

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;
    cout<<"Enter the size and elements of an array";
    for(j=n-1;j>=k-1;j--)
    {
        for(j=n;j>j+1)
            cout<<LA[j]<<" ";
        cout<<item;
        cout<<"\n";
        cout<<"*****Array is";
    }
    //Displaying the elements of an array
    for(j=0;j<n;j++)
        cout<<LA[j]<<"\n";
}
```

```

    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:

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

*****Array before INSERTION*****

1 8 5 6 7

Enter the element you want to insert and its position

8

1

*****Array after INSERTION*****

3 1 2 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;
}
```

```
#include <stdio.h>  
int main()  
{  
    int arr[5];
```

Output:

Enter the size and elements of an array:

```
5  
3  
6  
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
//Program to demonstrate bubble sort algorithm
int main()
{
    //Maps with CO1, CO6, PO1, PO2, PO3, PO4, PO6, PO7
    int data[MAX];
    int k, ptr, n, 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++)
    {
        for(ptr=0;ptr<n-k;ptr++)
        {
            if(data[ptr] > data[ptr+1])
            {
                temp = data[ptr];
                data[ptr] = data[ptr+1];
                data[ptr+1] = temp;
            }
        }
    }
    cout << "***** Array after sorting*****\n";
    for(k=1;k<n;k++)
    {
        cout << data[k] << "\t";
    }
    cout << "\n";
}
```

```
        data[ptr+1]=temp;
    }
    ptr++;
    cout<<"The array elements are ";
}
}

cout<<"*****Array after sorting*****\n";
for(i=0;i<n;i++)
{
    cout<<data[i]<<"\t";
    cout<<"\n";
}
return 0;
} // End of main()
}
```

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

Marks 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()
{
    // Input: CO1, CO6, PO1, PO2, PO3, PO4, PO6, PO7
    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;
    if(n==0)
        cout<<"Enter the size and element\n";
    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";
    cout<<"\n";
    cout<<"Enter the element you want to search:\n";
    cin>>item;
    data[n]=item;
    if(n==0)
        cout<<"Enter the size and element\n";
    while(data[i]!=item)
```

```
    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 n,i,j,k,l,m;
    int beg,end,mid,i,n,item;
    int data[MAX];
    cout<<"Enter the size and elements of an array in ascending order\n";
    cout<<"Enter the element you are searching for\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;
    int count,mid,i,n,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;
        cout<<"Enter the element you are searching for\n";
        cin>>item;
        beg=0;
        end=n-1;
        mid=(beg+end)/2;
    }
}
```

```
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;
}
cout<<"Element not found\n";
```

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;
    if(n>MAX)
        do
    {
        cout<<"Enter the size of the linked list\n";
        cin>>n;
    }while(n>MAX);
    cout<<"Enter the elements of the linked list\n";
    for(i=0;i<n;i++)
    {
        cin>>data;
        struct node *temp=new struct node;
        temp->info=data;
        temp->link=NULL;
        if(start==NULL)
        {
            start=temp;
            count++;
        }
        else
        {
            temp->link=start;
            start=temp;
        }
    }
    cout<<"Enter the element to be searched\n";
    cin>>item;
    temp_start=start;
    while(temp_start!=NULL)
    {
        if(temp_start->info==item)
        {
            cout<<"Element found\n";
            flag=1;
            break;
        }
        temp_start=temp_start->link;
    }
    if(flag==0)
        cout<<"Element not found\n";
}
```

```

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<<"Element found\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

Enter the size of the linked list

5

Elements in the linked list are

9 5 8 6

Enter the element you want to

Element found at position 3

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

Topic:

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, l;
    index = l;
    pivot = a[l];
    for(l = l + 1; l < h; l++) {
        if(a[l] < a[pivot]) {
            swap(&a[l], &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() {
    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

cout<<"\nSorted Data "

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

cout<<"->"<<arr[i];

return 0;

}

}

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 row, col;
public:
    Matrix(int x,int y);
    void getElement(int r,int c, int value)//function to get values in matrix
    {
        for(r=0;r<x;r++)
        {
            for(c=0;c<y;c++)
            {
                cout << "Enter element[" << r << "," << c << "]: ";
                cin >> value;
                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,Matrix)
{
    Matrix result;
    result.row=matrix1.row;
    result.col=matrix1.col;
    for(int r=0;r<result.row;r++)
    {
        for(int c=0;c<result.col;c++)
        {
            int sum=0;
            for(int i=0;i<matrix1.col;i++)
            {
                sum+=matrix1.ptrToMatrix[r][i]*matrix2.ptrToMatrix[i][c];
            }
            result.ptrToMatrix[r][c]=sum;
        }
    }
    return result;
}
Matrix Matrix::operator<<(int x, int y)
{
    cout << "Matrix A: " << endl;
    for(int r=0;r<row;r++)
    {
        for(int c=0;c<col;c++)
        {
            cout << ptrToMatrix[r][c] << " ";
        }
        cout << endl;
    }
    cout << "Matrix B: " << endl;
    for(int r=0;r<matrix2.row;r++)
    {
        for(int c=0;c<matrix2.col;c++)
        {
            cout << matrix2.ptrToMatrix[r][c] << " ";
        }
        cout << endl;
    }
    cout << "Product of Matrix A and Matrix B: " << endl;
    for(int r=0;r<result.row;r++)
    {
        for(int c=0;c<result.col;c++)
        {
            cout << result.ptrToMatrix[r][c] << " ";
        }
        cout << endl;
    }
    return result;
}
```

