**PRACTICAL-2**

**AIM:** To find the roots of given quadratic equations.

**EXPERIMENT:** Write a C-code to find the roots of the following equations:

a) x2+12x+4

b) 2x2+4x+2

c) 3x2+4x+7

**CODE:**

#include <stdio.h>

#include <math.h>

int main() {

    double a, b, c;

    double discriminant, root1, root2;

    // Input coefficients from the user

    printf("Enter coefficients (a, b, c) of the quadratic equation (ax^2 + bx + c 0):\n");

    printf("a: ");

    scanf("%lf", &a);

    printf("b: ");

    scanf("%lf", &b);

    printf("c: ");

    scanf("%lf", &c);

    // Calculate discriminant

    discriminant = b \* b - 4 \* a \* c;

    // Display the quadratic equation

    printf("\nQuadratic Equation: %.2fx^2 + %.2fx + %.2f = 0\n", a, b, c);

    // Display the discriminant

    printf("Discriminant: %.2f\n", discriminant);

    // Check the nature of roots

    if (discriminant > 0) {

        // Real and distinct roots

        root1 = (-b + sqrt(discriminant)) / (2 \* a);

        root2 = (-b - sqrt(discriminant)) / (2 \* a);

        printf("\nRoots are real and distinct:\n");

        printf("Root 1 = %.2f\n", root1);

        printf("Root 2 = %.2f\n", root2);

    } else if (discriminant == 0) {

        // Real and equal roots

        root1 = root2 = -b / (2 \* a);

        printf("\nRoots are real and equal:\n");

        printf("Root 1 = Root 2 = %.2f\n", root1);

    } else {

        // Complex roots

        double realPart = -b / (2 \* a);

        double imaginaryPart = sqrt(fabs(discriminant)) / (2 \* a);

        printf("\nRoots are complex:\n");

        printf("Root 1 = %.2f + %.2fi\n", realPart, imaginaryPart);

        printf("Root 2 = %.2f - %.2fi\n", realPart, imaginaryPart);

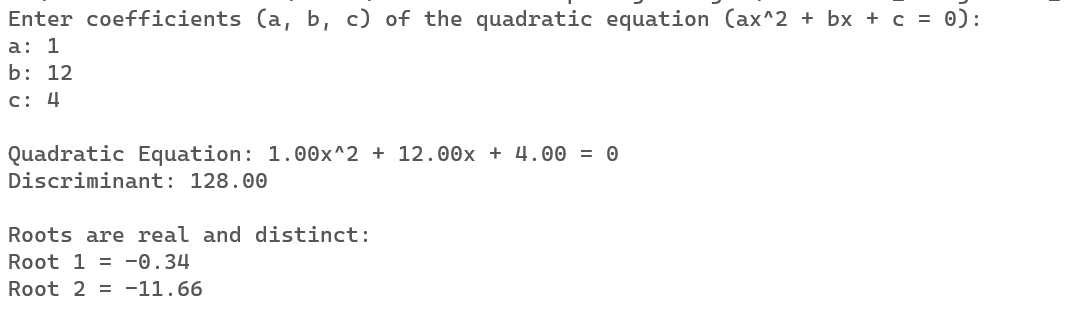
    }

    return 0;

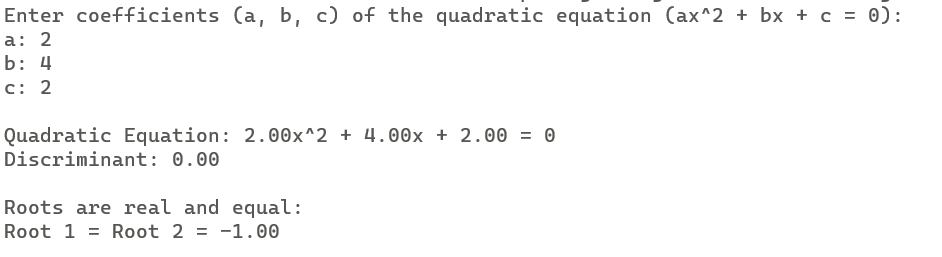
}

**Output :**

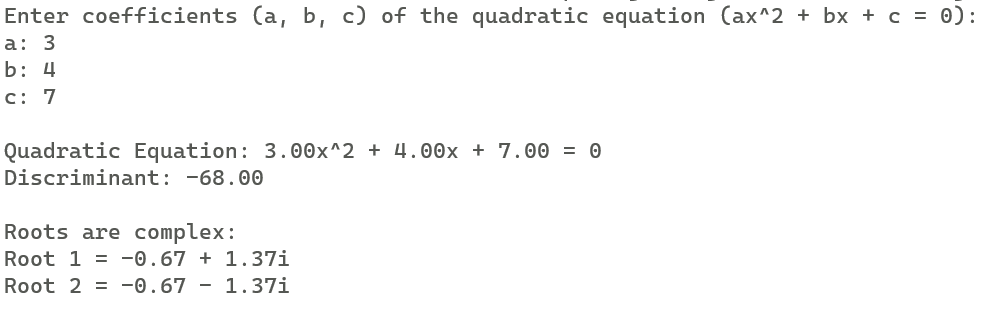
**Equation – a**



**Equation – b**



**Equation - c**



**Result :**

1. Equation a) (x2+12x+4) has 2 real roots:

a). x= -0.34

b). x= -11.66

2. Equation b) (2x2+4x+2) has a unique real root:

a). x= -1

3. Equation c) (3x2+4x+7) has complex roots:

a). x=-0.67-1.37i

b). x=-0.67+1.37i

Where i=

**Practical – 3**

**Aim :** To write a C-Code to plot a graph for the given function.

**Experiment :** Write a C-Code to plot a graph for the function y = .

**Code :**

#include <stdio.h>

#include <math.h>

// Function to calculate the value of y = exp(-x\*x/2)

double function(double x) {

    return exp(-x \* x / 2);

