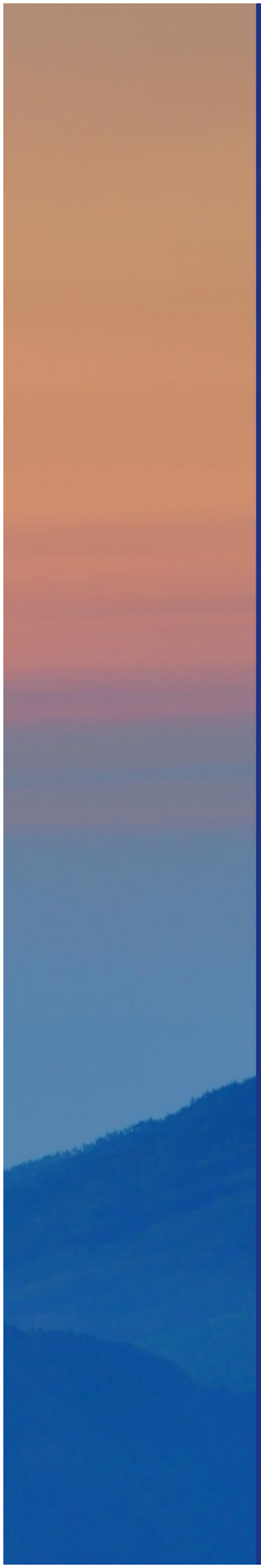


# DISCRETE STRUCTURE

## PRACTICAL FILE


SUBMITTED BY:

ABHISHEK SAHU
18012

SUBMITTED TO:

MRS. SHALINI GUPTA

# Practical 1

## Objective

Write a Program to create a SET A and determine the cardinality of SET for an input array of elements (repetition allowed) and perform the following operations on the SET:
a) ismember (a, A): check whether an element belongs to set or not and return value as true/false.
**b)** powerset(A): list all the elements of power set of A.

## CODE

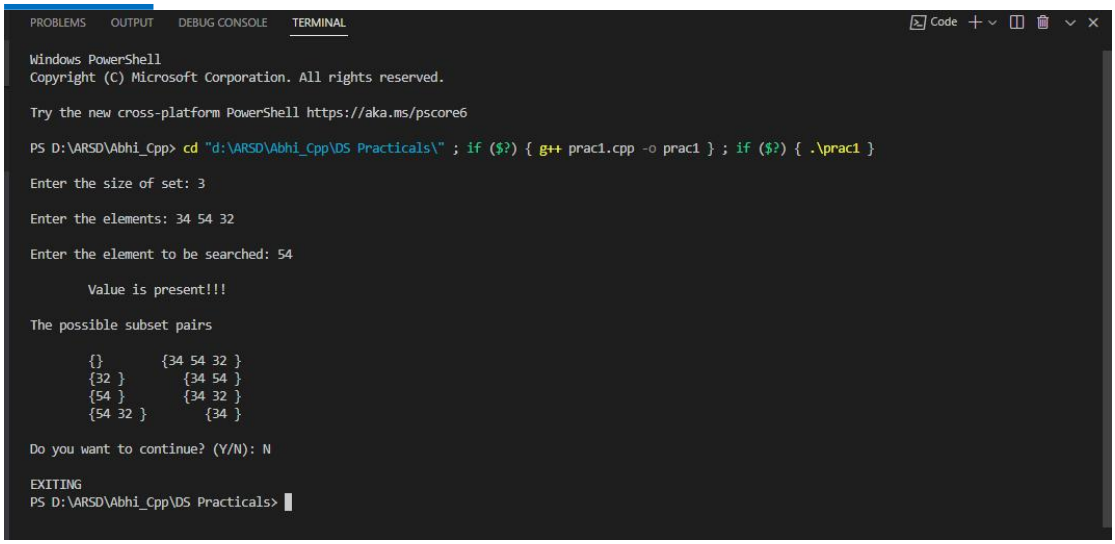
```
#include <iostream>
#include <math.h>
#include <iomanip>

using namespace std;
bool ismember(int size, int A[])
{
    int a;
    cout << "\nEnter the element to be searched: ";
    cin >> a;
    for (int i = 0; i < size; i++)
    {
        if (A[i] == a)
            return true;
    }
    return false;
}
void print(char code[], int arr[], int n)
{
    int i;
    cout << "\t{";
    for (i = 0; i < n; i++)
    {
        if (code[i] == '1')
            cout << arr[i] << " ";
    }
    cout << "}";
    cout << "        {";
    for (i = 0; i < n; i++)
    {
        if (code[i] == '0')
            cout << arr[i] << " ";
    }
```

```
    cout << "}\n";
}
void genUnionSet(int arr[], int n)
{
    int i, r, l;
    char binary[n];
    r = pow(2, n - 1);
    for (i = 0; i < n; i++)
        binary[i] = '0';
    for (i = 0; i < r; i++)
    {
        print(binary, arr, n);
        l = n - 1;
    h:
        if (binary[l] == '0')
            binary[l] = '1';
        else
        {
            binary[l] = '0';
            l--;
            goto h;
        }
    }
}
int main()
{
    bool x;
    int size;
    char ch = 'Y';
    while (ch == 'Y')
    {
        cout << "\nEnter the size of set: ";
        cin >> size;
        int A[size];
        cout << "\nEnter the elements: ";
        for (int i = 0; i < size; i++)
        {
            cin >> A[i];
        }
        x = ismember(size, A);
        if (x == true)
            cout << "\n\tValue is present!!!";
        else
            cout << "\n\tValue is not present!!!";
        cout << "\n\nThe possible subset pairs\n"
             << endl;
        genUnionSet(A, size);
        cout << "\nDo you want to continue? (Y/N): ";
```

```
        cin >> ch;
    }
    cout << "\nEXITING";
    return 0;
}
```

## OUTPUT




# Practical 2

## Objective

Create a class SET and take two sets as input from user to perform following SET Operations:

a) Subset: Check whether one set is a subset of other or not.

b) Union and Intersection of two Sets.

c) Complement: Assume Universal Set as per the input elements from the user.

d) Set Difference and Symmetric Difference between two SETS

e) Cartesian Product of Sets.

## CODE

```
#include <iostream>

using namespace std;
class SET
{
private:
    int i, j;
public:
    void Subset(int *arrA, int sizeA, int *arrB, int sizeB)
```

```
    {
        int c = 0;
        for (i = 0; i < sizeA; i++)
            for (j = 0; j < sizeB; j++)
                if (arrA[i] == arrB[j])
                    c++;
        if (c != sizeA)
            cout << "SET A is not a subset of SET B" << endl;
        else
            cout << "SET A is a subset of SET B" << endl;
        int c1 = 0;
        for (i = 0; i < sizeB; i++)
            for (j = 0; j < sizeA; j++)
                if (arrB[i] == arrA[j])
                    c1++;
        if (c != sizeB)
            cout << "SET B is not a subset of SET A" << endl;
        else
            cout << "SET B is a subset of SET A" << endl;
        cout << "-------------------------------------------------" <<
endl;
    }
    void UnionInter(int *setA, int sizeA, int *setB, int sizeB)
    {
        int uSize = sizeA + sizeB;
        int uSet[uSize];
        int unionSet[uSize];
        int iSet[uSize];
        int x = 0, y = 0;
        for (i = 0; i < sizeA; i++)
        {
            uSet[x] = setA[i];
            x++;
        }
        for (i = 0; i < sizeB; i++)
        {
            uSet[x] = setB[i];
            x++;
        }
        for (i = 0; i < x; i++)
        {
            for (j = i + 1; j < x; j++)
            {
                if (uSet[i] == uSet[j])
                {
                    iSet[y] = uSet[i];
                    y++;
                    for (int k = j; k < x - 1; k++)
```

```
                    uSet[k] = uSet[k + 1];
                x--;
            }
            else
                continue;
        }
    }
    cout << "Union of two sets is : {";
    for (i = 0; i < x; i++)
        cout << uSet[i] << " ";
    cout << "}";
    cout << endl;
    if (y != 0)
    {
        cout << "Intersection of two sets is : {";
        for (i = 0; i < y; i++)
            cout << iSet[i] << " ";
        cout << "}";
    }
    else
        cout << "No intersection found";
    cout << endl;
    cout << "------------------------------------------------" <<
endl;
}
void Complement(int *setA, int sizeA, int *setB, int sizeB)
{
    int sizeU;
    cout << "Enter the no. of elements of universal set : ";
    cin >> sizeU;
    cout << "Enter the elemnts of universal set : ";
    int U[sizeU];
    for (i = 0; i < sizeU; i++)
        cin >> U[i];
    int AC[sizeU], p = 0, c = 0;
    for (i = 0; i < sizeU; i++)
    {
        for (j = 0; j < sizeA; j++)
        {
            if (U[i] == setA[j])
                c++;
            else
                continue;
        }
        if (c == 0)
        {
            AC[p] = U[i];
            p++;
```

