  //Insert End  void list::insertEnd() {  int val;  string n;  cout<<"Enter PRN: ";  cin>>val;  cout<<"Enter Name: ";  cin>>n;  struct node \*t = head;  temp = create(val,n);  if (head == NULL) {  head = temp;  head -> next = NULL;  } else {  while ((t -> next) != NULL) {  t = t -> next;  }  temp -> next = NULL;  t -> next = temp;  cout<<"Element Inserted at Last"<<endl;  }  }  //Insert At  void list::insertAt(int i) {  int val,pos = i - 1,counter = 1;  string n;  struct node \*ptr;  struct node \*t = head;  while ((t -> next) != NULL) { //loop to count number of items in linked list.  t = t -> next;  counter++;  }  t = head; //traverse pointer is pointed to head again.  if (i == 1) { //equivalent to insert at start.  insertBeg();  } else if (pos > counter || i <= 0) { //if position is greater than the actual linked list.  cout<<"Entered position is out of scope."<<endl;  } else { //insert at required position.  cout<<"Enter PRN: ";  cin>>val;  cout<<"Enter Name: ";  cin>>n;  temp = create(val,n);  while (pos--) {  ptr = t;  t = t -> next;  }  temp -> next = t;  ptr -> next = temp;  cout<<"Member Inserted at Position: "<<i<<endl;  }  }  //Delete At  void list::deleteAt(int i) {  int val,pos = i - 1,counter = 1;  string n;  struct node \*ptrl,\*ptrr;  struct node \*t = head;  while ((t -> next) != NULL) {  t = t -> next;  counter++;  }  t = head;  if (i == 1) {  ptrl = head;  head = head -> next;  delete ptrl;  } else if (pos > counter || i <= 0) {  cout<<"Entered member doesn't exist."<<endl;  } else {  while (pos--) {  ptrl = t;  t = t -> next;  ptrr = t -> next;  }  ptrl -> next = ptrr;  delete t;  cout<<"Member Deleted at Position: "<<i<<endl;  }  }  //Insert Beg  void list::insertBeg() {  int val;  string n;  cout<<"Enter PRN: ";  cin>>val;  cout<<"Enter Name: ";  cin>>n;  //v = val;  struct node \*t = head;  temp = create(val,n);  if (head == NULL) {  head = temp;  head -> next = NULL;  } else {  temp -> next = head;  head = temp;  cout<<"We have a New President."<<endl;  }  }  //Display  void list::display() {  temp = head;  cout<<"President: ";  cout<< temp -> prn<<" — "<<temp -> name<<" -> ";  if(temp -> next != NULL) {  temp = temp -> next;  }  while (temp -> next != NULL) {  cout<< temp -> prn<<" — "<<temp -> name<<" -> ";  temp = temp -> next;  }  cout<<"Secretary: ";  cout<< temp -> prn<<" — "<<temp -> name<<" -> ";  cout<<"NULL"<<endl;  }  //Count  int list::count() {  temp = head;  int ct = 0;  while (temp != NULL) {  ct++;  temp = temp -> next;  }  return ct;  }  //Concatenate  void list::concatenate(list A,list B) {  struct node \* last,\*last1;  node\* t = A.head;  while (t != NULL) {  int val = t -> prn;  string n = t -> name;  temp = create(val,n);  if (head == NULL) {  head = temp;  head -> next = NULL;  last=head;  } else {  //temp -> next = NULL;  last -> next = t;  last=t;  }  t = t -> next;  }  last -> next = B.head;  t = B.head;  while (t != NULL) {  int val = t -> prn;  string n = t -> name;  temp = create(val,n);  last -> next = temp;  last= temp;  t = t -> next;  }  last->next=NULL;  }  //Accept  void list::op() {  while(1) {  int choice;  cout<<"\nEnter: \n1. Add \n2. Delete \n3. Member's Count \n4. Display \n5. Reverse the List \n0. Prev Menu"<<endl;  cin>>choice;  switch(choice) {  case 1: { //Add  char c;  cout<<"\nEnter: \nA. Add President \nB. Add Secretary \nC. Add Member"<<endl;  cin>>c;  switch(c) {  case 'A':  case 'a':{  insertBeg();  break;  }  case 'B':  case 'b': {  insertEnd();  break;  }  case 'C':  case 'c': {  insertAt(2);  break;  }  }  break;  }  case 2: { //Delete  char c;  cout<<"\nEnter: \nA. Delete President \nB. Delete Secretary \nC. Delete Member"<<endl;  cin>>c;  switch(c) {  case 'A': {  deleteAt(1);  cout<<"Club must have a President. Enter Details"<<endl;  insertBeg();  break;  }  case 'B': {  deleteAt(count());  cout<<"Club must have a Secretary. Enter Details"<<endl;  insertEnd();  break;  }  case 'C': {  int j;  cout<<"Enter Position for Deletion"<<endl;  cin>>j;  deleteAt(j);  break;  }  }  break;  }  case 3: { //Count  cout<<"Count: "<<count()<<endl;  break;  }  case 4: { //Display  if (head == NULL) {  cout<<"NULL"<<endl;  break;  } else {  display();  break;  }  }  case 5: { //Reverse  reverse();  break;  }  case 0: { //Prev Menu  return;  }  }  }  }  //Reverse Recursion  void list::rev(node \*t) {  if(t -> next != NULL) {  rev (t -> next);  }  if(t == head)  cout<<"Secretary: "<<t -> prn<<" — "<<t -> name<<endl;  else if(t -> next == NULL)  cout<<"President: "<<t -> prn<<" — "<<t -> name<<" -> ";  else  cout<<"Member: "<<t -> prn<<" — "<<t -> name<<" -> ";  }  //Reverse  void list::reverse() {  rev(head);  }  //Read At  node\* list::readAt(int i) {  struct node \*t = head;  int c = count();  while(c--) {  t = t-> next;  }  }  //Main  int main() {  list L,X,Y;  int c;  while(1) {  cout<<"Enter: \n1. List A \n2. List B \n3. Concatenate\n0. Exit"<<endl;  cin>>c;  switch(c) {  case 1: cout<<"\nList A:"; X.op(); break;  case 2: cout<<"\nList B:"; Y.op(); break;  case 3: L.concatenate(X,Y); L.display(); break;  case 0: return 0;  }  }  }  **OUTPUT:** |