}

int main() {

    // Print y-axis label

    printf("  y\n");

    printf("  ^\n");

    // Iterate over y values and print graphical representation of the function

    for (double y = 1.0; y >= 0.0; y -= 0.1) {

        printf("%.1f |", y);

        // Iterate over x values and print \* or space based on the function value

        for (double x = -3.0; x <= 3.0; x += 0.2) {

            double functionValue = function(x);

            if (functionValue >= y) {

                printf("\*");

            } else {

                printf(" ");

            }

        }

        printf("\n");

    }

    // Print x-axis label

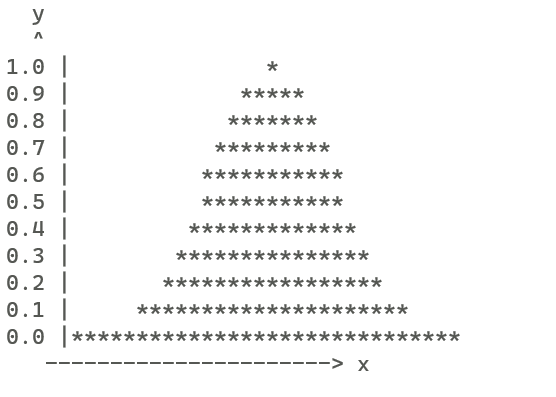
    printf("   ----------------------> x\n");

    return 0;

}

**Output :**

Graph - 2

****

**Result :**

* The graph for the given equation has been plotted and shown in Graph 2.
* The horizontal line in Graph-2 represents x-axis, whereas the vertical line represents y-axis.

**Practical – 5**

**Aim** : To prepare a frequency table.

**Experiment :** Write a C- code to prepare the following grouped frequency distribution :-

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C.I | 0-10 | 10-20 | 20-30 | 30-40 | 40-50 | 50-60 | 60-70 |
| Frequency | 4 | 8 | 11 | 15 | 12 | 6 | 3 |

**Code** :

#include<stdio.h>

// Function to display the grouped frequency table

void displayGroupedFrequencyTable(float lower[], float upper[], int frequencies[], int size)

{

    printf("\nGrouped Frequency Table:\n");

    printf("+----------------+-----------+\n");

    printf("| Class          | Frequency |\n");

    printf("+----------------+-----------+\n");

    for (int i = 0; i < size; i++) {

        printf("| %.2f - %.2f  | %d        |\n", lower[i], upper[i], frequencies[i]);

        printf("+----------------+-----------+\n");

    }

}

//main function

int main() {

    int size;

    // Input the size of the grouped frequency distribution from the user

    printf("Enter the size of the grouped frequency distribution: ");

    scanf("%d", &size);

    float lower[size], upper[size]; // Range boundaries for each class

    float midpoints[size];

    int frequencies[size];

    // Input lower, upper boundaries, and frequencies from the user

    printf("Enter lower and upper boundaries, and frequencies for each class:\n");

    for (int i = 0; i < size; i++) {

        printf("Lower bound for class %d: ", i + 1);

        scanf("%f", &lower[i]);

        printf("Upper bound for class %d: ", i + 1);

        scanf("%f", &upper[i]);

        printf("Frequency for class %d: ", i + 1);

        scanf("%d", &frequencies[i]);

    }

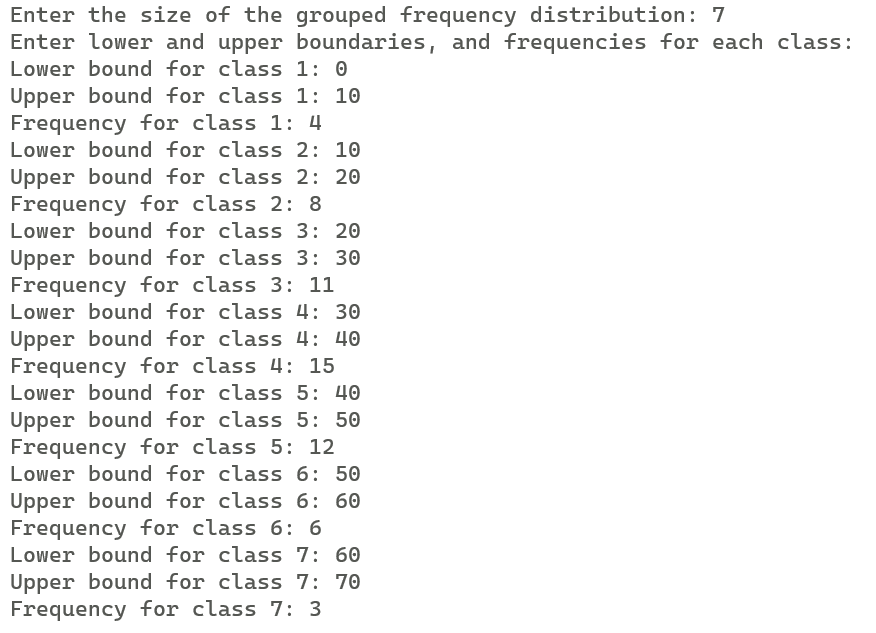
// Display the completed grouped frequency table

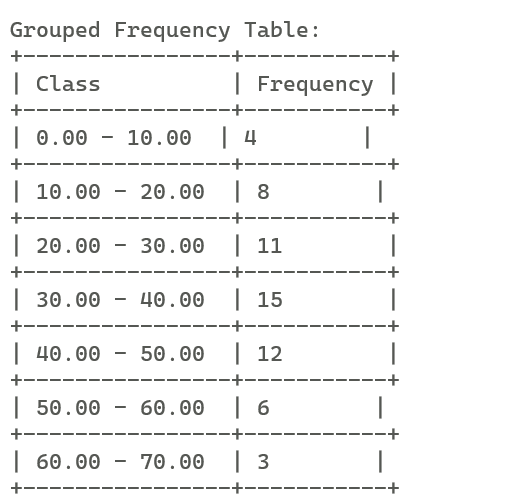
    displayGroupedFrequencyTable(lower, upper, frequencies, size);

    return 0;