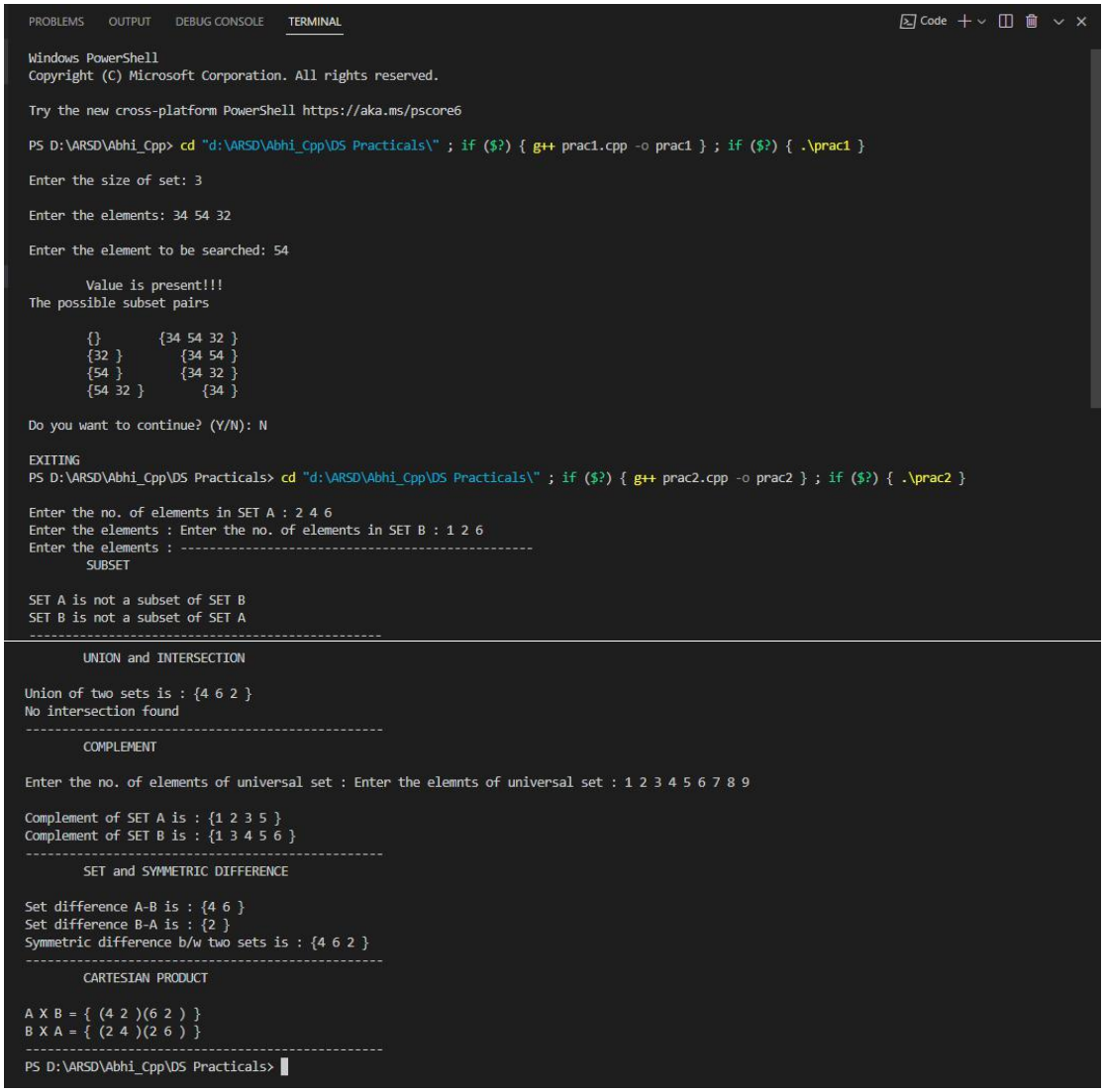
```
            }
            c = 0;
        }
        cout << endl;
        cout << "Complement of SET A is : {";
        for (i = 0; i < p; i++)
            cout << AC[i] << " ";
        cout << "}" << endl;
        int BC[sizeU], q = 0, ctr = 0;
        for (i = 0; i < sizeU; i++)
        {
            for (j = 0; j < sizeB; j++)
            {
                if (U[i] == setB[j])
                    ctr++;
                else
                    continue;
            }
            if (ctr == 0)
            {
                BC[q] = U[i];
                q++;
            }
            ctr = 0;
        }
        cout << "Complement of SET B is : {";
        for (i = 0; i < q; i++)
            cout << BC[i] << " ";
        cout << "}" << endl;
        cout << "-------------------------------------------------" <<
endl;
    }
    void setNSymDiff(int *setA, int sizeA, int *setB, int sizeB)
    {
        int ABDif[100], q = 0, ctr = 0;
        for (i = 0; i < sizeA; i++)
        {
            for (j = 0; j < sizeB; j++)
            {
                if (setA[i] == setB[j])
                    ctr++;
                else
                    continue;
            }
            if (ctr == 0)
            {
                ABDif[q] = setA[i];
                q++;
```

```
            }
            ctr = 0;
        }
        cout << "Set difference A-B is : {";
        for (i = 0; i < q; i++)
            cout << ABDif[i] << " ";
        cout << "}" << endl;
        int BADif[100], p = 0, c = 0;
        for (i = 0; i < sizeB; i++)
        {
            for (j = 0; j < sizeA; j++)
            {
                if (setB[i] == setA[j])
                    c++;
                else
                    continue;
            }
            if (c == 0)
            {
                BADif[p] = setB[i];
                p++;
            }
            c = 0;
        }
        cout << "Set difference B-A is : {";
        for (i = 0; i < p; i++)
            cout << BADif[i] << " ";
        cout << "}" << endl;
        int uSize = q + p;
        int symDif[uSize];
        int x = 0, y = 0;
        for (i = 0; i < q; i++)
        {
            symDif[x] = ABDif[i];
            x++;
        }
        for (i = 0; i < p; i++)
        {
            symDif[x] = BADif[i];
            x++;
        }
        cout << "Symmetric difference b/w two sets is : {";
        for (i = 0; i < x; i++)
            cout << symDif[i] << " ";
        cout << "}";
        cout << endl;
        cout << "-------------------------------------------------" <<
endl;
```

```
    }
    void cartesianPro(int *setA, int sizeA, int *setB, int sizeB)
    {
        int sizeAB, sizeBA, x = 0, y = 0;
        sizeAB = sizeA * sizeB;
        sizeBA = sizeAB;
        int AB[sizeAB * 2], BA[sizeBA * 2];
        for (i = 0; i < sizeA; i++)
        {
            for (j = 0; j < sizeB; j++)
            {
                AB[x++] = setA[i];
                AB[x++] = setB[j];
            }
        }
        for (i = 0; i < sizeB; i++)
        {
            for (j = 0; j < sizeA; j++)
            {
                BA[y++] = setB[i];
                BA[y++] = setA[j];
            }
        }
        cout << "A X B = { ";
        for (i = 0; i < x; i++)
        {
            if (i % 2 == 0)
                cout << "(";
            cout << AB[i] << " ";
            if (i % 2 != 0)
                cout << ")";
        }
        cout << " }" << endl;
        cout << "B X A = { ";
        for (i = 0; i < y; i++)
        {
            if (i % 2 == 0)
                cout << "(";
            cout << BA[i] << " ";
            if (i % 2 != 0)
                cout << ")";
        }
        cout << " }" << endl;
        cout << "-------------------------------------------------" <<
endl;
    }
};
int main()
```

```
{
    cout << endl;
    int i, sizeA, sizeB;
    cout << "Enter the no. of elements in SET A : ";
    cin >> sizeA;
    int arrA[sizeA];
    cout << "Enter the elements : ";
    for (i = 0; i < sizeA; i++)
        cin >> arrA[i];
    cout << "Enter the no. of elements in SET B : ";
    cin >> sizeB;
    int arrB[sizeB];
    cout << "Enter the elements : ";
    for (i = 0; i < sizeB; i++)
        cin >> arrB[i];
    cout << "-----------------------------------------------" << endl;
    SET ob;
    cout << "\tSUBSET\n"
         << endl;
    ob.Subset(arrA, sizeA, arrB, sizeB);
    cout << "\tUNION and INTERSECTION\n"
         << endl;
    ob.UnionInter(arrA, sizeA, arrB, sizeB);
    cout << "\tCOMPLEMENT\n"
         << endl;
    ob.Complement(arrA, sizeA, arrB, sizeB);
    cout << "\tSET and SYMMETRIC DIFFERENCE\n"
         << endl;
    ob.setNSymDiff(arrA, sizeA, arrB, sizeB);
    cout << "\tCARTESIAN PRODUCT\n"
         << endl;
    ob.cartesianPro(arrA, sizeA, arrB, sizeB);
    return 0;
}
```

## OUTPUT




# Practical 3

## Objective

Create a class RELATION, use Matrix notation to represent a relation. Include functions to check if a relation is reflexive, Symmetric, Anti-symmetric and Transitive. Write a Program to use this class.