**CONCLUSION:**

From the above practical , we have learnt how to do insert , delete and display operation in it and to display list in reverse order using recursion. Also learnt how concatenate two linked lists.

**PRACTICAL 7**

**AIM:**

Write a program to implement Circular Queue with all operations. Check the queue contents and conditions with different combinations of insert and delete operations. Show the content of circular queue with front and rear pointer after each operation. Initially, the queue is empty. The size of the queue is 5. The sequence of operation given below:

* Insert 10, 50, 40, 80
* Delete
* Insert 200, 70, 150
* Delete
* Delete
* Delete

**PROGRAM CODE:**

import java.io.\*;

public class Circular\_Queue{

int Q[] = new int[100];

int n, front, rear;

static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

public Circular\_Queue(int nn){

n=nn;

front = rear = 0;

}

public void add(int v){

if((rear+1) % n != front){

rear = (rear+1)%n;

Q[rear] = v;

}

else

System.out.println("Queue is full !");

}

public int del(){

int v;

if(front!=rear){

front = (front+1)%n;

v = Q[front];

return v;

}

else

return -9999;

}

public void disp(){

int i;

if(front != rear){

i = (front +1) %n;

while(i!=rear){

System.out.println(Q[i]);

i = (i+1) % n;

}

}

else

System.out.println("Queue is empty !");

}

public static void main(String args[]) throws IOException{

System.out.print("Enter the size of the queue : ");

int size = Integer.parseInt(br.readLine());

Circular\_Queue call = new Circular\_Queue(size);

int choice;

boolean exit = false;

while(!exit){

System.out.print("\n1 : Add\n2 : Delete\n3 : Display\n4 : Exit\n\nYour Choice : ");

choice = Integer.parseInt(br.readLine());

switch(choice){

case 1 :

System.out.print("\nEnter number to be added : ");

int num = Integer.parseInt(br.readLine());

call.add(num);

break;

case 2 :

int popped = call.del();

if(popped != -9999)

