lab08

```c
// EE231002 Lab08. Matrix Determinant
// 108061112, 林靖
// Nov. 16, 2019

#include <stdio.h>          // Standard input and output library

#if !defined(N)             // To facilitate recompilation
#define N 3                 // Dimension of matrix
#endif

double det(double A[N][N], int dim);        // Recursion with cofactor expansion

int main(void)              // Called at program startup
{
    int row, column;        // Index of matrix
    double A[N][N];         // Store matrix to compute determinant

    printf("Matrix A is\n");
    for (row = 0; row < N; row++) {                     // Loop for each element
        for (column = 0; column < N; column++) {
            scanf("%lg", &A[row][column]);              // Read in elements and
            printf("%3lg", A[row][column]);             // Print them out
        }
        printf("\n");                       // New line
    }
    printf("det(A) = %lg\n", det(A, N));    // Call to evaluate determinant
    return 0;                               // Normal program termination
}

double det(double A[N][N], int dim)         // Evaluate determinant by recursion
{
    int A_row, A_column;            // Indices of the determinant to be evaluate
    double sub[N][N];               // Sub-matrix of cofactor
    int sub_row, sub_column;        // Indices of sub-matrix
    double sum = 0;                 // Sum up each term of cofactor
    int sign = 1;                   // (-1)^n changing sign for each term
    int column_remove;              // To remove specific column for sub-matrix

    if (dim == 3) {                 // Solvable base case, return directly
        Should terminate recursion when dim = 2
        return (A[0][0] * (A[1][1] * A[2][2] - A[2][1] * A[1][2]) -
                A[0][1] * (A[1][0] * A[2][2] - A[2][0] * A[1][2]) +
                A[0][2] * (A[1][0] * A[2][1] - A[2][0] * A[1][1])   );
                A[0][2] * (A[1][0] * A[2][1] - A[2][0] * A[1][1]));
    }
    for (column_remove = 0; column_remove < dim; column_remove++) {
        sub_column = 0;                                     // Copy only the
        for (A_column = 0; A_column < dim; A_column++) {    // elements which
```

```
47              if (A_column != column_remove) {                // aren't in the
48                  sub_row = 0;                                // given column into
49                  for (A_row = 1; A_row < dim; A_row++) {     // the sub-matrix.
50                      sub[sub_row][sub_column] = A[A_row][A_column];
51                      sub_row++;
52                  }
53                  sub_column++;       // To copy elements into the next column
54              }
55          }               // Trigger recursion and sum up each term of cofactor
56          sum += sign * A[0][column_remove] * det(sub, dim - 1);
57          sign *= -1;     // Change the sign multiplier for the next term
58      }
59      return sum;         // Return to lower stack frame
60  }
```

[CPU time] 0.301049 sec
[Format] can be improved.
[Memory] efficiency can be improved.

Score: 90