## CODE

```cpp
#include <iostream>
#include <stdio.h>
#include <conio.h>

using namespace std;
```

```
class RELATION
{
private:
    int i, j, k, x, y, z, ctr, iA, iB, nA, nR, *A, *R, **RM, **T;
public:
    void empty();
    int inputSet();
    void inputRelation();
    void printSet();
    void printRelation();
    void Matrix();
    int reflexive();
    int symmetric();
    bool antiSymmetric();
    bool transitive();
};
void RELATION::empty()
{
    cout << "Set A is empty\n";
    printSet();
    cout << "Set A has no member.";
    cout << "\nHence, relation R is empty.\n";
    nR = 0;
    printRelation();
    cout << "Therefore, no matrix notation.";
    cout << "\nRelation R is NOT REFLEXIVE.";
    symmetric();
    antiSymmetric();
    transitive();
}
int RELATION::inputSet()
{
    cout << "Enter the size of SET A : ";
    cin >> nA;
    A = new int[nA];
    if (nA == 0)
        return 1;
    cout << "Enter the elements : ";
    for (i = 0; i < nA; i++)
        cin >> A[i];
}
void RELATION::inputRelation()
{
    cout << "Enter the no of relations (R on A) : ";
    cin >> nR;
    R = new int[nR * 2];
    cout << "Enter the relations in pair :\n";
    for (i = 0; i < nR * 2; i++)
```

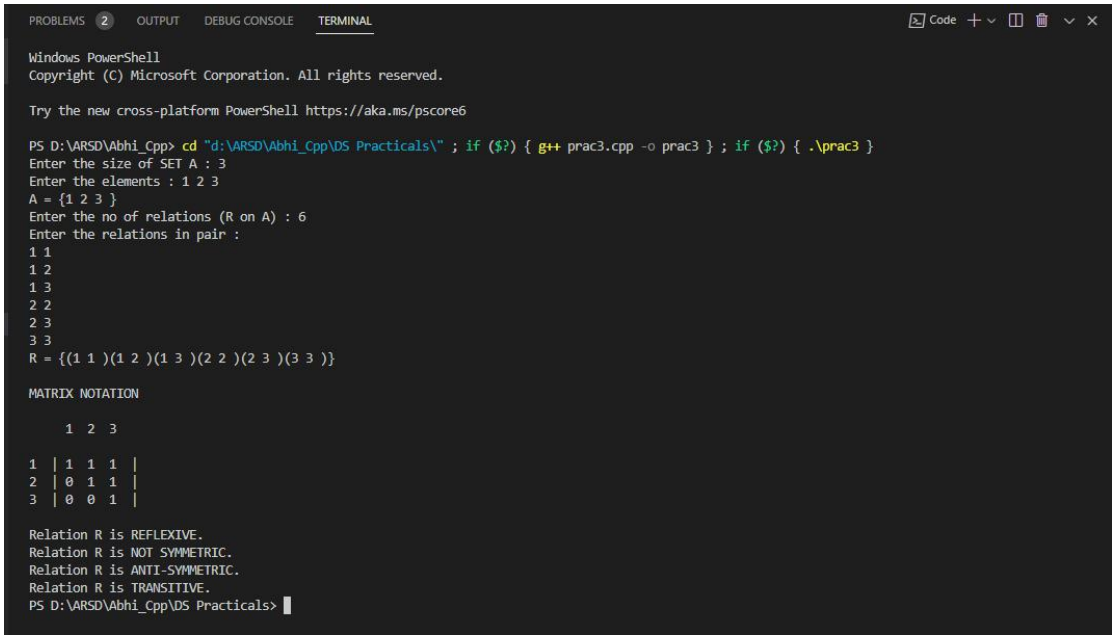
```
        cin >> R[i];
}
void RELATION::printSet()
{
    cout << "A = {";
    for (i = 0; i < nA; i++)
        cout << A[i] << " ";
    cout << "}\n";
}
void RELATION::printRelation()
{
    cout << "R = {";
    for (i = 0; i < nR * 2; i++)
    {
        if (i % 2 == 0)
            cout << "(";
        cout << R[i] << " ";
        if (i % 2 != 0)
            cout << ")";
    }
    cout << "}\n";
}
void RELATION::Matrix()
{
    cout << "\nMATRIX NOTATION\n\n";
    RM = new int *[nA];
    for (i = 0; i < nA; i++)
        RM[i] = new int[nA];
    for (i = 0; i < nA; i++)
    {
        for (j = 0; j < nA; j++)
        {
            RM[i][j] = 0;
        }
    }
    for (i = 0; i < nR * 2; i += 2)
    {
        for (j = 0; j < nA; j++)
        {
            if (R[i] == A[j])
            {
                iA = j;
                break;
            }
        }
        for (k = 0; k < nA; k++)
        {
            if (R[i + 1] == A[k])
```

```
            {
                iB = k;
                break;
            }
        }
        RM[iA][iB] = 1;
    }
    cout << "    ";
    for (int x = 0; x < nA; x++)
        cout << " " << A[x] << " ";
    cout << endl
         << endl;
    for (i = 0; i < nA; i++)
    {
        cout << A[i] << "  | ";
        for (j = 0; j < nA; j++)
        {
            cout << RM[i][j] << "  ";
        }
        cout << "|";
        cout << endl;
    }
}
int RELATION::reflexive()
{
    x = 0;
    for (i = 0; i < nA; i++)
    {
        if (RM[i][i] == 1)
            x++;
    }
    if (x == nA)
    {
        cout << "\nRelation R is REFLEXIVE.";
        return x = 0;
    }
    else
    {
        cout << "\nRelation R is NOT REFLEXIVE.";
        return x = 1;
    }
}
int RELATION::symmetric()
{
    ctr = 0;
    for (i = 0; i < nA; i++)
    {
        for (j = 0; j < nA; j++)
```

```
        {
            if (RM[i][j] == RM[j][i])
                continue;
            else
            {
                ctr++;
                break;
            }
        }
    }
    if (ctr != 0)
        cout << "\nRelation R is NOT SYMMETRIC.";
    else
        cout << "\nRelation R is SYMMETRIC.";
    return ctr;
}
bool RELATION::antiSymmetric()
{
    bool flag = true;
    for (i = 0; i < nR * 2; i += 2)
    {
        for (j = 0; j < nR * 2; j += 2)
        {
            if ((R[i] == R[j + 1]) && (R[i + 1] == R[j]))
                if (R[i] == R[i + 1])
                {
                    continue;
                }
                else
                {
                    flag = false;
                }
        }
    }
    if (flag != true)
        cout << "\nRelation R is NOT ANTI-SYMMETRIC.";
    else
        cout << "\nRelation R is ANTI-SYMMETRIC.";
    return flag;
}
bool RELATION::transitive()
{
    bool flag = true;
    for (i = 0; i < nR * 2; i += 2)
    {
        for (j = 0; j < nR * 2; j += 2)
        {
            if (R[i + 1] == R[j])
```

```
                for (k = 0; k < nR * 2; k += 2)
                {
                    if ((R[k] == R[i]) && (R[k + 1] == R[j + 1]))
                    {
                        flag = true;
                        break;
                    }
                    else
                        flag = false;
                }
        }
    }
    if (flag != true)
        cout << "\nRelation R is NOT TRANSITIVE.";
    else
        cout << "\nRelation R is TRANSITIVE.";
    return flag;
}
int main()
{
    int p = 0;
    RELATION ob;
    p = ob.inputSet();
    if (p == 1)
        ob.empty();
    else
    {
        ob.printSet();
        ob.inputRelation();
        ob.printRelation();
        ob.Matrix();
        ob.reflexive();
        ob.symmetric();
        ob.antiSymmetric();
        ob.transitive();
    }
    return 0;
}
```

## OUTPUT




# Practical 4

## Objective

Use the functions defined in Ques 3 to find check whether the given relation is:
a) Equivalent, or
b) Partial Order relation, or
c) None