}

**Output :**





**Result** : Frequency table is shown in Output.

**PRACTICAL – 6**

**AIM:** To find mean, mode and median of the given frequency distribution.

**EXPERIMENT:**  Write a C- code to find the mean, median and mode of the following grouped frequency distribution :-

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C.I | 0-10 | 10-20 | 20-30 | 30-40 | 40-50 | 50-60 | 60-70 |
| Frequency | 4 | 8 | 11 | 15 | 12 | 6 | 3 |

**CODE:**

#include <stdio.h>

// Function to calculate the midpoint of a class given its range

float calculateMidpoint(float lower, float upper) {

    return (lower + upper) / 2.0;

}

// Function to calculate the mean of a grouped frequency distribution

double calculateMean(float midpoints[], int frequencies[], int size) {

    double sum = 0.0;

    int totalFrequency = 0;

    for (int i = 0; i < size; i++) {

        sum += midpoints[i] \* frequencies[i];

        totalFrequency += frequencies[i];

    }

    return sum / totalFrequency;

}

// Function to calculate the median of a grouped frequency distribution

double calculateMedian(float midpoints[], int frequencies[], int size) {

    int totalFrequency = 0;

    for (int i = 0; i < size; i++) {

        totalFrequency += frequencies[i];

    }

    int cumulativeFrequency = 0;

    double median;

    for (int i = 0; i < size; i++) {

        cumulativeFrequency += frequencies[i];

        if (cumulativeFrequency >= totalFrequency / 2) {

            // If cumulative frequency is greater than or equal to half of total frequency

            // Median is the midpoint of the current class

            median = midpoints[i];

            break;

        }

    }

    return median;

}

// Function to calculate the mode of a grouped frequency distribution

double calculateMode(float midpoints[], int frequencies[], int size) {

    int maxFrequency = frequencies[0];

    double mode = midpoints[0];

    for (int i = 1; i < size; i++) {

        if (frequencies[i] > maxFrequency) {

            maxFrequency = frequencies[i];

            mode = midpoints[i];

        }

    }

    return mode;

}

// Function to display the grouped frequency table

void displayGroupedFrequencyTable(float lower[], float upper[], float midpoints[], int frequencies[], int size) {

    printf("\nGrouped Frequency Table:\n");

    printf("+----------------+-----------+\n");

    printf("| Class          | Frequency |\n");

    printf("+----------------+-----------+\n");

    for (int i = 0; i < size; i++) {

        printf("| %.2f - %.2f  | %d        |\n", lower[i], upper[i], frequencies[i]);

        printf("+----------------+-----------+\n");

    }

}

int main() {

    int size;

    // Input the size of the grouped frequency distribution from the user

    printf("Enter the size of the grouped frequency distribution: ");

    scanf("%d", &size);

    float lower[size], upper[size]; // Range boundaries for each class

    float midpoints[size];

    int frequencies[size];

    // Input lower, upper boundaries, and frequencies from the user

    printf("Enter lower and upper boundaries, and frequencies for each class:\n");

    for (int i = 0; i < size; i++) {

        printf("Lower bound for class %d: ", i + 1);

        scanf("%f", &lower[i]);

        printf("Upper bound for class %d: ", i + 1);

        scanf("%f", &upper[i]);

        midpoints[i] = calculateMidpoint(lower[i], upper[i]);

        printf("Frequency for class %d: ", i + 1);

        scanf("%d", &frequencies[i]);

    }

    // Display the completed grouped frequency table

    displayGroupedFrequencyTable(lower, upper, midpoints, frequencies, size);

    // Calculate and display the mean, median, and mode

    double mean = calculateMean(midpoints, frequencies, size);

    double median = calculateMedian(midpoints, frequencies, size);

    double mode = calculateMode(midpoints, frequencies, size);

    printf("\nMean: %.2lf\n", mean);

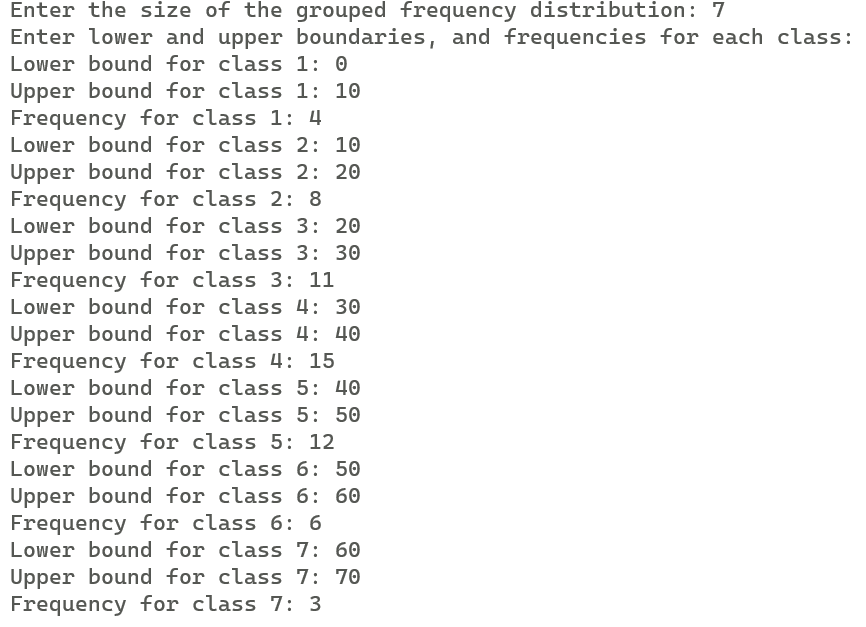
    printf("Median: %.2lf\n", median);