System.out.println("\nDeleted : " +popped);

else

System.out.println("\nQueue is empty !");

break;

case 3 :

call.disp();

break;

case 4 :

exit = true;

break;

default :

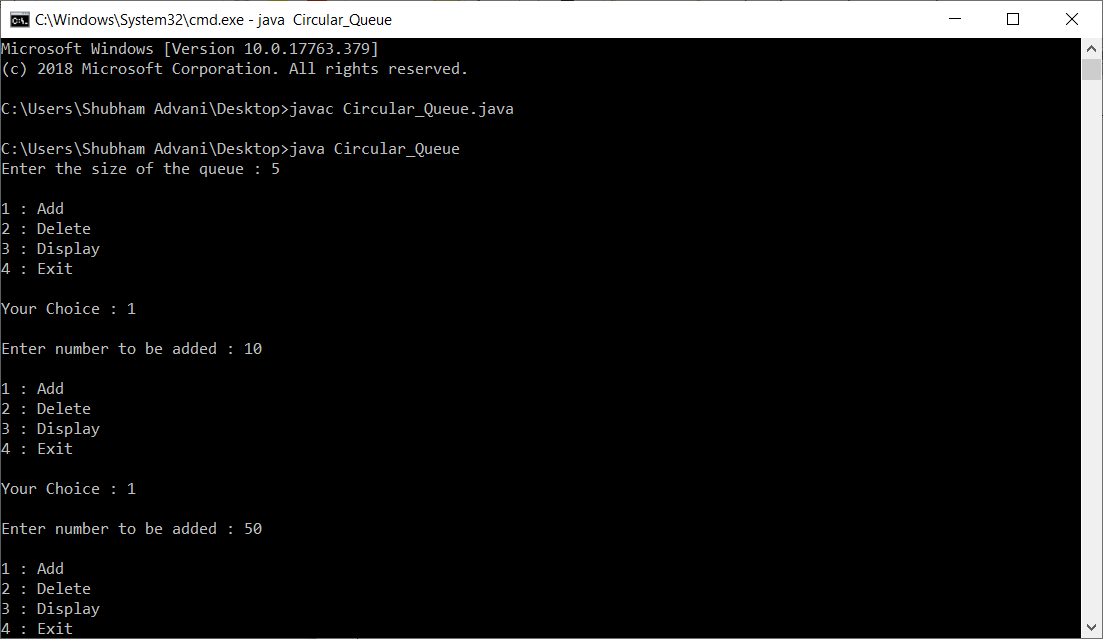
System.out.println("\nWrong Choice !");

break;