## CODE

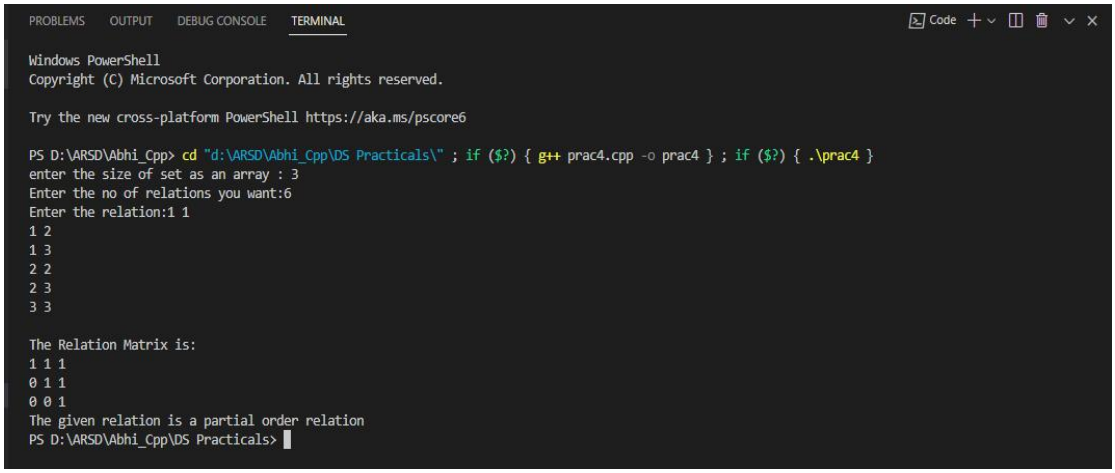
```cpp
#include <iostream>
using namespace std;
class relation
{
    int **ar;
    int n;
public:
    void input();
    bool reflexive();
    bool symmetric();
    bool antisymmetric();
    bool transitive();
    void display();
};
```

```
void relation::input()
{
    cout << "enter the size of set as an array : ";
    cin >> n;
    ar = new int *[n];
    for (int i = 0; i < n; i++)
        ar[i] = new int[n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            ar[i][j] = 0;
        }
    }
    int m;
    cout << "Enter the no of relations you want:";
    cin >> m;
    int a[m], b[m];
    cout << "Enter the relation:";
    for (int i = 0; i < m; i++)
        cin >> a[i] >> b[i];
    for (int i = 0; i < m; i++)
        ar[(a[i] - 1)][(b[i] - 1)] = 1;
}
void relation::display()
{
    cout << "\nThe Relation Matrix is:\n";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << ar[i][j] << " ";
        }
        cout << endl;
    }
}
bool relation::reflexive()
{
    int f = 1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j && ar[i][j] != 1)
            {
                f = 0;
                break;
            }
```

```
        }
        if (f == 0)
            break;
    }
    if (f == 1)
        return true;
    else
        return false;
}
bool relation::symmetric()
{
    int f = 1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (ar[i][j] != ar[j][i])
            {
                f = 0;
                break;
            }
        }
        if (f == 0)
            break;
    }
    if (f == 1)
        return true;
    else
        return false;
}
bool relation::transitive()
{
    int f = 1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (ar[i][j] == 1)
            {
                for (int x = 0; x < n; x++)
                {
                    if (ar[j][x] == 1 && ar[i][x] != 1)
                    {
                        f = 0;
                        break;
                    }
                }
            }
```

```
            if (f == 0)
                break;
        }
        if (f == 0)
            break;
    }
    if (f == 1)
        return true;
    else
        return false;
}
bool relation::antisymmetric()
{
    int f = 1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (ar[i][j] == 1 && ar[j][i] == 1 && i != j)
            {
                f = 0;
                break;
            }
        }
        if (f == 0)
            break;
    }
    if (f == 1)
        return true;
    else
        return false;
}
int main()
{
    relation r;
    r.input();
    r.display();
    if (r.reflexive() && r.symmetric() && r.transitive())
        cout << "The given relation is Equivalence";
    else if (r.reflexive() && r.antisymmetric() && r.transitive())
        cout << "The given relation is a partial order relation";
    else
        cout << "The relation is neither equivalence nor partial order
relation";
    return 0;
}
```

## OUTPUT




# Practical 5

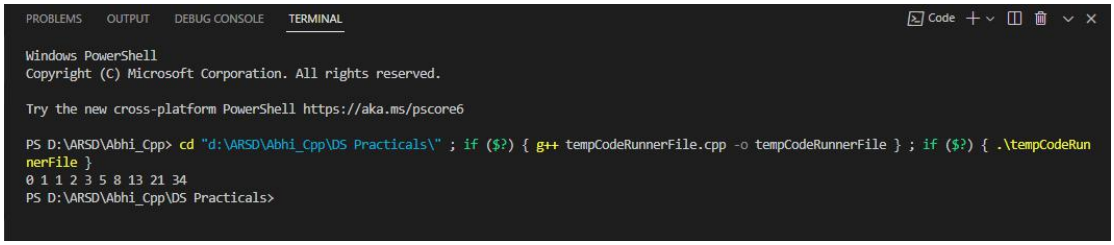
## Objective

Write a Program to generate the Fibonacci Series using recursion.

## CODE

```cpp
#include <iostream>
using namespace std;
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}
int main()
{
    int n = 10, i;
    for (i = 0; i < n; i++)
        cout << fib(i) << " ";
    return 0;
}
```

## OUTPUT




# Practical 6

## Objective

**Write a Program to implement Tower of Hanoi using recursion.**

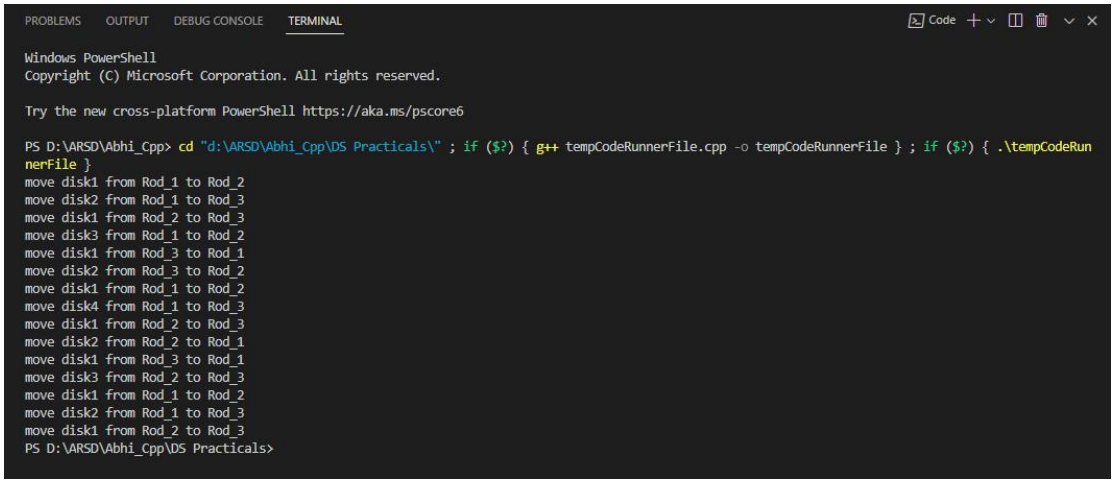
## CODE

```cpp
#include <iostream>
#include <string>
using namespace std;

void TowerOfHanoi(int start, int end, string starting, string temporary,
string destination)
{
    if (start > end)
    {
        return;
    }
    else
    {
        TowerOfHanoi(start, end - 1, starting, destination, temporary);
        cout << "move disk" << end << " from " << starting << " to " <<
destination << endl;
        TowerOfHanoi(start, end - 1, temporary, starting, destination);
    }
}
```

```cpp
int main()
{
    TowerOfHanoi(1, 4, "Rod_1", "Rod_2", "Rod_3");
    return 0;
}
```

## OUTPUT




# Practical 7

## Objective

**Write a Program to implement binary search using recursion.**

## CODE

```cpp
#include <iostream>
using namespace std;

int BinarySearch(int arr[], int n, int s, int e)
{
    int mid;
    if (s > e)
    {
        cout << "Number is not found";
        return 0;
    }
    else
    {
        mid = (s + e) / 2;
        if (arr[mid] == n)
        {
            cout << "Number is found at " << mid << " index \n";
            return 0;
        }
        else if (n > arr[mid])
        {
            BinarySearch(arr, n, mid + 1, e);
        }
        else if (n < arr[mid])
```

```
        {
            BinarySearch(arr, n, s, mid - 1);
        }
    }
}
int main()
{
    int arr[100], num, i, n, s, e;
    cout << "Enter the size of an array : ";
    cin >> n;
    cout << "Enter the sorted values :";
    for (i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    cout << "Enter a value to be search :  \n";
    cin >> num;
    s = 0;
    e = n - 1;
    BinarySearch(arr, num, s, e);
    return 0;
}
```

## OUTPUT


# Practical 8

## Objective

**Write a Program to implement Bubble Sort. Find the number of comparisons during each pass and display the intermediate result. Use the observed values to plot a graph to analyse the complexity of algorithm.**