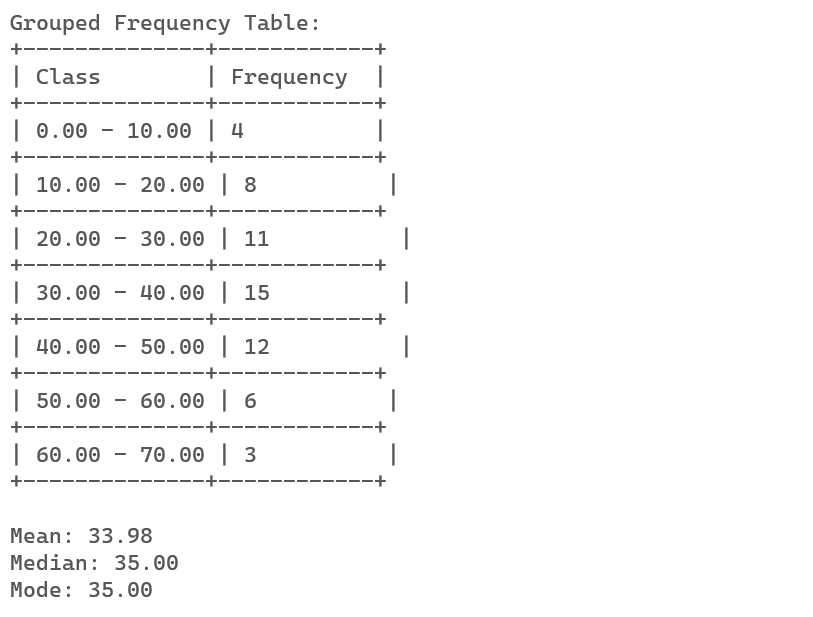
    printf("Mode: %.2lf\n", mode);

    return 0;

}

**Output :**

****

****

**Result :**

1. Mean of the data = 33.98 .
2. Mode of the data = 35.00 .
3. Median of the data = 35.00 .

**PRACTICAL – 10**

**AIM:** To read matrices and check their compatibility for multiplication. If compatible, print product and transpose.

**EXPERIMENT:** Write an C program to check the matrices for compatibility of multiplication and performs the following calculations :

1. Print C = A\*B and Transpose of C.

Where ,

A = and B =

**CODE:**

#include <stdio.h>

// Function prototypes

void inputMatrix(int rows, int cols, int matrix[rows][cols]);

void multiplyMatrices(int rows1, int cols1, int mat1[rows1][cols1], int rows2, int cols2, int mat2[rows2][cols2], int result[rows1][cols2]);

void printMatrix(int rows, int cols, int matrix[rows][cols]);

void transposeMatrix(int rows, int cols, int matrix[rows][cols], int result[cols][rows]);

int main() {

    int r1, c1, r2, c2;

    // Input sizes of Matrix A and Matrix B

    printf("Enter the size of Matrix A (rows columns): ");

    scanf("%d %d", &r1, &c1);

    printf("Enter the size of Matrix B (rows columns): ");

    scanf("%d %d", &r2, &c2);

    // Check if matrices can be multiplied

    if (c1 != r2) {

        printf("Matrices cannot be multiplied. Exiting...\n");

        return 1;

    }

    int mat1[r1][c1], mat2[r2][c2], result[r1][c2], transposeResult[c2][r1];

    // Input elements of Matrix A and Matrix B

    inputMatrix(r1, c1, mat1);

    inputMatrix(r2, c2, mat2);

    // Multiply matrices and store the result in Matrix C

    multiplyMatrices(r1, c1, mat1, r2, c2, mat2, result);

    // Print Matrix C

    printf("\nMATRIX C:\n");

    printMatrix(r1, c2, result);

    // Transpose Matrix C and store the result in transposeResult

    transposeMatrix(r1, c2, result, transposeResult);

    // Print transpose of Matrix C

    printf("\nTRANSPOSE(C):\n");

    printMatrix(c2, r1, transposeResult);

    return 0;

}

// Function to input elements of a matrix

void inputMatrix(int rows, int cols, int matrix[rows][cols]) {

    printf("Enter elements of the matrix:\n");

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

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

        }

    }

}

// Function to multiply two matrices

void multiplyMatrices(int rows1, int cols1, int mat1[rows1][cols1], int rows2, int cols2, int mat2[rows2][cols2], int result[rows1][cols2]) {

    for (int i = 0; i < rows1; i++) {

        for (int j = 0; j < cols2; j++) {

            result[i][j] = 0;

            for (int k = 0; k < cols1; k++) {

                result[i][j] += mat1[i][k] \* mat2[k][j];

            }

        }

    }

}

// Function to transpose a matrix

void transposeMatrix(int rows, int cols, int matrix[rows][cols], int result[cols][rows]) {

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            result[j][i] = matrix[i][j];

        }

    }

}

// Function to print elements of a matrix

void printMatrix(int rows, int cols, int matrix[rows][cols]) {

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            printf("%d\t", matrix[i][j]);

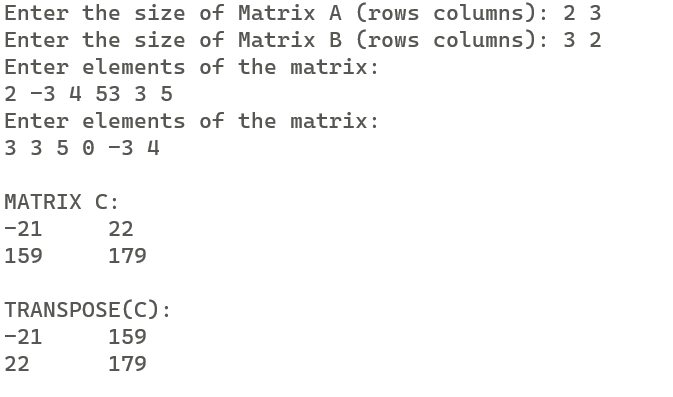
        }

        printf("\n");

    }

}

**Output :**



**Result :**

1. C = A\*B =
2. Transpose of C =

**PRACTICAL – 13**

**AIM :** To read the parameters from the user and generate random numbers following normal distribution using CLT. Calculate sample mean, sample variance and compare with population mean and variance.