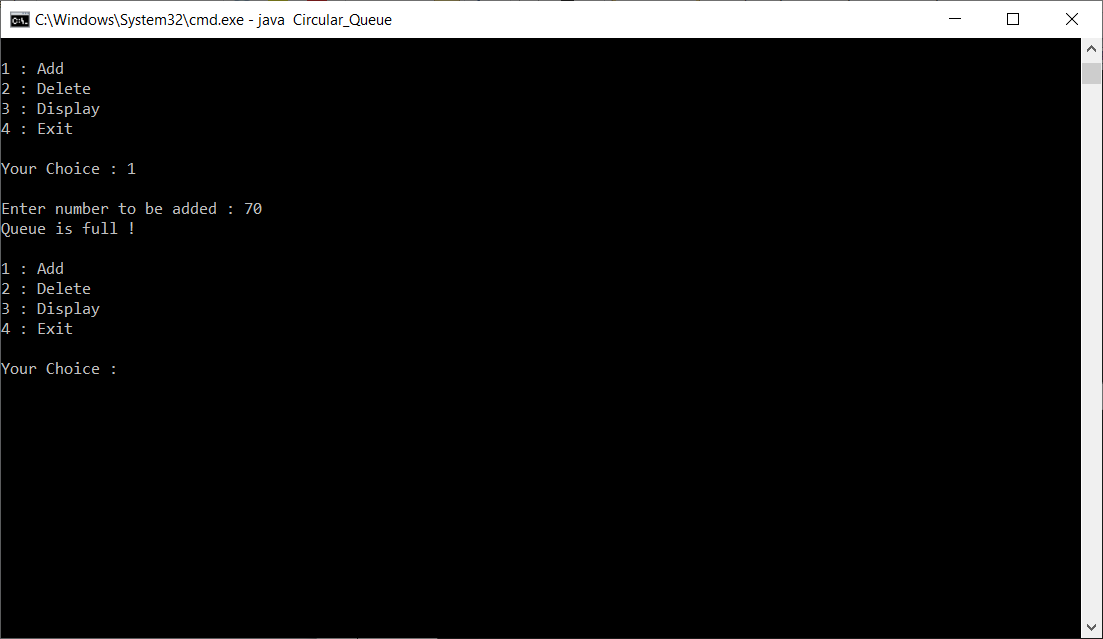
}

}}}

**OUTPUT:**







**CONCLUSION:**

From the above practical , we learned about circular queue and how to do insert , delete and display operation in it.

**PRACTICAL 8**

**AIM:**

Implement the program display Linked List in Reverse.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

static void reverse(struct Node\*\* head\_ref)

{

struct Node\* prev = NULL;

struct Node\* current = \*head\_ref;

struct Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*head\_ref = prev;

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void printList(struct Node\* head)

{

struct Node\* temp = head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

}

int main()

{

struct Node\* head = NULL;

push(&head, 20);

push(&head, 4);

push(&head, 15);

push(&head, 85);

printf("Given linked list\n");

printList(head);

reverse(&head);

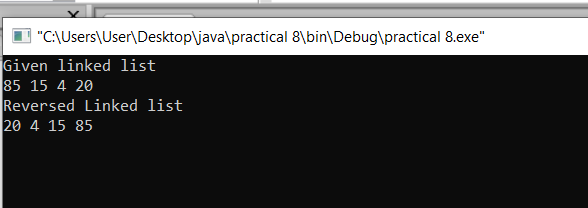
printf("\nReversed Linked list \n");

printList(head);

getchar();

}

**OUTPUT:**



**CONCLUSION:**

From the above practical , we learned about display Linked List in Reverse.

**PRACTICAL 9**

**AIM:**

Implement a city database using a BST to store the database records. Each database record contains the name of the city (a string of arbitrary length) and the coordinates of the city expressed as integer x- and y-coordinates The BST should be organized by city name. Your database should allow records to be inserted, deleted by name or coordinate, and searched by name or coordinate. Another operation that should be supported is to print all records within a given distance of a specified point. Collect running-time statistics for each operation. Which operations can be implemented reasonably efficiently (i.e., inΘ(logn)time in theaverage case) using a BST? Can the database system be made more efficient by using one or more additional BSTs to organize the records by location?

**PROGRAM CODE:**

import java.util.Random;

import java.util.Scanner;

class BSTNode {

BSTNode left, right;

int data;

public BSTNode() {

left = null;

right = null;

data = 0;

}

public BSTNode(int n) {

left = null;

right = null;

data = n;

}

public void setLeft(BSTNode n) {

left = n;

}

public void setRight(BSTNode n) {

right = n;

}

public BSTNode getLeft() {

return left;

}

public BSTNode getRight(){

return right;

}

public void setData(int d) {

data = d;

}

public int getData() {

return data;

}

}

class BST {

private BSTNode root;

public BST() {

root = null;

}

public boolean isEmpty() {

return root == null;

}

public void insert(int data) {

root = insert(root, data);

}

private BSTNode insert(BSTNode node, int data) {

if (node == null)

node = new BSTNode(data);

else {

if (data <= node.getData())

node.left = insert(node.left, data);

else

node.right = insert(node.right, data);

}

return node;

}

public void delete(int k) {

if (isEmpty())

System.out.println("Tree Empty");

else if (search(k) == false)

System.out.println("Sorry " + k + " is not present");

else {

root = delete(root, k);

System.out.println(k + " deleted from the tree");

}

}

private BSTNode delete(BSTNode root, int k) {

BSTNode p, p2, n;

if (root.getData() == k) {

BSTNode lt, rt;

lt = root.getLeft();

rt = root.getRight();

if (lt == null && rt == null)

return null;

else if (lt == null) {

p = rt;

return p;

}

else if (rt == null) {

p = lt;

return p;

}

else {

p2 = rt;

p = rt;

while (p.getLeft() != null)

p = p.getLeft();

p.setLeft(lt);

return p2;