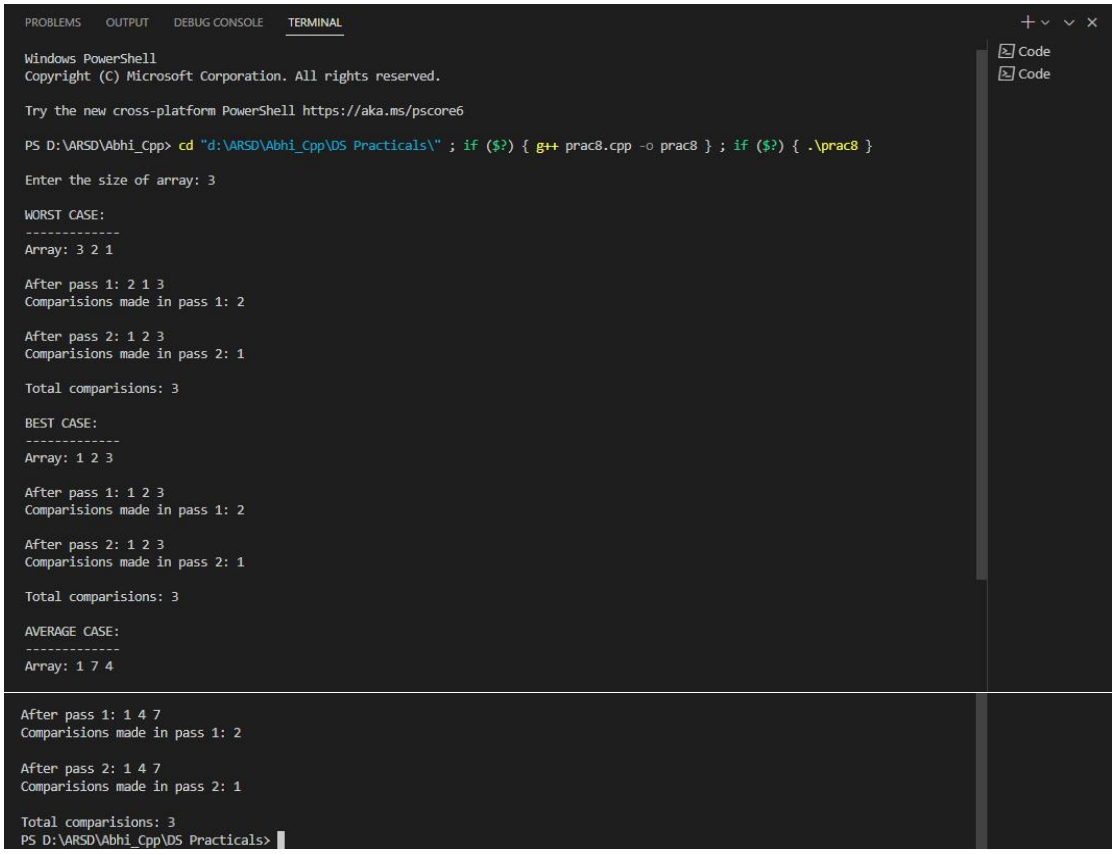
## CODE

```cpp
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <cstdlib>

using namespace std;
int i, j, k;
void bubbleSort(int *, int);
int main()
{
    int size, ele;
    cout << "\nEnter the size of array: ";
    cin >> size;
    int array[size];
    cout << "\nWORST CASE:";
    cout << "\n-------------\n";
    for (i = 0; i < size; i++)
        array[i] = size - i;
    bubbleSort(array, size);
    cout << "\n\nBEST CASE:";
    cout << "\n-------------\n";
    for (i = 0; i < size; i++)
        array[i] = i + 1;
    bubbleSort(array, size);
    cout << "\n\nAVERAGE CASE:";
    cout << "\n-------------\n";
    for (i = 0; i < size; i++)
    {
        ele = ((int)rand() % 10);
        if (ele == 0)
            continue;
        else
            array[i] = ele;
    }
    bubbleSort(array, size);
    return 0;
}
void bubbleSort(int *array, int size)
{
    int temp = 0;
    int ctr = 0;
    int totalCom = 0;
    cout << "Array: ";
    for (i = 0; i < size; i++)
        cout << array[i] << " ";
    cout << endl
         << endl;
```

```
    for (i = 0; i < size - 1; i++)
    {
        ctr = 0;
        for (j = 0; j < size - i - 1; j++)
        {
            if (array[j + 1] < array[j])
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
            ctr++;
            totalCom++;
        }
        cout << "After pass " << i + 1 << ": ";
        for (k = 0; k < size; k++)
            cout << array[k] << " ";
        cout << "\nComparisions made in pass " << i + 1 << ": " << ctr;
        cout << endl
             << endl;
    }
    cout << "Total comparisions: " << totalCom;
}
```

## OUTPUT



# Practical 9

## Objective

Write a Program to implement Insertion Sort. Find the number of comparisons during each pass and display the intermediate result. Use the observed values to plot a graph to analyse the complexity of algorithm.

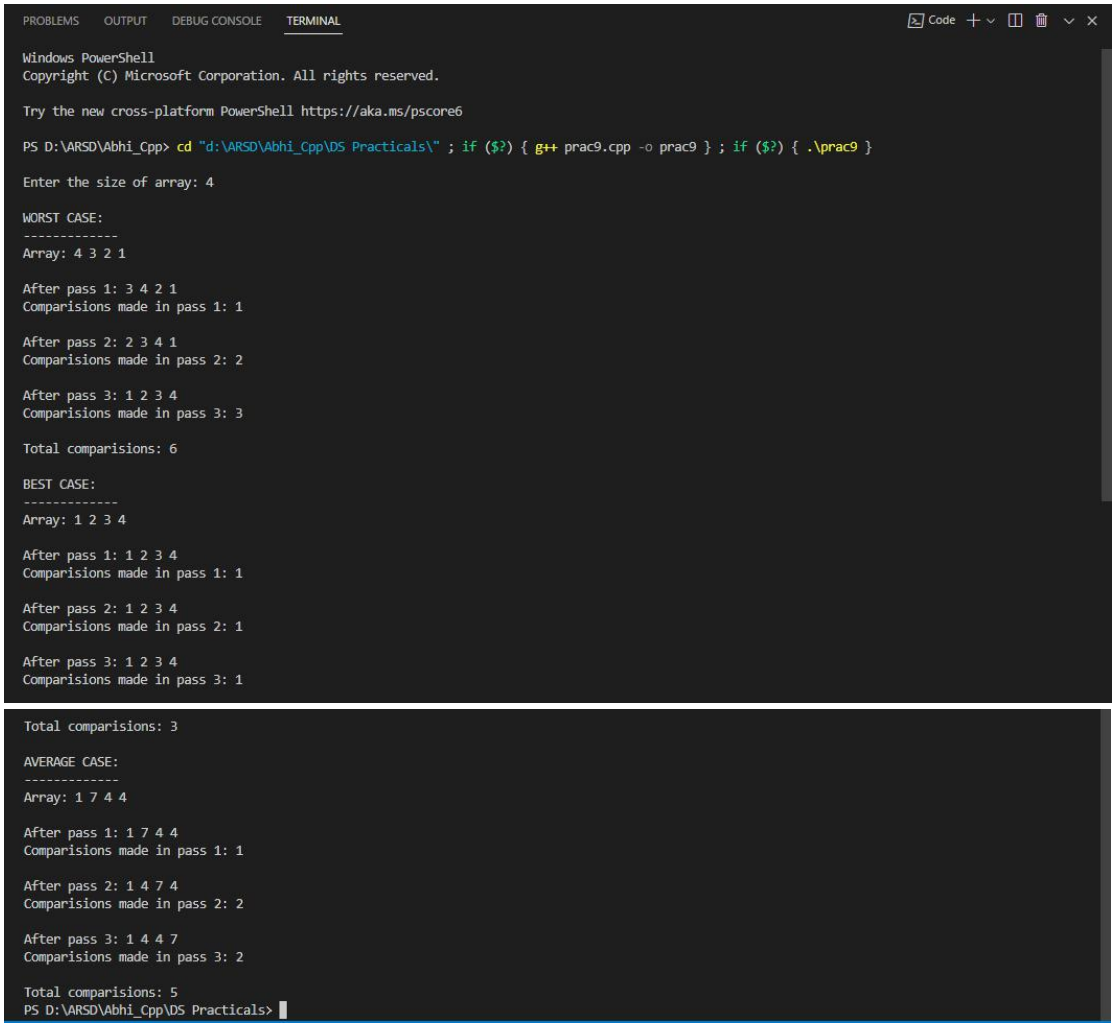
## CODE

```
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <cstdlib>

using namespace std;
int i, j, k;
void insertionSort(int *, int);
int main()
{
    int size, ele;
    cout << "\nEnter the size of array: ";
    cin >> size;
    int array[size];
    cout << "\nWORST CASE:";
    cout << "\n-------------\n";
    for (i = 0; i < size; i++)
        array[i] = size - i;
    insertionSort(array, size);
    cout << "\n\nBEST CASE:";
    cout << "\n-------------\n";
    for (i = 0; i < size; i++)
        array[i] = i + 1;
    insertionSort(array, size);
    cout << "\n\nAVERAGE CASE:";
    cout << "\n-------------\n";
    for (i = 0; i < size; i++)
    {
        ele = ((int)rand() % 10);
        if (ele == 0)
            continue;
        else
            array[i] = ele;
    }
```

```
    insertionSort(array, size);
    return 0;
}
void insertionSort(int *array, int size)
{
    int temp = 0;
    int ctr = 0;
    int totalCom = 0;
    cout << "Array: ";
    for (i = 0; i < size; i++)
        cout << array[i] << " ";
    cout << endl
         << endl;
    for (i = 1; i < size; i++)
    {
        temp = array[i];
        ctr = 0;
        for (j = i - 1; j >= 0; j--)
        {
            ctr++;
            totalCom++;
            if (array[j] > temp)
            {
                array[j + 1] = array[j];
            }
            else
                break;
        }
        array[j + 1] = temp;
        cout << "After pass " << i << ": ";
        for (k = 0; k < size; k++)
            cout << array[k] << " ";
        cout << "\nComparisions made in pass " << i << ": " << ctr;
        cout << endl
             << endl;
    }
    cout << "Total comparisions: " << totalCom;
}
```

## OUTPUT




# Practical 10

## Objective

**Write a Program that generates all the permutations of a given set of digits, with or without repetition. (For example, if the given set is {1,2}, the permutations are 12 and 21). (One method is given in Liu).**

