Question 1: *// Write a program to check whether a given matrix is orthogonal.*

*// Exam Roll No. - 20220PHY014*

#include <stdio.h>

#define MAX 100 *// maximum order of matrix can be 100*

void print\_matrix(float *matrix*[MAX][MAX], int *order*);

int isOrthogonal(float *matrix*[MAX][MAX], int *n*);

int main()

{

    float matrix[MAX][MAX] = {{1, 0, 0},

                              {0, 1, 0},

                              {0, 0, 1}};

    int matrix\_order = 3;

    printf("\nYour matrix is:\n");

    print\_matrix(matrix, matrix\_order);

    if (isOrthogonal(matrix, matrix\_order))

        puts("\nGiven matrix is orthogonal");

    else

        puts("\nGiven matrix is not orthogonal");

    return 0;

}

void print\_matrix(float *matrix*[MAX][MAX], int *order*)

{

    for (int i = 0; i < *order*; i++)

    {

        for (int j = 0; j < *order*; j++)

            printf("*%10f* ", *matrix*[i][j]);

        printf("\n");

    }

}

int isOrthogonal(float *matrix*[MAX][MAX], int *n*)

{

*// Find transpose*

    float transpose[MAX][MAX];

    for (int i = 0; i < *n*; i++)

    {

        for (int j = 0; j < *n*; j++)

            transpose[i][j] = *matrix*[j][i];

    }

    printf("\nTanspose of matrix is:\n");

    print\_matrix(transpose, *n*);

*// Find product of matrix and its transpose*

    float product[MAX][MAX];

    for (int i = 0; i < *n*; i++)

    {

        for (int j = 0; j < *n*; j++)

        {

            float sum = 0;

            for (int k = 0; k < *n*; k++)

                sum += (*matrix*[i][k] \* *matrix*[j][k]);

            product[i][j] = sum;

        }

    }

    printf("\nProduct of matrix and its tranpose is:\n");

    print\_matrix(product, *n*);

*// Check if the product is identity matrix*

    for (int i = 0; i < *n*; i++)

    {

        for (int j = 0; j < *n*; j++)

        {

            if (i != j && product[i][j] != 0)

                return 0;

            if (i == j && product[i][j] != 1)

                return 0;

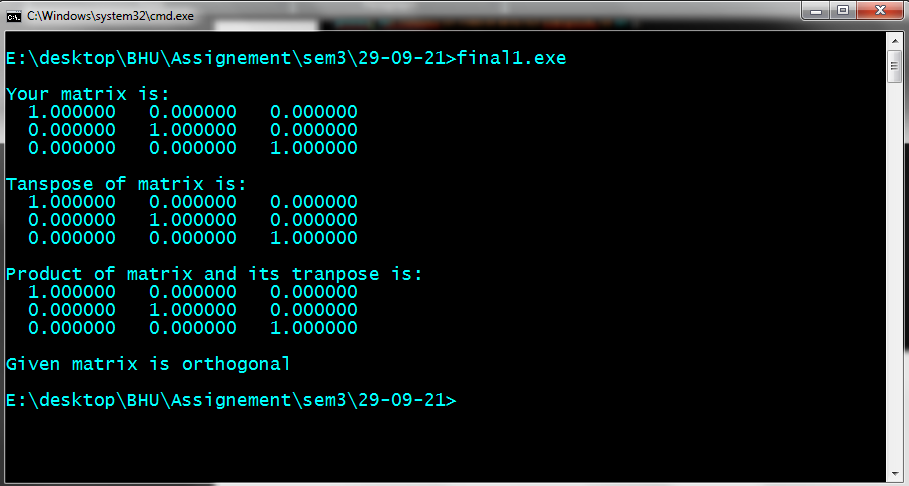
        }

    }

    return 1;

}

Output:



Question 2: *// Write a program to check whether the system of linear equations are consistency.*