}

}

if (k < root.getData()) {

n = delete(root.getLeft(), k);

root.setLeft(n);

}

else {

n = delete(root.getRight(), k);

root.setRight(n);

}

return root;

}

public int countNodes() {

return countNodes(root);

}

private int countNodes(BSTNode r) {

if (r == null)

return 0;

else {

int l = 1;

l += countNodes(r.getLeft());

l += countNodes(r.getRight());

return l;

}

}

public boolean search(int val) {

return search(root, val);

}

private boolean search(BSTNode r, int val) {

boolean found = false;

while ((r != null) && !found) {

int rval = r.getData();

if (val < rval)

r = r.getLeft();

else if (val > rval)

r = r.getRight();

else {

found = true;

break;

}

found = search(r, val);

}

return found;

}

public void inorder() {

inorder(root);

}

private void inorder(BSTNode r) {

if (r != null) {

inorder(r.getLeft());

System.out.print(r.getData() + " ");

inorder(r.getRight());

}

}

public void preorder() {

preorder(root);

}

private void preorder(BSTNode r) {

if (r != null) {

System.out.print(r.getData() + " ");

preorder(r.getLeft());

preorder(r.getRight());

}

}

public void postorder() {

postorder(root);

}

private void postorder(BSTNode r) {

if (r != null) {

postorder(r.getLeft());

postorder(r.getRight());

System.out.print(r.getData() + " ");

}

}

}

public class Binary\_Search\_Tree {

public static int N = 20;

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

BST bst = new BST();

boolean loop=false;

while(!loop){

System.out.println("Enter your choice");

System.out.println("1.Insert\n2.Delete\n3.Search\n4.Exit");

int choice=sc.nextInt();

switch (choice) {

case 1:

System.out.println("Enter the element to be inserted:");

int in=sc.nextInt();

bst.insert(in);

break;

case 2:

System.out.println("Enter the number to be deleted:");

int out=sc.nextInt();

bst.delete(out);

break;

case 3:

System.out.println("Enter the number to be searched:");

int see=sc.nextInt();

System.out.println(bst.search(see));

break;

case 4:

loop=true;

break;

default:

System.out.println("Incorrect Input! Enter again!");

break;

}

}

System.out.println("In order traversal of the tree :");

bst.inorder();

System.out.println();

System.out.println("Post order traversal of the tree :");

bst.postorder();

System.out.println();