## CODE

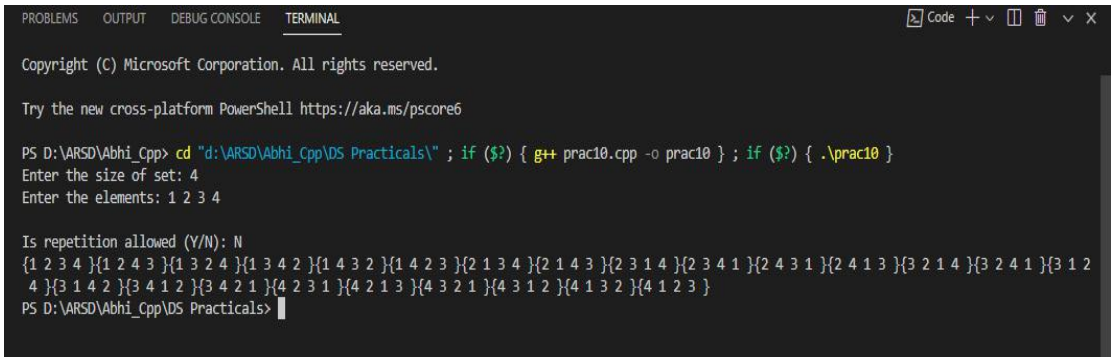
```
#include <iostream>
#include <stdio.h>
#include <conio.h>
#define MAX_DIM 100

using namespace std;
void withRepetition(int *, int);
void withoutRepetition(int *, int);
```

```
void printWithRepetition(int *, int, int *, int, int);
void printWithoutRepetition(int *, int, int, int);
void swap(int &, int &);
int main()
{
    int size;
    char ch;
    cout << "Enter the size of set: ";
    cin >> size;
    int array[MAX_DIM];
    cout << "Enter the elements: ";
    for (int i = 0; i < size; i++)
        cin >> array[i];
    cout << "\nIs repetition allowed (Y/N): ";
    cin >> ch;
    switch (ch)
    {
    case 'Y':
        withRepetition(array, size);
        break;
    case 'N':
        withoutRepetition(array, size);
        break;
    default:
        cout << "\nWrong Choice";
    }
    return 0;
}
void withRepetition(int *array, int size)
{
    int data[MAX_DIM] = {0};
    printWithRepetition(array, size, data, size - 1, 0);
    cout << endl;
}
void printWithRepetition(int *array, int size, int *data, int last, int
index)
{
    for (int i = 0; i < size; i++)
    {
        data[index] = array[i];
        if (index == last)
        {
            cout << "{";
            for (int j = 0; j < index + 1; j++)
                cout << data[j] << " ";
            cout << "}";
        }
        else
```

```
        {
            printWithRepetition(array, size, data, last, index + 1);
        }
    }
}
void withoutRepetition(int *array, int size)
{
    printWithoutRepetition(array, size, 0, size - 1);
    cout << endl;
}
void printWithoutRepetition(int *array, int size, int start, int end)
{
    if (start == end)
    {
        cout << "{";
        for (int i = 0; i < size; i++)
            cout << array[i] << " ";
        cout << "}";
    }
    else
    {
        for (int i = start; i < end + 1; i++)
        {
            swap(array[start], array[i]);
            printWithoutRepetition(array, size, start + 1, end);
            swap(array[start], array[i]);
        }
    }
}
void swap(int &a, int &b)
{
    int t = b;
    b = a;
    a = t;
}
```

## OUTPUT



# Practical 11

## Objective

**Write a Program to calculate Permutation and Combination for an input value n and r using recursive formula of n Cr and n Pr .**

## CODE

```
#include <iostream>

using namespace std;
int nCr(int, int);
int nPr(int, int);
int nPr(int n, int r)
{
    if (r == 0)
        return 1;
    if (r > n)
        return 0;
    return nPr(n - 1, r) + r * nPr(n - 1, r - 1);
}
int nCr(int n, int r)
{
    if (r == 0 || r == n)
        return 1;
    return nCr(n - 1, r) + nCr(n - 1, r - 1);
}
int main()
{
    int n, r;
    cout << "\nEnter the value of n: ";
    cin >> n;
    cout << "\nEnter the value of r: ";
    cin >> r;
    cout << "\nPERMUTATION "
         << "P(" << n << ", " << r << "): " << nPr(n, r);
    cout << "\nCOMBINATION "
         << "C(" << n << ", " << r << "): " << nCr(n, r);
    return 0;
}
```

## OUTPUT


# Practical 12

## Objective

For any number n, write a program to list all the solutions of the equation x1 + x2 + x3 + ...+ xn = C, where C is a constant (C<=10) and x1, x2,x3,...,xn are nonnegative integers using brute force strategy.

## CODE

```cpp
#include <iostream>

using namespace std;
void bruteForce(int *, int, int *, int, int, int, int &);
int main()
{
    int n, C, counter = 0, size = 11;
    int arr[size], data[100] = {0};
    cout << "\nFinding solutions to x1 + x2 + ... + xn = C\n";
    cout << "Enter the value of n: ";
    cin >> n;
    for (int i = 0; i <= 10; i++)
        arr[i] = i;
    cout << "Enter the sum constant (C <= 10): ";
    cin >> C;
    cout << "Possible Non-negative Integral solutions [ ";
    for (int i = 0; i < n; i++)
        cout << "x" << i + 1 << " ";
    cout << " ] :" << endl;
    bruteForce(arr, size, data, n - 1, 0, C, counter);
    cout << "\nFound " << counter << " Solutions\n";
    return 0;
}
void bruteForce(int *arr, int size, int *data, int last, int index, int
C, int &counter)
{
    for (int i = 0; i < size; i++)
```

```
    {
        data[index] = arr[i];
        if (index == last)
        {
            int sum = 0;
            for (int j = 0; j < index + 1; j++)
                sum += data[j];
            if (sum == C)
            {
                cout << "[ ";
                for (int j = 0; j < index + 1; j++)
                    cout << data[j] << " ";
                cout << "] ";
                counter++;
            }
        }
        else
            bruteForce(arr, size, data, last, index + 1, C, counter);
    }
}
```

## OUTPUT


# Practical 13

## Objective

Write a Program to accept the truth values of variables x and y, and print the truth table of the following logical operations:

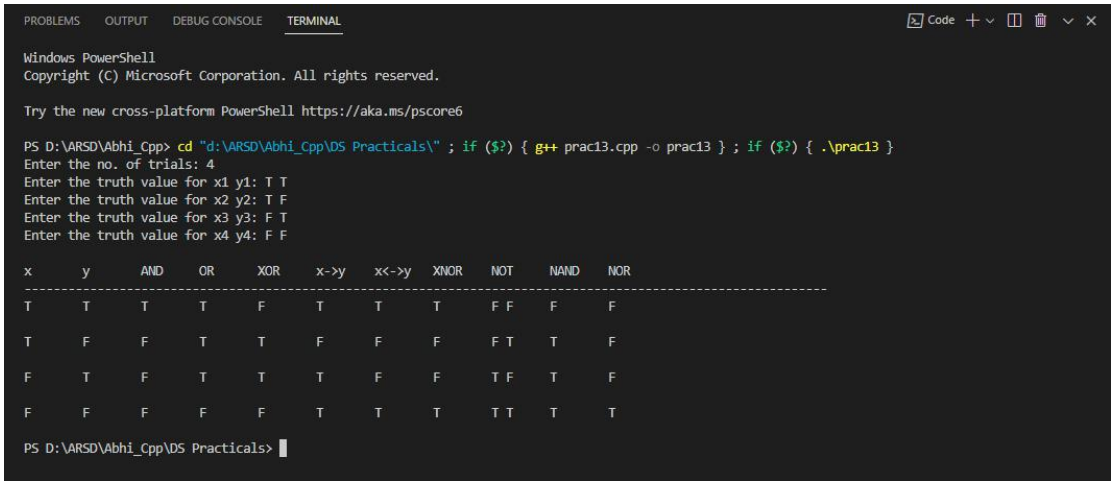
a) Conjunction
b) Disjunction
c) Exclusive OR
d) Conditional
e) Bi-conditional
f) Exclusive NOR
g) Negation
h) NAND
I) NOR

## CODE

```
#include <iostream>
using namespace std;

int main()
{
    int n;
    char x, y;
    cout << "Enter the no. of trials: ";
    cin >> n;
    bool value[n][2];
    for (int i = 0; i < n; i++)
    {
        cout << "Enter the truth value for x" << i + 1 << " y" << i + 1
<< ": ";
        cin >> x >> y;
        value[i][0] = (x == 't' || x == 'T');
        value[i][1] = (y == 't' || y == 'T');
    }
    cout << endl;
    cout << "x\ty\tAND\tOR\tXOR\tx->y\tx<->y\tXNOR\tNOT\tNAND\tNOR";
    cout << "\n-----------------------------------------------------------"
         << "-----------------------------------------------------\n";
    for (int i = 0; i < n; i++)
    {
        int x = value[i][0], y = value[i][1];
        cout << (x ? "T" : "F") << "\t" << (y ? "T" : "F") << "\t"
             << ((x && y) ? "T" : "F") << "\t"
             << ((x || y) ? "T" : "F") << "\t"
             << (((x || y) && !(x && y)) ? "T" : "F") << "\t"
             << ((!x || y) ? "T" : "F") << "\t"
             << (((!x || y) && (!y || x)) ? "T" : "F") << "\t"
             << ((!((x || y) && !(x && y))) ? "T" : "F") << "\t"
             << ((!x) ? "T" : "F") << " " << ((!y) ? "T" : "F") << "\t"
             << (!(x && y) ? "T" : "F") << "\t"
             << (!(x || y) ? "T" : "F") << "\n";
        cout << endl;
    }
    return 0;
}
```

## OUTPUT




# Practical 14

## Objective

Write a program to accept an input n from the user and graphically represent the values of T(n) where n varies from 0 to n for the recurrence relations. For e.g. T(n) = T(n-1) + n, T(0) = 1, T(n) = T(n-1) + n^2, T(0) =1, T(n) = 2*T(n)/2 + n, T(1)=1.