**EXPERIMENT :** Write an C-code to generate the following random numbers using CLT

1. 10 random numbers following N(0,1)
2. 10 random numbers following N(10,5)

calculate the sample mean and variance and compare with the statistics based on the respective population parameters.

**CODE :**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

int main() {

    // Declare variables

    int i, j, n;

    float norm[150], m, var, m1, var1, m2, var2, sum = 0, sumsq = 0;

    // Seed the random number generator

    srand(time(0));

    // User input: Number of random numbers to generate

    printf("Enter the number of random numbers to be generated: ");

    scanf("%d", &n);

    // User input: Mean and Variance

    printf("\nEnter the mean (1st parameter): ");

    scanf("%f", &m);

    printf("Enter the variance (2nd parameter): ");

    scanf("%f", &var);

    printf("%d Random Numbers generated from N(%0.2f, %0.2f) are as follows:\n", n, m, var);

    // Generate random numbers

    for (i = 0; i < n; i++) {

        norm[i] = 0;

        for (j = 0; j < n; j++) {

            norm[i] += (float)rand() / RAND\_MAX;

        }

        norm[i] = (norm[i] - (n / 2.0)) / sqrt(n / 12.0);

        norm[i] = norm[i] \* sqrt(var) + m;

        if (i == n - 1) {

            printf("%f", norm[i]);

        } else {

            printf("%f, ", norm[i]);

        }

        sum += norm[i];

        sumsq += pow(norm[i], 2);

    }

    m1 = sum / n;

    var1 = sumsq / n - pow(m1, 2);

    m2 = m;

    var2 = var;

    // Display results based on sample data

    printf("\n\nBASED ON SAMPLE DATA:");

    printf("\n1. Mean=%f", m1);

    printf("\n2. Variance=%f", var1);

    // Display results based on population parameters

    printf("\n\nBASED ON POPULATION PARAMETERS:");

    printf("\n1. Mean=%f", m2);

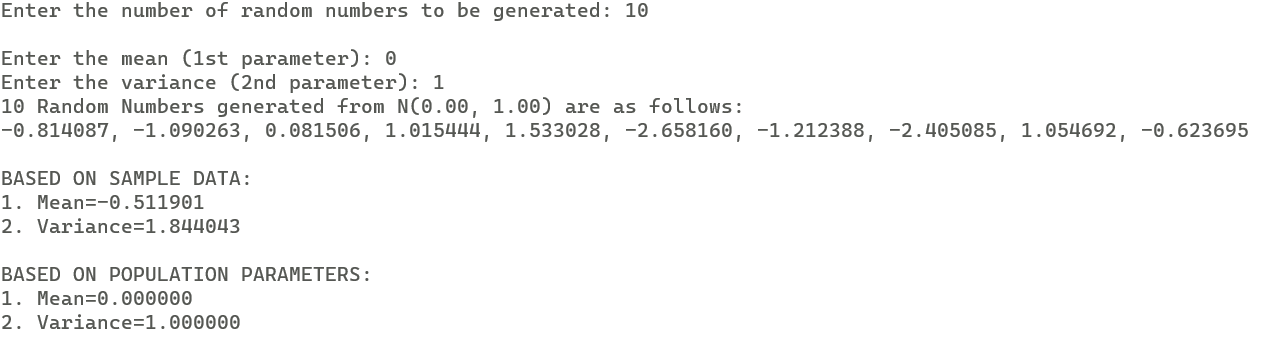
    printf("\n2. Variance=%f\n", var2);

    return 0;

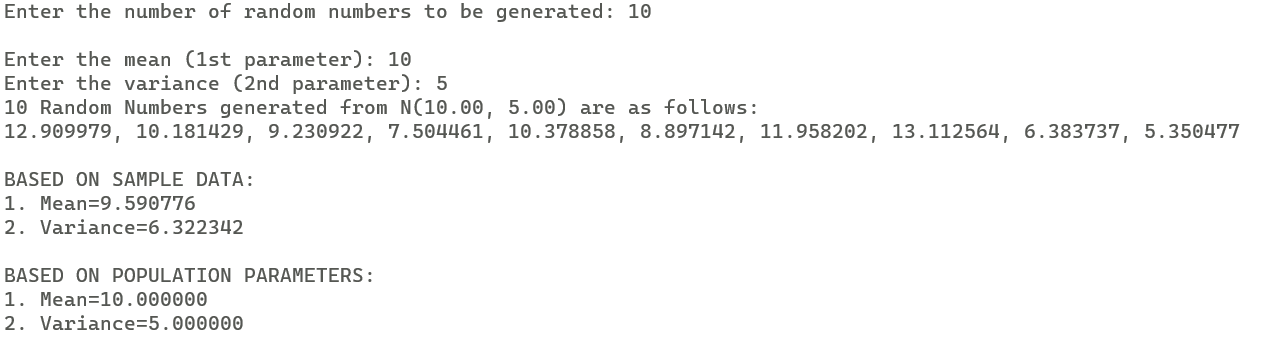
}

**Output :**

Case a)



Case b)



**Result :**

Case a) :

1. BASED ON SAMPLE DATA :

Mean = -0.511901

Median = 1.844043

1. BASED ON POPULATION PARAMETERS :

Mean = 0

Variance = 1

Case b) :