*// Exam Roll No. - 20220PHY014*

#include <stdio.h>

#define M 10

void swap(float *a*[][M], int *j*, int *k*, int *c*);

int PerformOperation(float *a*[][M], int *n*);

int CheckConsistencY(float *a*[][M], int *n*, int *flag*);

void PrintResult(float *a*[][M], int *n*, int *flag*);

int main()

{

    float matrix[M][M] = {{0, 2, 1, 4},  *// 2y + z = 4*

                          {1, 1, 2, 6},  *// x + y + 2z = 6*

                          {2, 1, 1, 7}}; *// 2x + y + z = 7*

    int order = 3, flag = 0;

*// Performing Matrix transformation*

    flag = PerformOperation(matrix, order);

    if (flag == 1)

        flag = CheckConsistencY(matrix, order, flag);

    PrintResult(matrix, order, flag);

    return 0;

}

void swap(float *a*[][M], int *j*, int *k*, int *c*) *// swap values of a[j][k] and a[j + c][k]*

{

    float temp = *a*[*j*][*k*];

*a*[*j*][*k*] = *a*[*j* + *c*][*k*];

*a*[*j* + *c*][*k*] = temp;

}

int PerformOperation(float *a*[][M], int *n*) *// function to reduce matrix to reduced row echelon form*

{

    int i, j, k = 0, c, flag = 0, m = 0;

    float pro = 0;

    for (i = 0; i < *n*; i++) *// Performing elementary opertions*

    {

        if (*a*[i][i] == 0)

        {

            c = 1;

            while ((i + c) < *n* && *a*[i + c][i] == 0)

                c++;

            if ((i + c) == *n*)

            {

                flag = 1;

                break;

            }

            for (j = i, k = 0; k <= *n*; k++)

                swap(*a*, j, k, c);

        }

        for (j = 0; j < *n*; j++)

        {

            if (i != j) *// Excluding all i == j*

            {

                float pro = *a*[j][i] / *a*[i][i]; *// converting matrix to reduced row, echelon form (diagonal matrix)*

                for (k = 0; k <= *n*; k++)

*a*[j][k] = *a*[j][k] - (*a*[i][k] \* pro);

            }

        }

    }

    return flag;

}

int CheckConsistencY(float *a*[][M], int *n*, int *flag*)

{

    int i, j;

    float sum;

*flag* = 3; *// flag == 2 for infinite solution & flag == 3 for No solution*

    for (i = 0; i < *n*; i++)

    {

        sum = 0;

        for (j = 0; j < *n*; j++)

            sum += *a*[i][j];

        if (sum == *a*[i][j])

*flag* = 2;

    }

    return *flag*;

}

void PrintResult(float *a*[][M], int *n*, int *flag*)

{

    if (*flag* == 2)

        printf("The matrix has infinite solutions --> consistent\n");

    else if (*flag* == 3)

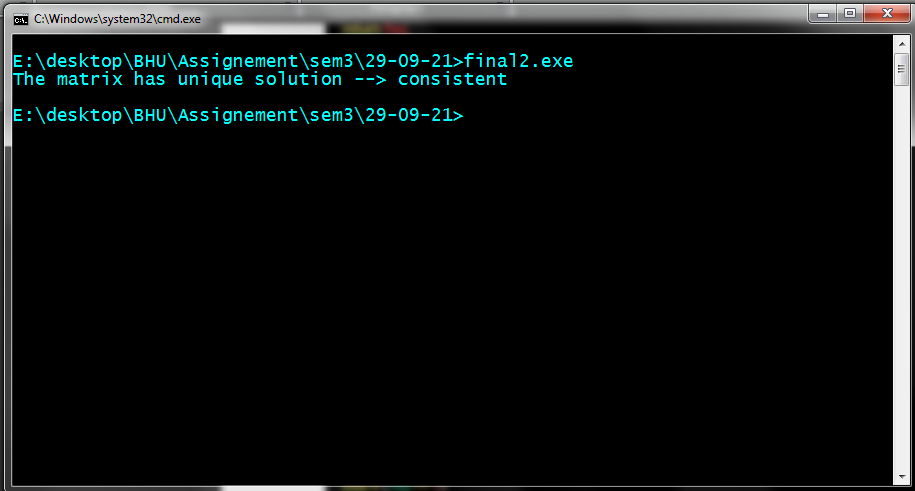
        printf("The matrix has no solution --> Inconsistent\n");

    else

        printf("The matrix has unique solution --> consistent\n");