## CODE

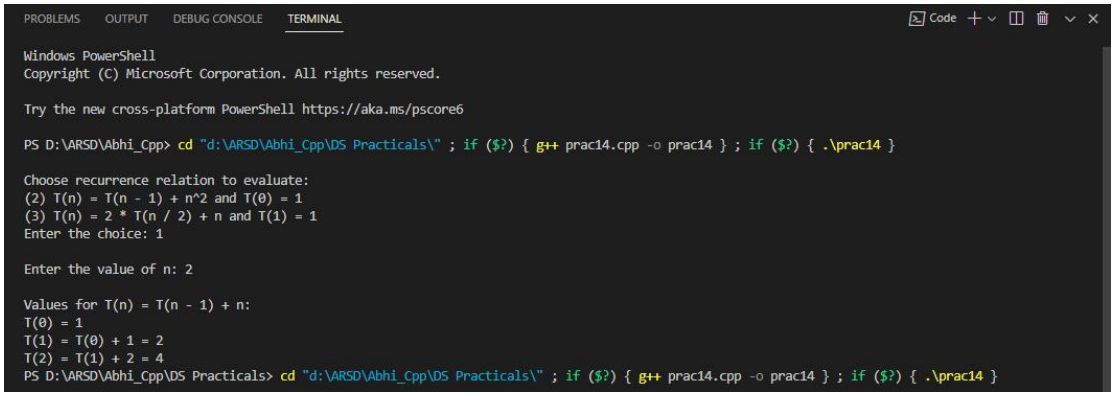
```cpp
#include <iostream>

using namespace std;
int firstRecurrence(int n)
{
    if (n == 0)
        return 1;
    return firstRecurrence(n - 1) + n;
}
int secondRecurrence(int n)
{
    if (n == 0)
        return 1;
    return secondRecurrence(n - 1) + n * n;
}
int thirdRecurrence(int n)
{
    if (n == 1)
        return 1;
    return 2 * thirdRecurrence(n / 2) + n;
}
```

```
int main()
{
    int n, ch;
    cout << "\nChoose recurrence relation to evaluate:\n"
         << "(1) T(n) = T(n - 1) + n and T(0) = 1\n"
         << "(2) T(n) = T(n - 1) + n^2 and T(0) = 1\n"
         << "(3) T(n) = 2 * T(n / 2) + n and T(1) = 1\n";
    cout << "Enter the choice: ";
    cin >> ch;
    switch (ch)
    {
    case 1:
        cout << "\nEnter the value of n: ";
        cin >> n;
        cout << "\nValues for T(n) = T(n - 1) + n:\n";
        for (int i = 0; i <= n; i++)
        {
            if (i == 0)
                cout << "T(0) = " << firstRecurrence(i) << endl;
            else
                cout << "T(" << i << ") = T(" << (i - 1) << ") + "
                     << i << " = "
                     << firstRecurrence(i) << endl;
        }
        break;
    case 2:
        cout << "\nEnter the value of n: ";
        cin >> n;
        cout << "\nValues for T(n) = T(n - 1) + n^2:\n";
        for (int i = 0; i <= n; i++)
        {
            if (i == 0)
                cout << "T(0) = " << secondRecurrence(i) << endl;
            else
                cout << "T(" << i << ") = T(" << (i - 1) << ") + "
                     << i * i << " = "
                     << secondRecurrence(i) << endl;
        }
        break;
    case 3:
        cout << "\nEnter the value of n: ";
        cin >> n;
        cout << "\nValues for T(n) = 2 * T(n / 2) + n:\n";
        for (int i = 1; i <= n; i++)
        {
            if (i == 1)
                cout << "T(1) = " << thirdRecurrence(i) << endl;
            else
```

```
                cout << "T(" << i << ") = 2 * T(" << i << " / 2) + "
                     << i << " = "
                     << "2 * T(" << i / 2 << ") + "
                     << i << " = "
                     << thirdRecurrence(i) << endl;
        }
        break;
    default:
        cout << "\nWrong choice!!!";
        break;
    }
    return 0;
}
```

## OUTPUT




# Practical 15

## Objective

Write a Program to store a function (polynomial/exponential), and then evaluate the polynomial. (For example store f(x) = 4n3 + 2n + 9 in an array and for a given value of n, say n = 5, evaluate (i.e. compute the value of f(5)).

## CODE

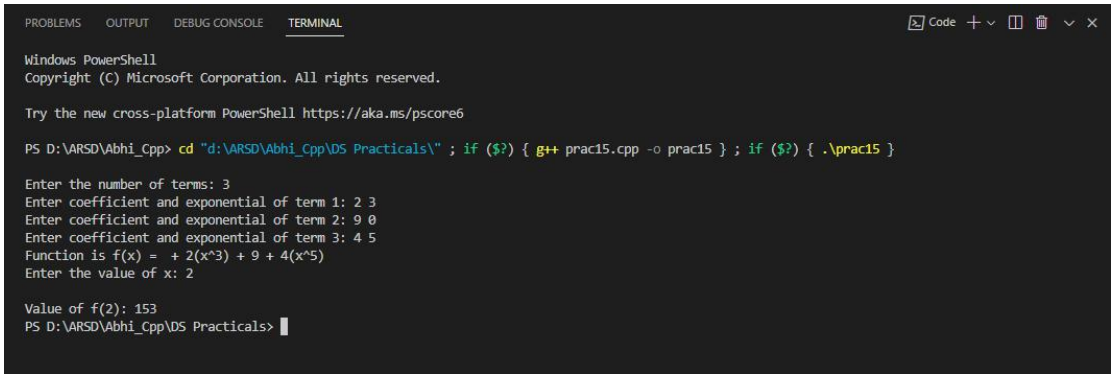
```
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <cmath>

using namespace std;
int i;
```

```
class FUNCTION
{
private:
    int n;
    double *coefficient;
    double *exponential;
public:
    void input();
    void display();
    double evaluate(double);
};
void FUNCTION::input()
{
    int n;
    cout << "\nEnter the number of terms: ";
    cin >> this->n;
    coefficient = new double[n];
    exponential = new double[n];
    for (i = 0; i < this->n; i++)
    {
        cout << "Enter coefficient and exponential of term " << i + 1
<< ": ";
        cin >> coefficient[i] >> exponential[i];
    }
}
void FUNCTION::display()
{
    for (i = 0; i < this->n; i++)
    {
        if (coefficient[i] >= 0)
            cout << " + ";
        else
            cout << " - ";
        cout << abs(coefficient[i]);
        if (exponential[i] != 0)
            cout << "(x^" << exponential[i] << ")";
    }
}
double FUNCTION::evaluate(double x)
{
    double result = 0.0;
    for (i = 0; i < this->n; i++)
    {
        result += coefficient[i] * (pow(x, exponential[i]));
    }
    return result;
}
int main()
```

```
{
    double x;
    FUNCTION ob;
    ob.input();
    cout << "Function is f(x) = ";
    ob.display();
    cout << "\nEnter the value of x: ";
    cin >> x;
    cout << "\nValue of f(" << x << "): " << ob.evaluate(x) << endl;
    return 0;
}
```

## OUTPUT




# Practical 16

## Objective

**Write a Program to represent Graphs using the Adjacency Matrices and check if it is a complete graph.**

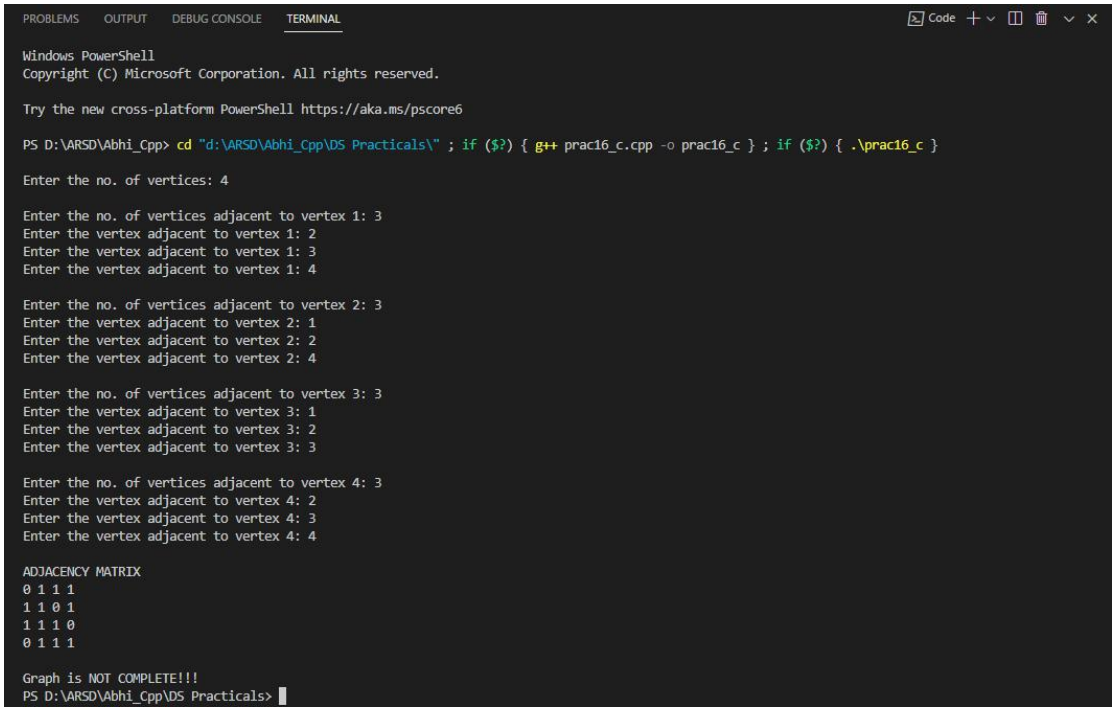
## CODE

```
#include <iostream>

using namespace std;
int main()
{
    int n, c = 0, x, p;
    cout << "\nEnter the no. of vertices: ";
    cin >> n;
    int matrix[n][n];
    for (int i = 0; i < n; i++)
```

```
        for (int j = 0; j < n; j++)
            matrix[i][j] = 0;
    for (int i = 0; i < n; i++)
    {
        cout << "\nEnter the no. of vertices adjacent to vertex " << i
+ 1 << ": ";
        cin >> x;
        for (int j = 0; j < x; j++)
        {
            cout << "Enter the vertex adjacent to vertex " << i + 1 <<
": ";
            cin >> p;
            for (int a = 0; a < n; a++)
                if (a + 1 == p)
                {
                    matrix[i][a] = 1;
                    break;
                }
        }
    }
    cout << "\nADJACENCY MATRIX\n";
    for (int i = 0; i < n; i++)
    {
        int sum = 0;
        for (int j = 0; j < n; j++)
        {
            cout << matrix[i][j] << " ";
            if (matrix[i][i] == 0)
                sum += matrix[i][j];
        }
        cout << endl;
        if (sum == (n - 1))
            c++;
    }
    if (c == n)
        cout << "\nGraph is COMPLETE!!!";
    else
        cout << "\nGraph is NOT COMPLETE!!!";
    return 0;
}
```