1. BASED ON SAMPLE DATA :

Mean = 9.590776

Median = 6.322342

1. BASED ON POPULATION PARAMETERS :

Mean = 10

Variance = 5

**PRACTICAL – 15**

**AIM:** To fit binomial distribution in the given data, and check for the goodness of fit using chi-square test.

**EXPERIMENT:** Write a function program to calculate the binomial coefficientand call it to fit a binomial distribution. Further, also write acode to perform a chi-square goodness fit test, to conclude, whether the following frequency distribution of rotten orangesfollows binomial distribution.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rotten Oranges | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Frequency | 334 | 369 | 191 | 63 | 22 | 12 | 9 | 0 | 0 | 0 |

**CODE:**

#include <stdio.h>

#include <math.h>

// Function to draw a border with a specified character

void drawBorder(int n, char a) {

    for (int i = 0; i < n; i++) {

        printf("%c", a);

    }

    printf("\n");

}

// Function to calculate factorial

int factorial(int num) {

    if (num <= 1) {

        return 1;

    } else {

        return num \* factorial(num - 1);

    }

}

// Function to calculate binomial coefficient

int binomialCoefficient(int n, int r) {

    int nf = factorial(n);

    int rf = factorial(r);

    int nrf = factorial(n - r);

    return nf / (rf \* nrf);

}

int main() {

    int n, a, count = 0, sum = 0, sumfreq = 0;

    float mean, p, q, chi, cal\_chi = 0.0;

    printf("No. of Entries: ");

    scanf("%d", &n);

    a = n - 1;

    // Declare arrays

    int x[n], f[n], pf[n], ef1[n], pef1[n];

    float pmf[n], ef[n], pef[n];

    printf("\nEnter the data (Rotten Oranges and Frequencies):\n");

    // Input data array

    for (int i = 0; i < n; i++) {

        scanf("%d %d", &x[i], &f[i]);

        sumfreq += f[i];

        sum += (f[i] \* x[i]);

    }

    printf("\nData Entered:\n");

    drawBorder(33, '-');

    printf("|Rotten Oranges\t|Frequency\t|\n");

    drawBorder(33, '-');

    for (int i = 0; i < n; i++) {

        printf("|%d\t\t|%d\t\t|\n", x[i], f[i]);

    }

    drawBorder(33, '-');

    mean = 1.0 \* sum / sumfreq;

    p = mean / a;

    q = 1 - p;

    // Calculate probabilities and expected frequencies

    for (int i = 0; i < n; i++) {

        pmf[i] = binomialCoefficient(a, x[i]) \* pow(p, i) \* pow(q, a - i);

        ef[i] = sumfreq \* pmf[i];

        ef1[i] = ef[i] + 0.5;

        pef[i] = ef[i];

        pf[i] = f[i];

    }

    printf("\nFitted Binomial Distribution:\n");

    drawBorder(64, '-');

    printf("|Rotten Oranges\t|Frequency\t|pmf\t\t|Expected Freq\t|\n");

    drawBorder(64, '-');

    for (int i = 0; i < n; i++) {

        printf("|%d\t\t|%d\t\t|%0.4f\t\t|%d\t\t|\n", x[i], f[i], pmf[i], ef1[i]);

    }

    drawBorder(64, '-');

    // Pooling step

    for (int i = a; i >= 0; i--) {

        if (pef[i] <= 5 || pf[i] <= 5) {

            pef[i - 1] += pef[i];

            pf[i - 1] += pf[i];

            pf[i] = 0;

            pef[i] = 0;

            count++;

        }

    }

    // Calculate chi-square statistic

    for (int i = 0; i < n; i++) {

        if (pef[i] != 0) {

            cal\_chi += (pow((pf[i] - pef[i]), 2)) / pef[i];

        }

        pef1[i] = pef[i] + 0.5;

    }

    printf("\nChi-Sq. Goodness of Fit Test:\n\t\t");

    drawBorder(64, '-');

    printf("\t\t|Before Pooling\t\t\t|After Pooling\t\t\t|\n");

    drawBorder(80, '-');

    printf("|Rotten Oranges\t|Frequency\t|Expected Freq\t|Frequency\t|Expected Freq\t|\n");

    drawBorder(80, '-');

    for (int i = 0; i < n; i++) {

        printf("|%d\t\t|%d\t\t|%d\t\t|%d\t\t|%d\t\t|\n", x[i], f[i], ef1[i], pf[i], pef1[i]);

    }

    drawBorder(80, '-');

    printf("\nDegrees of Freedom = %d", n - (1 + 1 + count));

    printf("\nCalculated Value of Chi-sq is = %0.3f\n", cal\_chi);

    printf("\nTabulated Value of Chi-sq for %d d.f. = ", n - (1 + 1 + count));

    scanf("%f", &chi);

    printf("\nTabulated Value of Chi-sq is = %0.3f", chi);

    printf("\n\nHypothesis:\nNull Hypothesis(H0): The data follows binomial distribution.\nAlternate Hypothesis(H1): The data does not follow binomial distribution.");

    if (cal\_chi < chi) {

        printf("\n\nAs the calculated value of chi-sq is less than the tabulated value, this means that we do not have enough evidence to reject the null hypothesis. Hence we accept the null hypothesis at the 5 percent level of significance. This implies that it is a good fit!");

