Dr. Babasaheb Ambedkar Technological University, Lonere

Diploma in Computer Engineering

Semester: IV Course: Data Structures Lab Course Code: DCE2209

Lab Manual

Program No. 1. Program to insert an element in an array

Maps with: CO1, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the operation of insertion in an array

Program:

```
#include<iostream>
using namespace std;
#define MAX 100
int main()
{
int j,n,k,item;
int LA[MAX];
//Creation of an array
cout<<"Enter the size and elements of an array:\n";
cin>>n;
for(j=0;j<n;j++)
cin>>LA[j];
cout<<"**********Array before INSERTION**********\n";
//Displaying the elements of an array
for(j=0;j<n;j++)
cout<<LA[j]<<"\t";
cout<<"\n";
//Insertion of element in an array
cout<<"Enter the element you want to insert and its position";
cin>>item>>k;
for(j=n-1;j>=k-1;j--)
```

```
{
LA[j+1]=LA[j];
}
LA[k-1]=item;
n=n+1;
cout<<"**********Array after INSERTION**********\n";
//Displaying the elements of an array
for(j=0;j<n;j++)
cout<<LA[j]<<"\t";
cout<<"\n";
return 0;
}
```

Output:

```
Enter the size and elements of an array:
5
1
9
5
6
7
**********Array before INSERTION**********
1       9       5       6       7
Enter the element you want to insert and its position
8
1
**********Array after INSERTION**********
8       1       9       5       6       7
```

Program No. 2. Program to delete an element from an array

Maps with: CO1, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the operation of deletion from an array

Program:

```
#include<iostream>
using namespace std;
#define MAX 100
int main()
{
int j,k,n,item;
int LA[MAX];
cout<<"Enter size and elements of the array\n";
cin>>n;
for(j=0;j<n;j++)
cin>>LA[j];
cout<<"*****Array before deletion********\n";
for(j=0;j<n;j++)
cout<<LA[j]<<"\t";
cout<<"\n";
cout<<"Enter the position of element you want to delete\n";
cin>>k;
item=LA[k-1];
for(j=k-1;j<=n-2;j++)
LA[j]=LA[j+1];
n=n-1;
cout<<"*****Array after deletion********\n";
for(j=0;j<n;j++)
cout<<LA[j]<<"\t";
cout<<"\n";
return 0;
```

}

Output:

Enter the size and elements of an array:

5

1

9

5

6

7

**********Array before deletion**********

1 9 5 6 7

Enter the element you want to delete and its position

1

**********Array after deletion**********

9 5 6 7

Program No. 3. Write a Program to Implement bubble sort algorithm.

Maps with: CO1, CO6, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the operation of sorting elements of an array using bubble sort algorithm

Program:

```
#include<iostream>
using namespace std;
#define MAX 100
int main()
{
int data[MAX];
int k,ptr,n,i,temp;
cout<<'Enter the size and elements of the array\n";
cin>>n;
for(i=0;i<n;i++)
cin>>data[i];
cout<<"******Array before sorting**********\n";
for(i=0;i<n;i++)
cout<<data[i]<<"\t";
cout<<"\n";
//Use of bubble sort for sorting
for(k=1;k<n;k++)
{
ptr=0;
while(ptr<n-k)
{
if(data[ptr]>data[ptr+1])
{
temp=data[ptr];
data[ptr]=data[ptr+1];
```

```
data[ptr+1]=temp;
}
ptr++;
}
}
cout<<"******Array after sorting**********\n";
for(i=0;i<n;i++)
cout<<data[i]<<"\t";
cout<<"\n";
return 0;
}
```

Output:

```
Enter the size and elements of the array:
5
9
8
7
6
5
The sorted array after bubble sort is:
5       6       7       8       9
```

Program No. 4. Write a Program to Implement linear search algorithm

Maps with: CO1, CO6, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the operation of searching an element in an array using linear search algorithm

Program:

```
#include<iostream>
using namespace std;
#define MAX 100
int main()
{
int data[MAX];
int beg,end,mid,item,i,n;
cout<<"Enter the size and elements of the array\n";
cin>>n;
for(i=0;i<n;i++)
cin>>data[i];
cout<<"Following is the given array\n";
for(i=0;i<n;i++)
cout<<data[i]<<"\t";
cout<<"\n";
cout<<"Enter the element you want to search:\n";
cin>>item;
data[n]=item;
i=0;
while(data[i]!=item)
i=i+1;
if (i==n)
cout<<"The element that you are searching for is not present in the array\n";
else
cout<<"The element is found at location= "<<i+1<<"\n";
```

```
return 0;
}
```

Output:

```
Enter the size and elements of the array:
5
9
8
7
6
5
Enter value to be search:
8
Element found at index 2
```

Program No. 5: Write a Program to implement binary search algorithm.

Maps with: CO1, CO6, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the operation of searching an element in an array sorted in ascending order using binary search algorithm

Program:

```
#Include<iostream>
using namespace std;
#define MAX 100
int main()
{
int beg,end,mid,i,n,item;
int data[MAX];
cout<<"Enter the size and elements of an array in ascending order\n";
cin>>n;
for(i=0;i<n;i++)
cin>>data[i];
cout<<"****Given Array**********\n";
for(i=0;i<n;i++)
cout<<data[i]<<"\t";
cout<<"\n";
cout<<'Enter the element you are searching for\n";
cin>>item;
beg=0;
end=n-1;
mid=(beg+end)/2;
while(beg<=end and data[mid]!=item)
{
if(item<data[mid])
end=mid-1;
else
```

```
beg=mid+1;
mid=(beg+end)/2;
}
if(data[mid]==item)
cout<<"The item is found at location: "<<mid+1<<"\n";
else
cout<<"Item not found\n";
return 0;
}
```

Output:

Enter the size and elements of the array (in ascending order):

5

2

4

6

8

10

Enter value to be searched:

8

Element found at index 2

Program No. 6: Write a Program to search an element in an unsorted linked list

Maps with: CO2, CO6, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the operation of searching an element in an unsorted linked list

Program:

```
#include<iostream>
#include<stdlib.h>
#include<malloc.h>
using namespace std;
#define MAX 100
struct node{
int info;
struct node *link;
}*start=NULL;
int main()
{
int i=0,n,item,data,count=0,flag=0;
struct node *ptr;
struct node *temp_start;
cout<<"Enter the size of the linked list\n";
cin>>n;
do
{
cin>>data;
struct node *temp=new struct node;
temp->info=data;
temp->link=NULL;
if(start==NULL)
start=temp;
else
{
```

```
ptr=start;
while(ptr->link!=NULL)
ptr=ptr->link;
ptr->link=temp;
}
i=i+1;
}
while(i<n);
cout<<"Elements in the linked list are as follows:\n";
struct node *p;
p=start;
while(p!=NULL)
{
cout<<p->info<<"\t";
p=p->link;
}
cout<<"\n";
cout<<"Enter the element you want to search\n";
cin>>item;
//Searching in an unsorted linked list
struct node *q;
q=start;
while(q!=NULL)
{
count++;
if(q->info==item)
{
cout<<"Element found at position "<<count<<"\n";
flag=1;
break;
}
else
q=q->link;
}
```

```
if(flag==0)
cout<<"Element not found\n";
return 0;
}
```

Output:

```
Enter the size of the linked list
5
Elements in the linked list are as follows:
9       5       8       6       7
Enter the element you want to search
8
Element found at position 3
```

Program No. 7. Write a Program to implement quick sort algorithm.

Maps with: CO1, CO3, CO6, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the operation of sorting elements of an array using quick sort algorithm. This also demonstrates quick sort as an application of stack data structure

Program:

```
#include<iostream>
#include<cstdlib>
using namespace std;
void swap(int *a, int *b) {
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
int Partition(int a[], int l, int h) {
  int pivot, index, i;
  index = l;
  pivot = h;
  for(i = l; i < h; i++) {
    if(a[i] < a[pivot]) {
      swap(&a[i], &a[index]);
      index++;
    }
  }
  swap(&a[pivot], &a[index]);
  return index;
}
int RandomPivotPartition(int a[], int l, int h) {
  int pvt, n, temp;
  n = rand();
  pvt = l + n%(h-l+1);
```

```
    swap(&a[h], &a[pvt]);
    return Partition(a, l, h);
}
int QuickSort(int a[], int l, int h) {
    int pindex;
    if(l < h) {
        pindex = RandomPivotPartition(a, l, h);
        QuickSort(a, l, pindex-1);
        QuickSort(a, pindex+1, h);
    }
    return 0;
}
int main() {
    int n, i;
    cout<<"\nEnter the number of data element to be sorted: ";
    cin>>n;
    int arr[n];
    for(i = 0; i < n; i++) {
        cout<<"\nEnter element "<<i+1<<": ";
        cin>>arr[i];
    }
    QuickSort(arr, 0, n-1);
    cout<<"\nSorted Data ";
    for (i = 0; i < n; i++)
        cout<<"->"<<arr[i];
    return 0;
}
```

Output:

```
Enter the number of data element to be sorted: 4
Enter element 1: 3
Enter element 2: 4
Enter element 3: 7
Enter element 4: 6
```

Sorted Data ->3->4->6->7

Program No. 8. Write a Program to find product of two matrices.

Maps with: CO1, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the use of multidimensional array

Program:

```
#include<iostream>
using namespace std;
class Matrix      //Class definition
{
int **ptrToMatrix;
public:
int row,col;
Matrix(int x,int y);
void getElement(int r,int c, int value)//function to get values in matrix
{
ptrToMatrix[r][c]=value;
}
int & display(int r, int c)//function to display values in matrix
{
return (ptrToMatrix[r][c]);
}
friend Matrix matrixmultiplication(Matrix,Matrix);
};
Matrix :: Matrix(int x, int y)//CONSTRUCTOR to allocate space to each row in matrix
{
row=x;
col=y;
ptrToMatrix=new int *[row];
for(int i=0;i<row;i++)
ptrToMatrix[i]=new int[col];
}
```

```
Matrix matrixmultiplication(Matrix m1,Matrix m2)//FRIEND FUNCTION to multiply matrices
{
 int i,j,k,p;
 p=m1.col;
 Matrix m3(m1.row,m2.col);
 for(i=0;i<m3.row;i++)
 {
 for(j=0;j<m3.col;j++)
 {
 m3.getElement(i,j,0);
 for(k=0;k<p;k++)
 {
 m3.ptrToMatrix[i][j]=m3.ptrToMatrix[i][j]+(m1.ptrToMatrix[i][k])*(m2.ptrToMatrix[k][j]);
 }
 }
 }
 return (m3);
 }
 int main()//main function
 {
 int i,j,r,c,value;
 cout<<"Enter the dimension of first matrix\n";
 cin>>r>>c;
 Matrix mat1(r,c);
 cout<<"Enter the elements of first matrix row by row\n";
 for(i=0;i<r;i++)
 {
 for(j=0;j<c;j++)
 {
 cin>>value;
 mat1.getElement(i,j,value);
 }
 }
 cout<<"Enter the dimension of second matrix\n";
```

```
cin>>r>>c;
Matrix mat2(r,c);
cout<<"Enter the elements of second matrix row by row\n";
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
cin>>value;
mat2.getElement(i,j,value);
}
}
Matrix mat3(mat1.row,mat2.col);//matrix to store the product
mat3=matrixmultiplication(mat1,mat2);
for(i=0;i<mat3.row;i++)
{
for(j=0;j<mat3.col;j++)
{
cout<<mat3.display(i,j)<<"\t";
}
cout<<"\n";
}
return 0;
}
```

Output:

```
Enter the dimension of first matrix
2
3
Enter the elements of first matrix row by row
3
2
5
6
3
```

2

Enter the dimension of second matrix

3

2

Enter the elements of second matrix row by row

1

2

3

4

5

6

34 44

25 36

Program No. 9. Write a Program to insert an element into a BST

Maps with: CO4, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the use of multidimensional array.

Program:

```
#include <iostream>
using namespace std;
class BST
{
        int data;
        BST *left, *right;
        public:
        // Default constructor.
        BST();
        // Parameterized constructor.
        BST(int);
        // Insert function.
        BST* Insert(BST *, int);
        // Inorder traversal.
        void Inorder(BST *);
};
// Default Constructor definition.
BST :: BST() : data(0), left(NULL), right(NULL){}
// Parameterized Constructor definition.
BST :: BST(int value)
{
        data = value;
        left = right = NULL;
}
// Insert function definition.
BST* BST :: Insert(BST *root, int value)
```

```
{
    if(!root)
    {
        // Insert the first node, if root is NULL.
        return new BST(value);
    }
    // Insert data.
    if(value > root->data)
    {
        // Insert right node data, if the 'value'
        // to be inserted is greater than 'root' node data.

        // Process right nodes.
        root->right = Insert(root->right, value);
    }
    else
    {
        // Insert left node data, if the 'value'
        // to be inserted is greater than 'root' node data.

        // Process left nodes.
        root->left = Insert(root->left, value);
    }
    // Return 'root' node, after insertion.
    return root;
}
// Inorder traversal function.
// This gives data in sorted order.
void BST :: Inorder(BST *root)
{
    if(!root)
    {
        return;
    }
```

```
        Inorder(root->left);
        cout << root->data << endl;
        Inorder(root->right);
}
// Driver code
int main()
{
        BST b, *root = NULL;
        root = b.Insert(root, 50);
        b.Insert(root, 30);
        b.Insert(root, 20);
        b.Insert(root, 40);
        b.Insert(root, 70);
        b.Insert(root, 60);
        b.Insert(root, 80);
        b.Inorder(root);
        return 0;
}
```

Output:

```
20
30
40
50
60
70
80
```

Program No. 10. Write a Program to implement Floyd-Warshall's algorithm.

Maps with: CO5, PO1, PO2, PO3, PO4, PO6, PO7

Theory: This program demonstrates the use of Floyd-Warshall's algorithm to find the shortest path among the vertices in a graph

Program:

```
#include <bits/stdc++.h>
using namespace std;
// Number of vertices in the graph
#define V 4
/* Define Infinite as a large enough
value.This value will be used for
vertices not connected to each other */
#define INF 99999
// A function to print the solution matrix
void printSolution(int dist[][V]);
// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall (int graph[][V])
{
        /* dist[][] will be the output matrix
        that will finally have the shortest
        distances between every pair of vertices */
        int dist[V][V], i, j, k;
        /* Initialize the solution matrix same
        as input graph matrix. Or we can say
        the initial values of shortest distances
        are based on shortest paths considering
        no intermediate vertex. */
        for (i = 0; i < V; i++)
                for (j = 0; j < V; j++)
                        dist[i][j] = graph[i][j];
```

```
/* Add all vertices one by one to
the set of intermediate vertices.
---> Before start of an iteration,
we have shortest distances between all
pairs of vertices such that the
shortest distances consider only the
vertices in set {0, 1, 2, .. k-1} as
intermediate vertices.
----> After the end of an iteration,
vertex no. k is added to the set of
intermediate vertices and the set becomes {0, 1, 2, .. k} */
for (k = 0; k < V; k++)
{
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++)
        {
                // Pick all vertices as destination for the
                // above picked source
                for (j = 0; j < V; j++)
                {
                        // If vertex k is on the shortest path from
                        // i to j, then update the value of dist[i][j]
                        if (dist[i][k] + dist[k][j] < dist[i][j])
                                dist[i][j] = dist[i][k] + dist[k][j];
                }
        }
}
// Print the shortest distance matrix
printSolution(dist);
}
/* A utility function to print solution */
void printSolution(int dist[][V])
{
        cout<<"The following matrix shows the shortest distances"
```

```
                    " between every pair of vertices \n";
        for (int i = 0; i < V; i++)
        {
                for (int j = 0; j < V; j++)
                {
                        if (dist[i][j] == INF)
                                cout<<"INF"<<"  ";
                        else
                                cout<<dist[i][j]<<"          ";
                }
                cout<<endl;
        }
}
// Driver code
int main()
{
        /* Let us create the following weighted graph
                        10
        (0)------->(3)
                |       /|\
        5 |     |
                |       | 1
        \|/     |
        (1)------->(2)
                        3           */
        int graph[V][V] = { {0, 5, INF, 10},
                                                {INF, 0, 3, INF},
                                                {INF, INF, 0, 1},
                                                {INF, INF, INF, 0}
                                        };
        // Print the solution
        floydWarshall(graph);
        return 0;
}
```

Output:

Following matrix shows the shortest distances between every pair of vertices

```
  0    5    8    9
INF    0    3    4
INF  INF    0    1
INF  INF  INF    0
```