## OUTPUT




# Practical 17

## Objective

**Write a Program to accept a directed graph G and compute the in-degree and out-degree of each vertex.**

## CODE

```
#include <bits/stdc++.h>
using namespace std;

// Function to print the in and out degrees
// of all the vertices of the given graph
void findInOutDegree(vector<vector<int>> adjlist,
                     int n)
{
    vector<int> iN(n, 0);
    vector<int> ouT(n, 0);
    for (int i = 0; i < n; i++)
    {
        // Out degree for ith vertex will be the count
        // of direct paths from i to other vertices
        ouT[i] = adjlist[i].size();
        // Every vertex that has an incoming
        // edge from i
```

```
        for (int j = 0; j < adjlist[i].size(); j++)
            iN[adjlist[i][j]]++;
    }
    cout << "Vertex\t\tIn\t\tOut" << endl;
    for (int k = 0; k < n; k++)
    {
        cout << k << "\t\t"
             << iN[k] << "\t\t"
             << ouT[k] << endl;
    }
}
// Driver code
int main()
{
    // Adjacency list representation of the graph
    vector<vector<int>> adjlist;
    // Vertices 1 and 2 have an incoming edge
    // from vertex 0
    vector<int> tmp;
    tmp.push_back(1);
    tmp.push_back(2);
    adjlist.push_back(tmp);
    tmp.clear();
    // Vertex 3 has an incoming edge
    // from vertex 1
    tmp.push_back(3);
    adjlist.push_back(tmp);
    tmp.clear();
    // Vertices 0, 5 and 6 have an incoming
    // edge from vertex 2
    tmp.push_back(0);
    tmp.push_back(5);
    tmp.push_back(6);
    adjlist.push_back(tmp);
    tmp.clear();
    // Vertices 1 and 4 have an incoming
    // edge from vertex 3
    tmp.push_back(1);
    tmp.push_back(4);
    adjlist.push_back(tmp);
    tmp.clear();
    // Vertices 2 and 3 have an incoming
    // edge from vertex 4
    tmp.push_back(2);
    tmp.push_back(3);
    adjlist.push_back(tmp);
    tmp.clear();
    // Vertices 4 and 6 have an incoming
```

```
    // edge from vertex 5
    tmp.push_back(4);
    tmp.push_back(6);
    adjlist.push_back(tmp);
    tmp.clear();
    // Vertex 5 has an incoming
    // edge from vertex 6
    tmp.push_back(5);
    adjlist.push_back(tmp);
    tmp.clear();
    int n = adjlist.size();
    findInOutDegree(adjlist, n);
}
```

## OUTPUT


# Practical 18

## Objective

Given a graph G, Write a Program to find the number of paths of length n between the source and destination entered by the user.

## CODE

```
#include <iostream>

using namespace std;
int countPaths(int graph[][100], int n, int src, int dest, int len)
{
    int count[n][n][len + 1];
    for (int e = 0; e <= len; e++)
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
```

```
            {
                count[i][j][e] = 0;
                if (e == 0 && i == j)
                    count[i][j][e] = 1;
                if (e == 1 && graph[i][j])
                    count[i][j][e] = 1;
                if (e > 1)
                    for (int a = 0; a < n; a++)
                        if (graph[i][a])
                            count[i][j][e] += count[a][j][e - 1];
            }
        }
    }
    return count[src][dest][len];
}
int main()
{
    int v;
    cout << "\nEnter the nunber of vertices: ";
    cin >> v;
    int matrix[100][100];
    cout << "Enter the adjacency matrix:\n";
    for (int i = 0; i < v; i++)
        for (int j = 0; j < v; j++)
            cin >> matrix[i][j];
    int src, dest;
    cout << "Enter the source node: ";
    cin >> src;
    cout << "Enter the destinastion node: ";
    cin >> dest;
    int len;
    cout << "Enter the path length: ";
    cin >> len;
    cout << "Total paths from node " << src
         << " to node " << dest << " having "
         << len << " edges: "
         << countPaths(matrix, v, src - 1, dest - 1, len);
    return 0;
}
```

## OUTPUT


# Practical 19

## Objective

Given an adjacency matrix of a graph, write a program to check whether a given set of vertices {v1,v2,v3.....,vk} forms an Euler path / Euler Circuit (for circuit assume vk=v1).

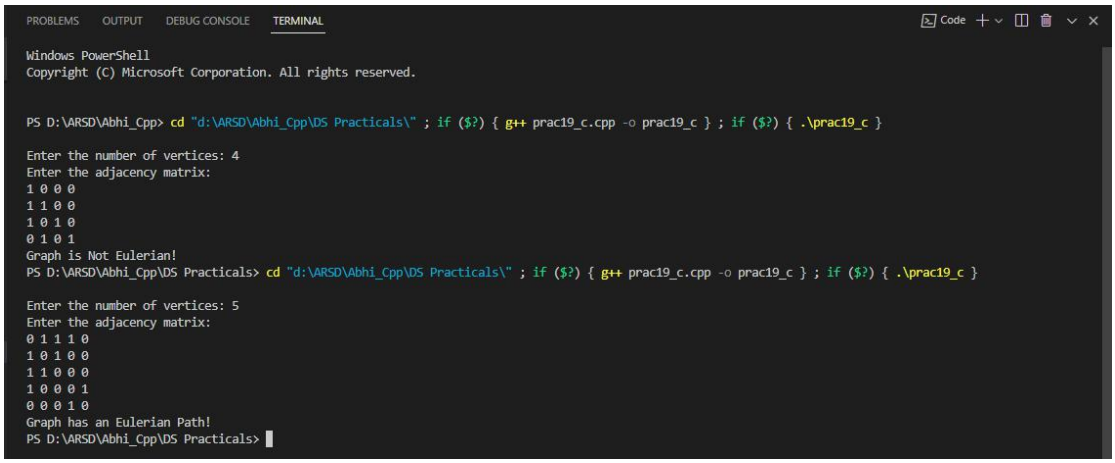
## CODE

```cpp
#include <iostream>

using namespace std;
int main()
{
    int n;
    cout << "\nEnter the number of vertices: ";
    cin >> n;
    int matrix[n][n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            matrix[i][j] = 0;
    cout << "Enter the adjacency matrix:\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> matrix[i][j];
    int degree, order = 0;
    for (int i = 0; i < n; i++)
    {
        degree = 0;
        for (int j = 0; j < n; j++)
            degree += matrix[i][j];
        if (degree % 2 != 0)
            order++;
```

```
    }
    if (order == 0)
        cout << "Graph has an Eulerian Circuit!" << endl;
    else if (order == 2)
        cout << "Graph has an Eulerian Path!" << endl;
    else
        cout << "Graph is Not Eulerian!" << endl;
    return 0;
}
```

## OUTPUT




# Practical 20

## Objective

Given a full m-ary tree with i internal vertices, Write a Program to find the number of leaf nodes.

## CODE

```
#include <iostream>

using namespace std;
int calcNodes(int m, int I)
{
    int result = 0;
    result = I * (m - 1) + 1;
    return result;
}
int main()
{
```

```
    int m, I, N;
    cout << "\nEnter the maximum no. of children in full m-ary tree: ";
    cin >> m;
    cout << "Enter the number of internal vertices: ";
    cin >> I;
    N = calcNodes(m, I);
    cout << "Number of Leaf Nodes in the full m-ary tree: " << N;
    return 0;
}
```

## OUTPUT