    } else {

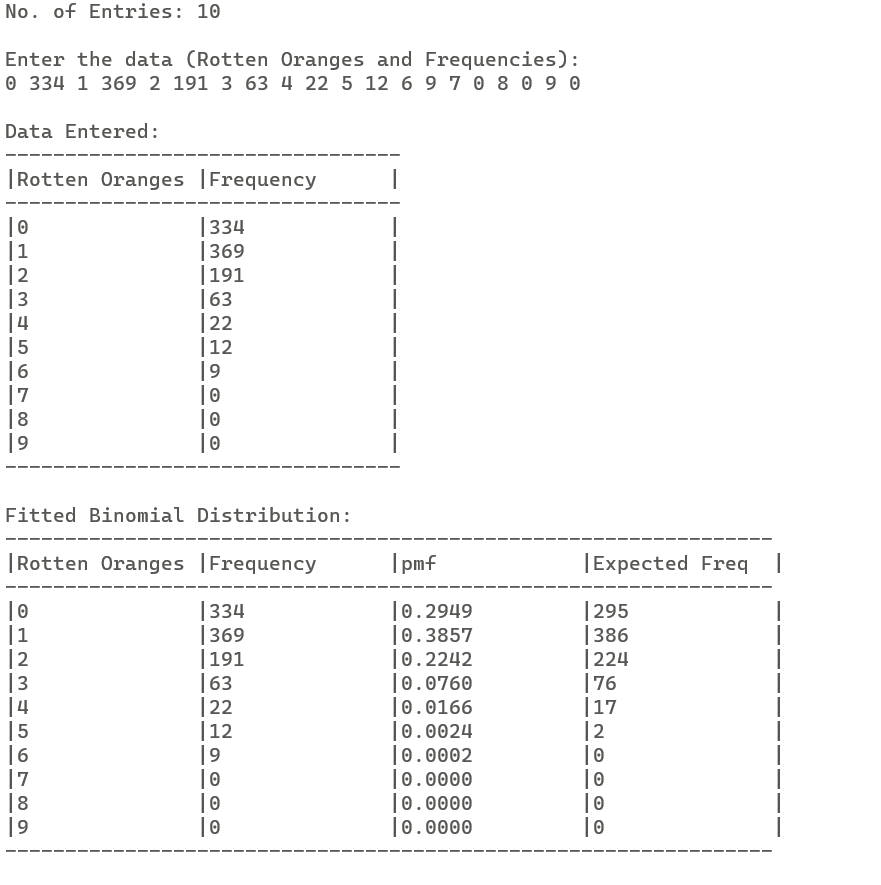
        printf("\n\nAs the calculated value of chi-sq is greater than the tabulated value, we reject our null hypothesis at the 5 percent level of significance. This implies that it is not a good fit!");

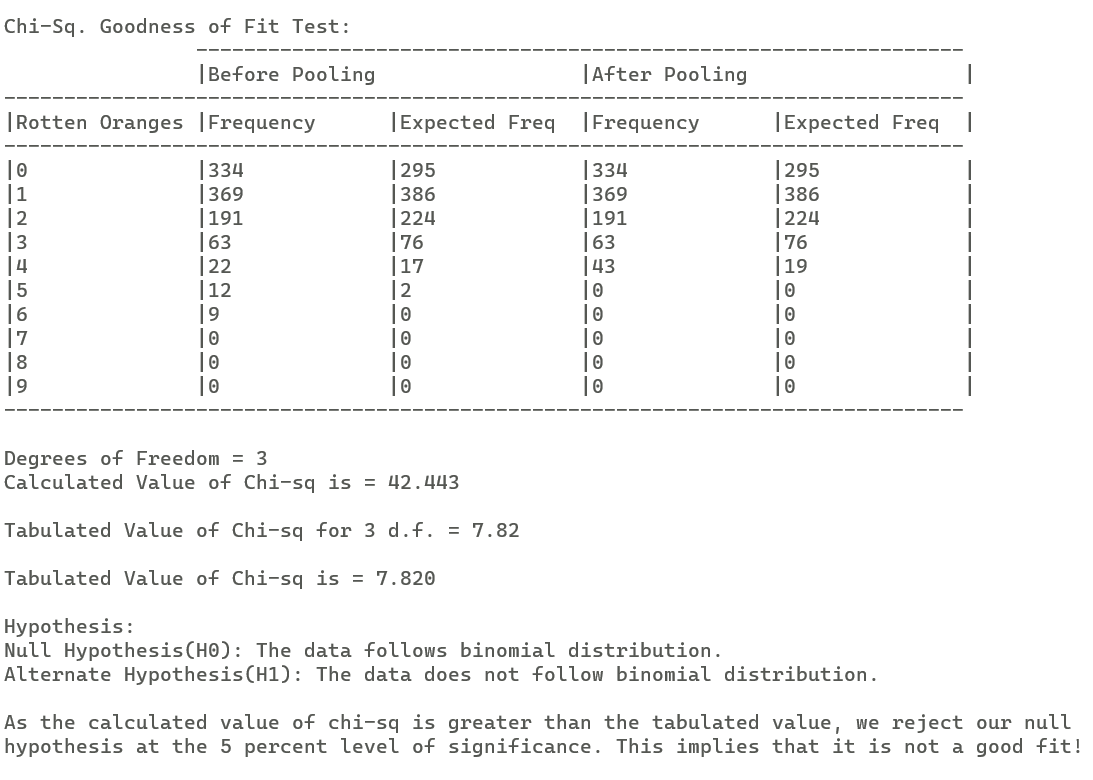
    }

    return 0;

}

**Output :**





**Result :**

* Calculated Value of Chi-sq is = 42.443
* Tabulated Value of Chi-sq is = 7.820
* As the calculated value of chi-sq is greater than the tabulated value, we reject our

null hypothesis at 5 per-cent level of significance. This implies that it is not a good fit!

This implies that the frequency distribution of rotten oranges does not follow binomial distribution. **AIM :** To calculate correlation coefficient for the given data and thus calculate multiple and partial correlation coefficients.