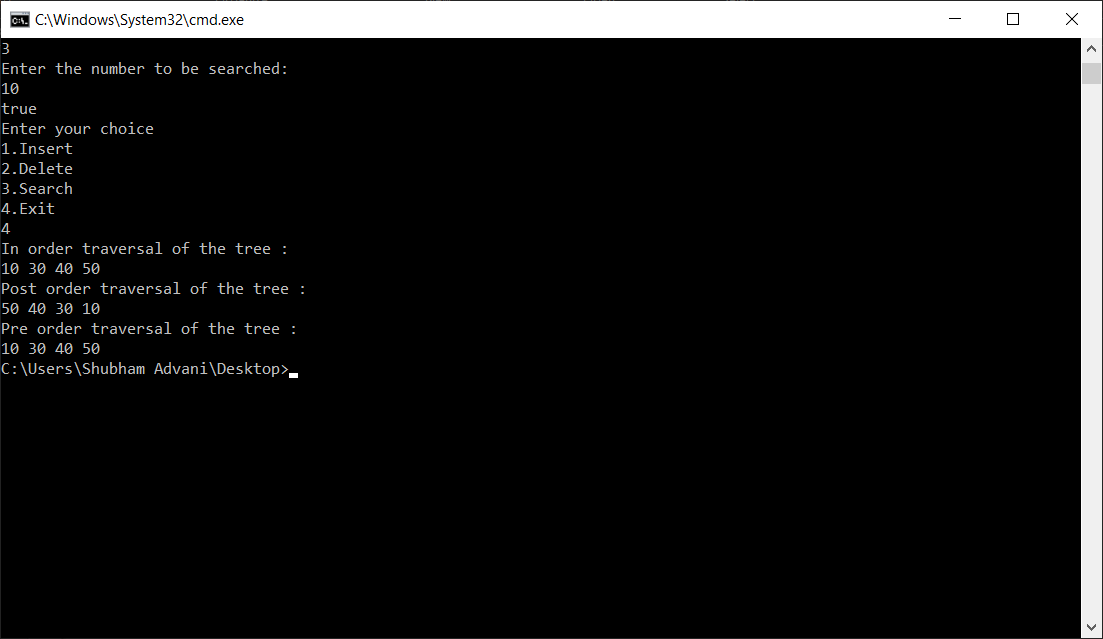
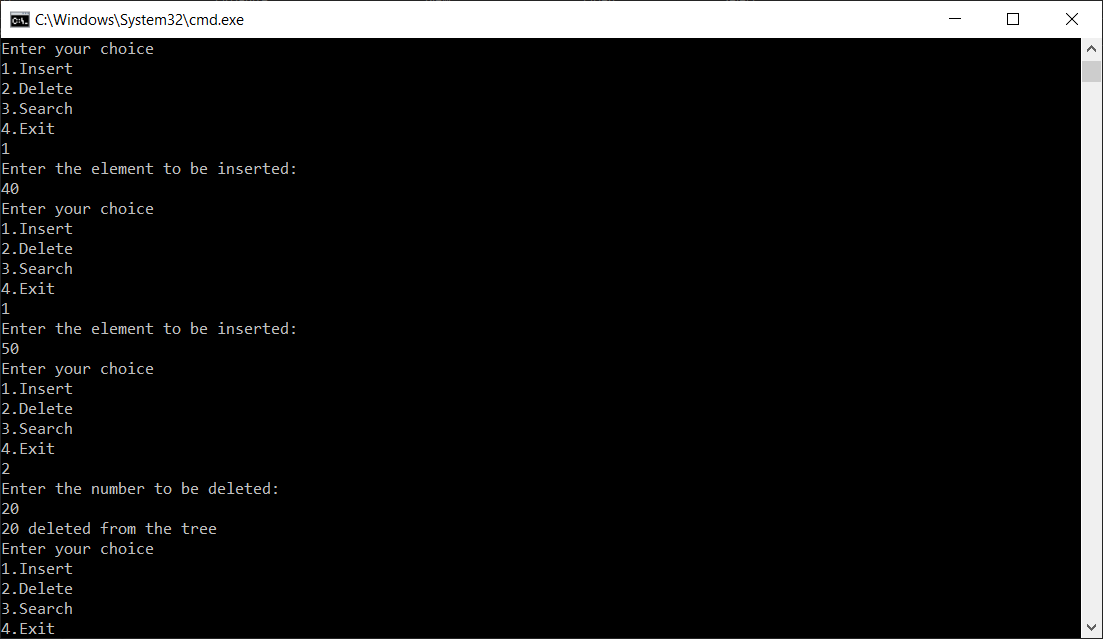
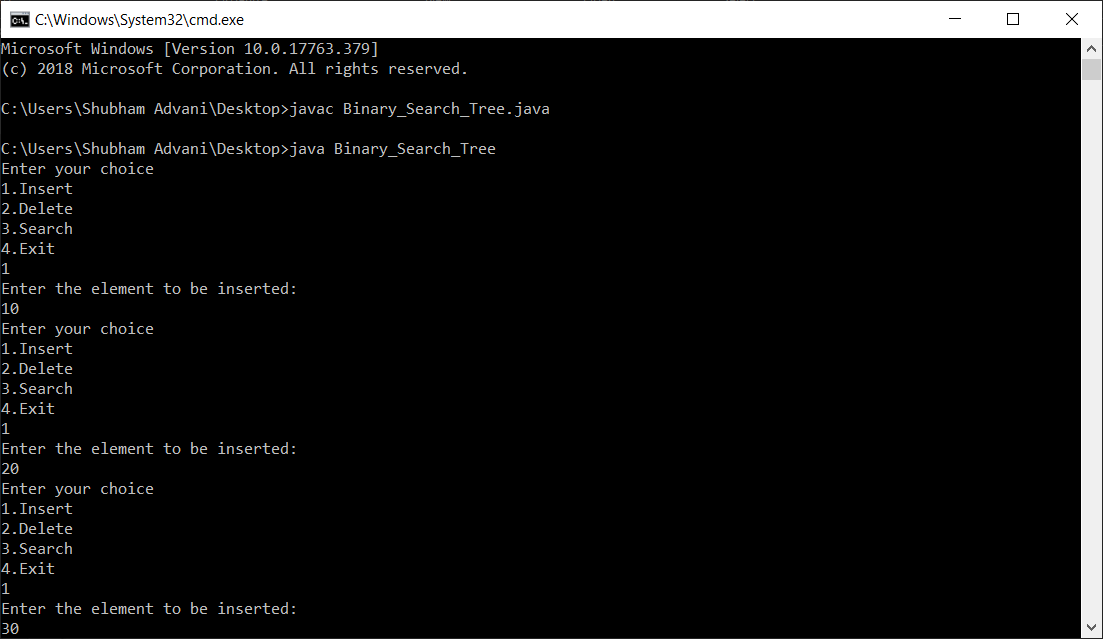
System.out.println("Pre order traversal of the tree :");