}

Output:



Question 3: *// Write a program to implement Gauss elimination method*

*// Exam Roll No. - 20220PHY014*

#include <stdio.h>

#include <math.h>

#define N 3

int forwardElim(double *mat*[N][N + 1]); *// tells whether a matrix is singular or not*

void backSub(double *mat*[N][N + 1]);

void gaussianElimination(double *mat*[N][N + 1]);

void swap\_row(double *mat*[N][N + 1], int *i*, int *j*);

void print(double *mat*[N][N + 1]);

int main()

{

    double mat[N][N + 1] = {{3, 2, -4, 3},   *// 3x + 2y - 4z = 3*

                            {2, 3, 3, 15},   *// 2x + 3y + 3z = 15*

                            {5, -3, 1, 14}}; *// 5x - 3y + z = 14*

    gaussianElimination(mat);

    return 0;

}

void gaussianElimination(double *mat*[N][N + 1])

{

    int singular\_flag = forwardElim(*mat*);

    if (singular\_flag != -1) *// if matrixis singular*

    {

        printf("Singular Matrix\n");

        if (*mat*[singular\_flag][N]) *// if the RHS of equation corresponding to zero row is 0, system has infinitely many solutios, else inconsistent*

            printf("Inconsistent System\n");

        else

            printf("May have infinitely many solutions\n");

        return;

    }

    backSub(*mat*); *// get solution to system and print it using backward subsitution*

}

void swap\_row(double *mat*[N][N + 1], int *i*, int *j*)

{

    for (int k = 0; k <= N; k++)

    {

        double temp = *mat*[*i*][k];

*mat*[*i*][k] = *mat*[*j*][k];

*mat*[*j*][k] = temp;

    }

}

void print(double *mat*[N][N + 1])

{

    for (int i = 0; i < N; i++, printf("\n"))

    {

        for (int j = 0; j <= N; j++)

            printf("*%lf* ", *mat*[i][j]);

    }

    printf("\n");

}

int forwardElim(double *mat*[N][N + 1])

{

    for (int k = 0; k < N; k++)

    {

*// Initialize maximum value and index for pivot*

        int i\_max = k;

        int v\_max = *mat*[i\_max][k];

        for (int i = k + 1; i < N; i++) *// find greater amplitude for pivot if any*

        {

            if (abs(*mat*[i][k]) > v\_max)

                v\_max = *mat*[i][k], i\_max = i;

        }

        if (!*mat*[k][i\_max]) *// if a prinicipal diagonal element is zero, it denotes that matrix is singular, and will lead to a division by zero later*

            return k;       *// matrix is singular*

        if (i\_max != k) *// Swap the greatest value row with current row*

            swap\_row(*mat*, k, i\_max);

        for (int i = k + 1; i < N; i++)

        {

            double f = *mat*[i][k] / *mat*[k][k]; *// factor f to set current row kth element to 0, and subsequentl remaining kth column to 0*

            for (int j = k + 1; j <= N; j++) *// subtract fth multiple of corresponding kth row element*

*mat*[i][j] -= (*mat*[k][j] \* f);

*mat*[i][k] = 0; *// filling lower triangular matrix with zeros*

        }

    }

    return -1;

}

void backSub(double *mat*[N][N + 1])

{

    double x[N]; *// An array to store solution*

    for (int i = N - 1; i >= 0; i--) *// start calculating from last equation up to the first*