**EXPERIMENT:** Write an C program to calculate the correlation coefficient . Use this to compute the multiple correlation and Partial correlation coefficient for the following data:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Marks in Maths(X) | 35 | 37 | 38 | 42 | 44 | 46 | 51 | 54 | 55 | 56 |
| Marks in Stats(Y) | 40 | 32 | 39 | 42 | 41 | 31 | 50 | 52 | 46 | 55 |
| Marks in Comp.(Z) | 35 | 90 | 70 | 40 | 95 | 40 | 66 | 80 | 80 | 50 |

**CODE:**

#include <stdio.h>

#include <math.h>

// Function to read input from the user for an array

void read(float x[], int n);

// Function to calculate the correlation coefficient between two arrays

float corr(float x[], float y[], int n);

// Function to calculate the multiple correlation coefficient

float multiple(float a, float b, float c);

// Function to calculate the partial correlation coefficient

float partial(float a, float b, float c);

// Function to print a border

void border(int n, char a);

int main() {

    int n, i;

    float x[20], y[20], z[20], rxy, ryz, rxz, mult, part;

    printf("Enter the size (no. of entries): ");

    scanf("%d", &n);

    // Read data from the user for each array

    printf("Enter data for X:\n");

    read(x, n);

    printf("Enter data for Y:\n");

    read(y, n);

    printf("Enter data for Z:\n");

    read(z, n);

    // Print entered data

    printf("\nEntered data is:\n");

    border(25, '-');

    printf("|X\t|Y\t|Z\t|\n");

    border(25, '-');

    for (i = 0; i < n; i++) {

        printf("|%0.2f\t|%0.2f\t|%0.2f\t|\n", x[i], y[i], z[i]);

    }

    border(25, '-');

    // Calculate correlation coefficients

    rxy = corr(x, y, n);

    ryz = corr(y, z, n);

    rxz = corr(x, z, n);

    // Print correlation coefficients

    printf("\nCorrelation Coefficients:\n");

    printf("\t1) r(X,Y)=r(Y,X)=%f\n", rxy);

    printf("\t2) r(Y,Z)=r(Z,Y)=%f\n", ryz);

    printf("\t3) r(X,Z)=r(Z,X)=%f\n", rxz);

    // Calculate and print multiple correlation coefficient

    mult = multiple(rxy, ryz, rxz);

    printf("\nMultiple Correlation Coefficient:\n");

    printf("\t1) R2y.xz=%f\n", mult);

    // Calculate and print partial correlation coefficient

    part = partial(rxy, rxz, ryz);

    printf("\nPartial Correlation Coefficient:\n");

    printf("\t1) rxy.z=%f\n", part);

    printf("\nRESULTS HAVE BEEN SUCCESSFULLY PRINTED!");

    return 0;

}

// Function to read input from the user for an array

void read(float x[], int n) {

    printf("Enter %d elements separated by spaces: ", n);

    for (int i = 0; i < n; i++) {

        scanf("%f", &x[i]);

    }

}

// Function to calculate the correlation coefficient between two arrays

float corr(float x[], float y[], int n) {

    float r, covar\_xy = 0.0, covar\_xx = 0.0, covar\_yy = 0.0;

    for (int i = 0; i < n; i++) {

        covar\_xy += (x[i] \* y[i]);

        covar\_xx += (x[i] \* x[i]);

        covar\_yy += (y[i] \* y[i]);

    }

    r = covar\_xy / (sqrt(covar\_xx) \* sqrt(covar\_yy));

    return r;

}

// Function to calculate the multiple correlation coefficient

float multiple(float a, float b, float c) {

    return (pow(a, 2) + pow(b, 2) - (2 \* a \* b \* c)) / (1.0 - pow(c, 2));

}

// Function to calculate the partial correlation coefficient

float partial(float a, float b, float c) {

    return (a - (b \* c)) / sqrt((1.0 - pow(b, 2)) \* (1.0 - pow(c, 2)));

}

// Function to print a border

void border(int n, char a) {

    for (int i = 0; i < n; i++) {

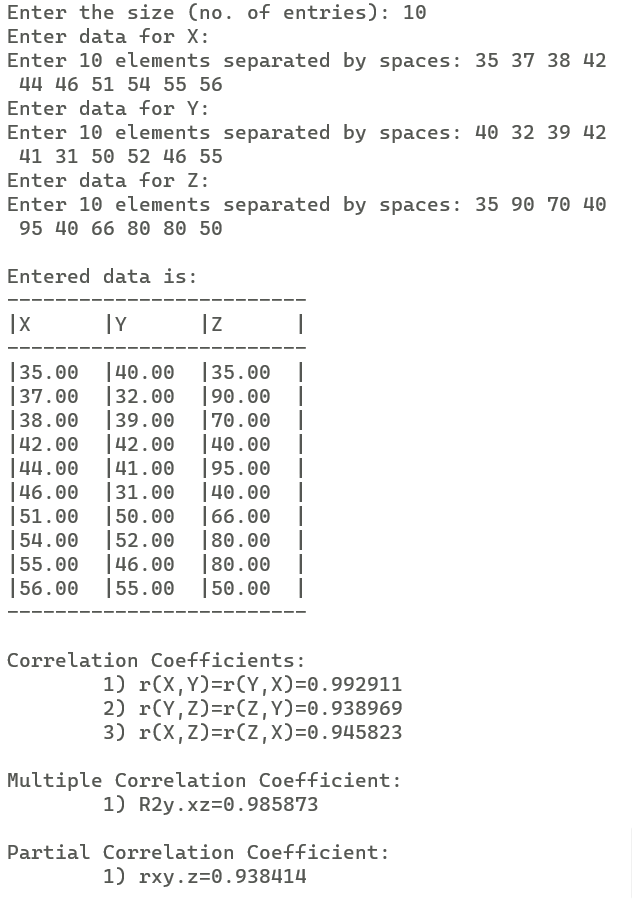
        printf("%c", a);

    }

    printf("\n");

}

**Output :**

****

**Result :**

1. On calculation

Multiple correlation R2y.xz= 0.985873

Partial corelation coefficient rxy.z= 0.938414