bst.preorder();

sc.close();

}}

**OUTPUT:**



**CONCLUSION:**

From the above practical , we learned about BST(Binary Search tree) and then take data from user and store it and display it in inorder, postorder and preorder.

**PRACTICAL 10**

**AIM:**

Write a program that enters vertices, edges of a Graph and display sequence of vertices to traverse the graph in Depth First Search method.

**PROGRAM CODE:**

import java.util.\*;

public class Hash\_Key{

public static void main(String args[]) {

Random random=new Random();

long array[]=new long[20];

long temp;

int i,count=0,hk=0;

System.out.println("Data generated randomly:");

for (i = 0;; i++) {

if(count<20){

temp=(long)(Math.random()%10000000)\*7;

if(temp>=100000&&temp<=999999){

hk=(int)temp%18+2;

System.out.println("Hash Key= "+hk+" -> \t "+temp);

if(array[hk]!=0){

while (array[hk]!=0) {

hk++;

if(hk>20){

hk=0;

}

}

array[hk]=temp;

}

else{

array[hk]=temp;

count++;

}

}

}

else{

break;

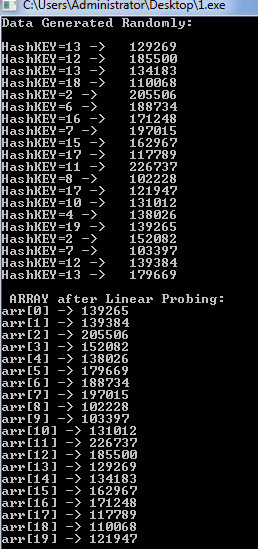
}

}

}

}

**OUTPUT:**



**CONCLUSION:**

In this practical, we learnt how to use hash function to generate key and linear probing to avoid collision.

**PRACTICAL 11**

**AIM:**

In an array of 20 elements, arrange 15 different values, which are generated randomly. Use hash function to generate key and linear probing to avoid collision. H(x) = (x mod 18) + 2. Write a program to input and display the final values of array.

**PROGRAM CODE 1:**

#include<stdio.h>

#include<stdlib.h>

/\* to store a data (consisting of key and value) in hash table array \*/

struct item

{

int key;

int value;

};

/\* each hash table item has a flag (status) and data (consisting of key and value) \*/

struct hashtable\_item

{

int flag;

/\*

\* flag = 0 : data does not exist

\* flag = 1 : data exists

\* flag = 2 : data existed at least once

\*/

struct item \*data;

};

struct hashtable\_item \*array;

int size = 0;

int max = 10;

/\* initializing hash table array \*/

void init\_array()

{

int i;

for (i = 0; i < max; i++)

{

array[i].flag = 0;

array[i].data = NULL;

}

}

/\* to every key, it will generate a corresponding index \*/

int hashcode(int key)

{

return (key % max);

}

/\* to insert an element in the hash table \*/

void insert(int key, int value)

{

int index = hashcode(key);

int i = index;

/\* creating new item to insert in the hash table array \*/

struct item \*new\_item = (struct item\*) malloc(sizeof(struct item));

new\_item->key = key;

new\_item->value = value;