    {

        x[i] = *mat*[i][N]; *// start with the RHS of the equation*

        for (int j = i + 1; j < N; j++) *// Initialize j to i+1 since matrix is upper triangular*

            x[i] -= *mat*[i][j] \* x[j];   *// subtract all the lhs values except the coefficient of the variable whose value is being calculated*

        x[i] /= *mat*[i][i]; *// divide the RHS by the coefficient of the unknown being calculated*

    }

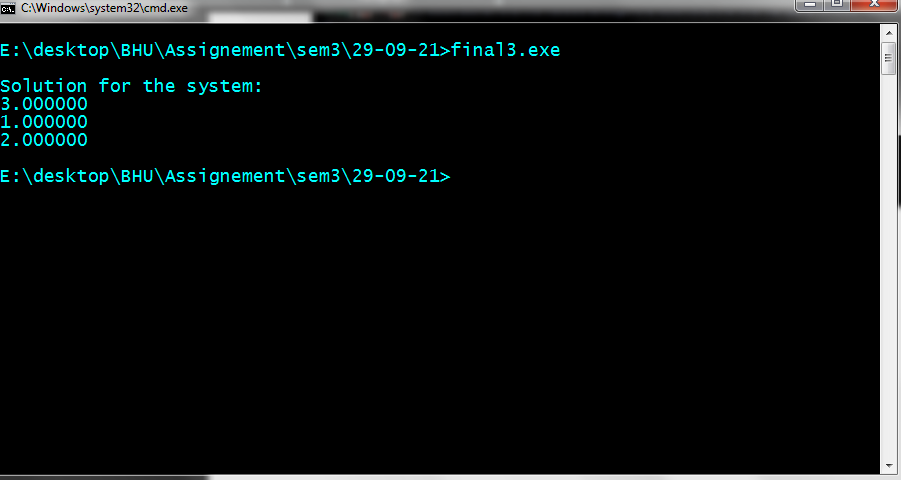
    printf("\nSolution for the system:\n");

    for (int i = 0; i < N; i++)

        printf("*%lf*\n", x[i]);

}

Output:



Question 4: *// Write a program to decompose a matrix into Lower triangular and upper triangular matrices.*

*// Exam Roll No. - 20220PHY014*

#include <stdio.h>

void printMatrix(float *matrix*[10][10], int *size*);

int main()

{

    int size, row, column, ctr1, ctr2;

    size = 4;

    float matrix[10][10] = {{2, 3, 1, 5},

                            {6, 13, 5, 19},

                            {2, 19, 10, 23},

                            {4, 10, 11, 31}};

    printf("\nYour matrix is:\n");

    printMatrix(matrix, size);

    float lowerTriangle[10][10];

    float upperTriangluar[10][10];

    for (row = 0; row < size; row++)

    {

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

        {

            if (row > column) *// initialise all elements of lower trianle in upper triangular matrix*

                upperTriangluar[row][column] = 0;

            if (row < column) *// initialise all elements of upper triangle in lower triangular matrix*

                lowerTriangle[row][column] = 0;

            if (row == column) *// initialise all diagonal elements of lower triangle in lower triangular matrix*

                lowerTriangle[row][column] = 1;

        }

    }

    for (ctr1 = 0; ctr1 < size; ctr1++)

    {

        upperTriangluar[ctr1][ctr1] = matrix[ctr1][ctr1];

        for (ctr2 = ctr1 + 1; ctr2 < size; ctr2++)

        {

            upperTriangluar[ctr1][ctr2] = matrix[ctr1][ctr2];

            lowerTriangle[ctr2][ctr1] = matrix[ctr2][ctr1] / upperTriangluar[ctr1][ctr1];

        }

        for (row = ctr1 + 1; row < size; row++)

        {

            for (column = ctr1 + 1; column < size; column++)

                matrix[row][column] -= (lowerTriangle[row][ctr1] \* upperTriangluar[ctr1][column]);