```

Matrix matrixMultiplication(Matrix m1, Matrix m2) //FRIEND FUNCTION to multiply matrices
{
    int r1, c1, r2, c2;
    r1 = m1.row;
    c1 = m1.col;
    r2 = m2.row;
    c2 = m2.col;

    if (c1 != r2)
        cout << "Error! Dimension mismatch." << endl;
    else
    {
        Matrix m3(r1, c2);
        for (int i = 0; i < r1; i++)
        {
            for (int j = 0; j < c2; j++)
            {
                for (int k = 0; k < c1; k++)
                {
                    m3.ptrToMatrix[i][j] = m3.ptrToMatrix[i][j] + (m1.ptrToMatrix[i][k]) * (m2.ptrToMatrix[k][j]);
                }
            }
        }
        return m3;
    }
}

int main() //main function
{
    cout << "Enter the dimension of first matrix\n";
    cout << "Enter the dimension of second matrix\n";
    int r, c, r1, c1, r2, c2;
    cin >> r >> c;
    cin >> r1 >> c1;
    cin >> r2 >> c2;

    Matrix mat1(r, c);
    Matrix mat2(c, r);

    cout << "Enter the elements of first matrix row by row\n";
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
        {
            cout << "Enter element at position (" << i << ", " << j << ")\n";
            cin >> value;
            mat1.getElement(i, j, value);
        }
    }

    cout << "Enter the elements of second matrix row by row\n";
    for (int i = 0; i < c; i++)
    {
        for (int j = 0; j < r; j++)
        {
            cout << "Enter element at position (" << i << ", " << j << ")\n";
            cin >> value;
            mat2.getElement(i, j, value);
        }
    }

    cout << "The result of multiplication is\n";
    cout << mat1.matrixMultiplication(mat2);
}

```

```

    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 b;
}
int main()
{
    matrix multiplication();
}

```

Output:

```

Enter the dimension of first matrix
2
3
Enter the elements of first matrix row by row
3
2
5
6
7
8
Enter the dimension of second matrix
1
dimension of first matrix
1
elements of first matrix

```

2

Enter the dimension of second matrix

36

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
{
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)
{
    // Parameterized constructor.
    data = value;
    left = right = NULL;
}
// Insert function definition.
BST* BST::Insert(BST *root, int value)
{
    if (root == NULL)
    {
        root = new BST(value);
        return root;
    }
    if (value < root->data)
        root->left = Insert(root->left, value);
    else
        root->right = Insert(root->right, value);
    return root;
}

// Inorder traversal.
void BST::Inorder(BST *root)
{
    if (root != NULL)
    {
        Inorder(root->left);
        cout << root->data << " ";
        Inorder(root->right);
    }
}
```

```

if(!root)
{
    // Insert the first node, if root is NULL.
    bst = 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.
}

// Return 'root'.
return root;
}

// Inorder traversal function.
else
{
    // This gives data in sorted order.
    BST *bst = Insert(root->right, value);

    // Insert left node data, if the 'value'
    // to be inserted is greater than 'root' node data.
    if(!root)
    {
        // Process left nodes.
        bst = 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.Inorder(root);
    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
20
30
40
50
60
70
80
```

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

Maps with: C05, P01, P02, P03, P04, P06, P07

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++)
{
    /* Add k-th vertex to the set of
    intermediate vertices */
    // Pick all vertices as source one by one
    for (i = 0; i < V; i++)
        // Initialize shortest distance
        // of i-th vertex to infinity
        dist[i][k] = INT_MAX;
        // Pick all vertices as destination for the
        // shortest distance calculation
        for (j = 0; j < V; j++)
            if (j != i)
                // If vertex k is on the shortest path from
                // vertex 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];

    /* Pick a vertex */
    for (i = 0; i < V; i++)
    {
        /* 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"
    cout<<endl;
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
            cout<<dist[i][j]<<" ";
        cout<<endl;
    }
}

```

```

    cout << endl;
}

" 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)
      // Print graph /| \
      floydWarshall
      5 | | 1
      return 0 | | 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