/\* probing through the array until we reach an empty space \*/

while (array[i].flag == 1)

{

if (array[i].data->key == key)

{

/\* case where already existing key matches the given key \*/

printf("\n Key already exists, hence updating its value \n");

array[i].data->value = value;

return;

}

i = (i + 1) % max;

if (i == index)

{

printf("\n Hash table is full, cannot insert any more item \n");

return;

}

}

array[i].flag = 1;

array[i].data = new\_item;

size++;

printf("\n Key (%d) has been inserted \n", key);

}

/\* to remove an element from the hash table \*/

void remove\_element(int key)

{

int index = hashcode(key);

int i = index;

/\* probing through array until we reach an empty space where not even once an element had been present \*/

while (array[i].flag != 0)

{

if (array[i].flag == 1 && array[i].data->key == key )

{

// case when data key matches the given key

array[i].flag = 2;

array[i].data = NULL;

size--;

printf("\n Key (%d) has been removed \n", key);

return;

}

i = (i + 1) % max;

if (i == index)

{

break;

}

}

printf("\n This key does not exist \n");

}

/\* to display all the elements of hash table \*/

void display()

{

int i;

for (i = 0; i < max; i++)

{

struct item \*current = (struct item\*) array[i].data;

if (current == NULL)

{

printf("\n Array[%d] has no elements \n", i);

}

else

{

printf("\n Array[%d] has elements -: \n %d (key) and %d(value) ", i, current->key, current->value);

}

}

}

int size\_of\_hashtable()

{

return size;

}

void main()

{

int choice, key, value, n, c;

array = (struct hashtable\_item\*) malloc(max \* sizeof(struct hashtable\_item\*));

init\_array();

do {

printf("Implementation of Hash Table in C with Linear Probing \n\n");

printf("MENU-: \n1.Inserting item in the Hashtable"

"\n2.Removing item from the Hashtable"

"\n3.Check the size of Hashtable"

"\n4.Display Hashtable"

"\n\n Please enter your choice-:");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Inserting element in Hashtable\n");

printf("Enter key and value-:\t");

scanf("%d %d", &key, &value);

insert(key, value);

break;

case 2:

printf("Deleting in Hashtable \n Enter the key to delete-:");

scanf("%d", &key);

remove\_element(key);

break;

case 3:

n = size\_of\_hashtable();

printf("Size of Hashtable is-:%d\n", n);

break;

case 4:

display();

break;

default:

printf("Wrong Input\n");

}

printf("\n Do you want to continue-:(press 1 for yes)\t");

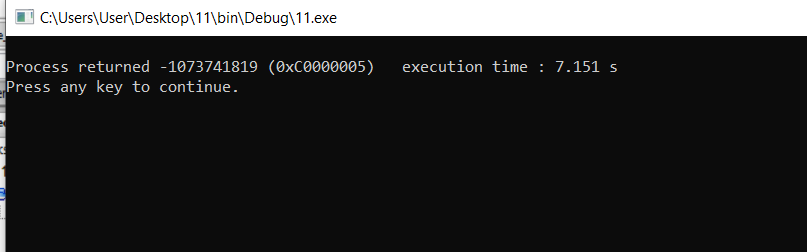
scanf("%d", &c);

}while(c == 1);

getch();

}

**OUTPUT:**



**PROGRAM CODE 2:**

#include <stdio.h>

int main()

{

int a[1000],i,n;

printf("Enter size of array: ");

scanf("%d",&n);

printf("Enter %d elements in the array : ", n);

for(i=0;i<n;i++)

{

scanf("%d", &a[i]);

}

printf("\nElements in array are: ");

for(i=0;i<n;i++)

{

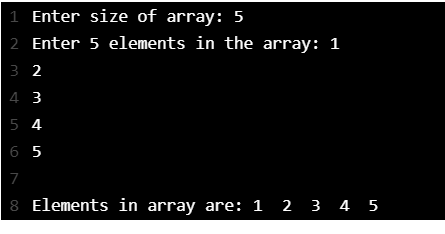
printf("%d ", a[i]);

}

return 0;

}

**OUTPUT:**



**CONCLUSION:**

In this practical, we learnt hash function to generate key and linear probing to avoid collision and to input and display the final values of array.