        }

    }

    printf("\nUpper Triangular Matrix:\n");

    printMatrix(upperTriangluar, size);

    printf("\nLower Triangular Matrix:\n");

    printMatrix(lowerTriangle, size);

    return 0;

}

void printMatrix(float *matrix*[10][10], int *size*)

{

    for (int row = 0; row < *size*; row++)

    {

        for (int column = 0; column < *size*; column++)

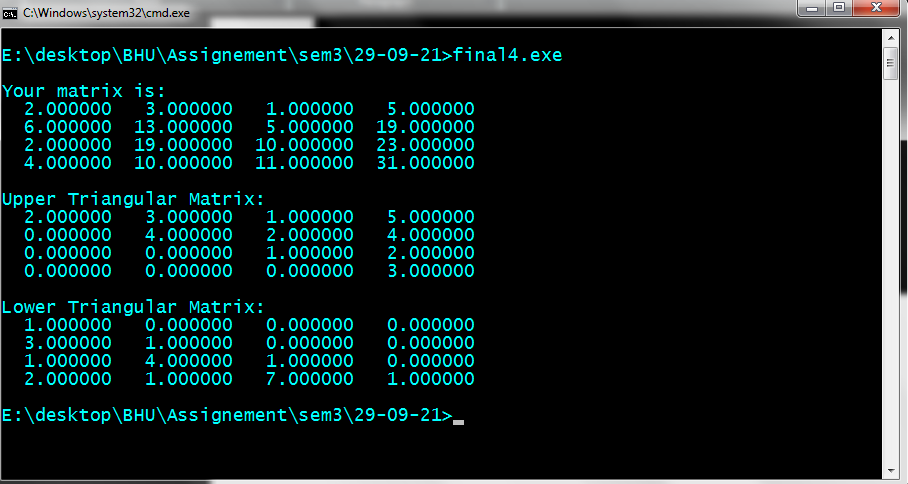
            printf("*%10f* ", *matrix*[row][column]);

        printf("\n");

    }

}

Output:



Question 5: *// Write a program to implement Crout's method.*

*// Exam Roll No. - 20220PHY014*

#include <stdio.h>

#include <math.h>

void printMatrix(float *matrix*[10][10], int *order*);

int main()

{

    int order = 3;

    float solution[10], lower[10][10], upper[10][10];

    float linear\_equation[10][10] = {{2, 1, 4, 12},    *// 2x + y + 4z = 12*

                                     {8, -3, 2, 20},   *// 8x - 3y + 2z = 20*

                                     {4, 11, -1, 33}}; *// 4x + 11y - z = 33*

    for (int i = 0; i < order; i++)

        lower[i][0] = linear\_equation[i][0];

    for (int j = 1; j <= order; j++)

        upper[0][j] = (linear\_equation[0][j] / lower[0][0]);

    for (int p = 1; p < order; p++)

    {

        for (int i = p; i < order; i++)

        {

            for (int k = 0; k < p; k++)

                linear\_equation[i][p] -= lower[i][k] \* upper[k][p];

            lower[i][p] = linear\_equation[i][p];

        }

        for (int j = p + 1; j <= order; j++)

        {

            for (int k = 0; k < p; k++)

                linear\_equation[p][j] -= lower[p][k] \* upper[k][j];

            upper[p][j] = linear\_equation[p][j] / lower[p][p];

        }

    }

    for (int k = 0; k < order; k++)

    {

        int i = (order - k - 1);

        solution[i] = upper[i][order];

        for (int j = i + 1; j < order; j++)

            solution[i] -= (upper[i][j] \* solution[j]);

    }

    printf("\nThe result is:\n");

    for (int i = 0; i < order; i++)

        printf("x[*%d*] = *%f*\n", i, solution[i]);

    return 0;

}

void printMatrix(float *matrix*[10][10], int *order*)

{

    for (int row = 0; row < *order*; row++)

    {

        for (int column = 0; column < *order* + 1; column++)

            printf("*%10f* ", *matrix*[row][column]);

        printf("\n");

    }

}

Output:

