two-dimensional arrays

Oftentimes, there are advantages to defining an array of variables using a “two-dimensional” arrangement. A two-dimensional array could be considered to have “rows” and “columns”. The declaration of a two-dimensional array is extension of the declaration for a 1-D (linear) array. The first dimension is the “row” and the second is the “column”.

```
int array2D[3][3];        \\A “3x3” array of integers
```

array2D[0][0]	array2D[0][1]	array2D[0][2]
array2D[1][0]	array2D[1][1]	array2D[1][2]
array2D[2][0]	array2D[2][1]	array2D[2][2]

To access a particular array element, just use the appropriate index values. For example: `array2D[1][2] = 17;`

However, in memory, the array is not stored in a 2-D fashion. The elements are still in a linear arrangement, with the first row stored first, followed by the second row, then the third row, etc.

When accessing the variable using the array name, the distinction is not important. It will be important to understand this arrangement when we try to access using memory location. (pointers)

address	array element
00000	array2D[0][0]
00001	array2D[0][1]
00010	array2D[0][2]
00011	array2D[1][0]
00100	array2D[1][1]
00101	array2D[1][2]
00110	array2D[2][0]
00111	array2D[2][1]
01000	array2D[2][2]
01001	
01010	
01011	

The 2D nature of the array arrangement leads naturally to nested loops.

The array elements can be filled or changed using the usual assignment statements within the program. The array can also be initialized at the time that it is defined.

```
int array2D[3][3] = { {1,2,3}, {4,5,6}, {7,8,9} };
```

Or the statement can be spread over multiple lines to make the two-dimensional nature of the array more obvious.

```
int array2D[3][3] = {

    {1,2,3},

    {4,5,6},

    {7,8,9} };
```

Higher dimensionality is possible.

For example: `int array3D[3][3][3];`

However, memory requirements expand rapidly, and keeping track of the elements becomes complicated. Generally, stick to 1-D or 2-D arrays unless a particular problem would benefit from a higher dimension.

Example program with 2-D arrays.

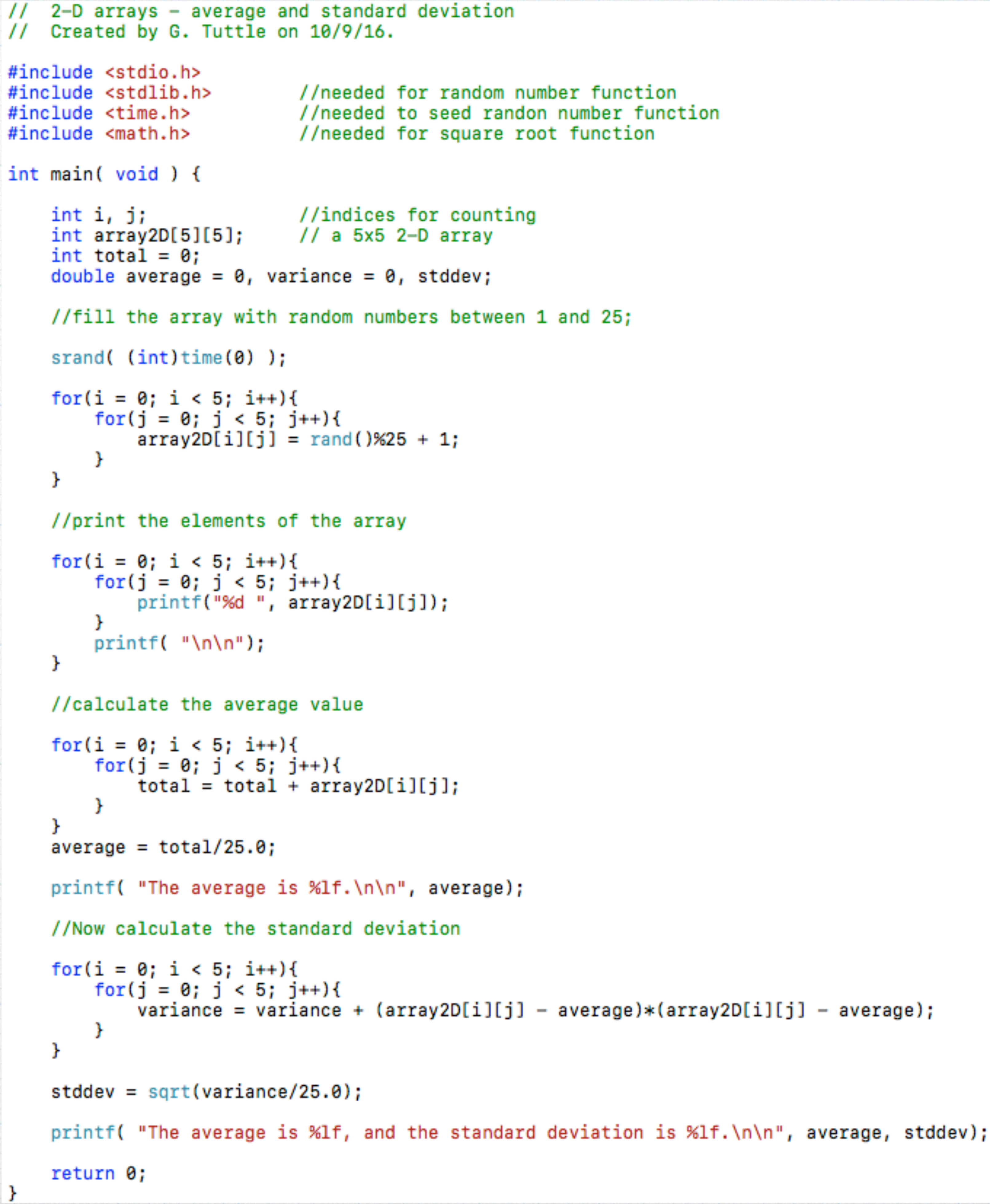
1) Fill a 5x5 array with random values.

2) Print the values.

3) Calculate average.

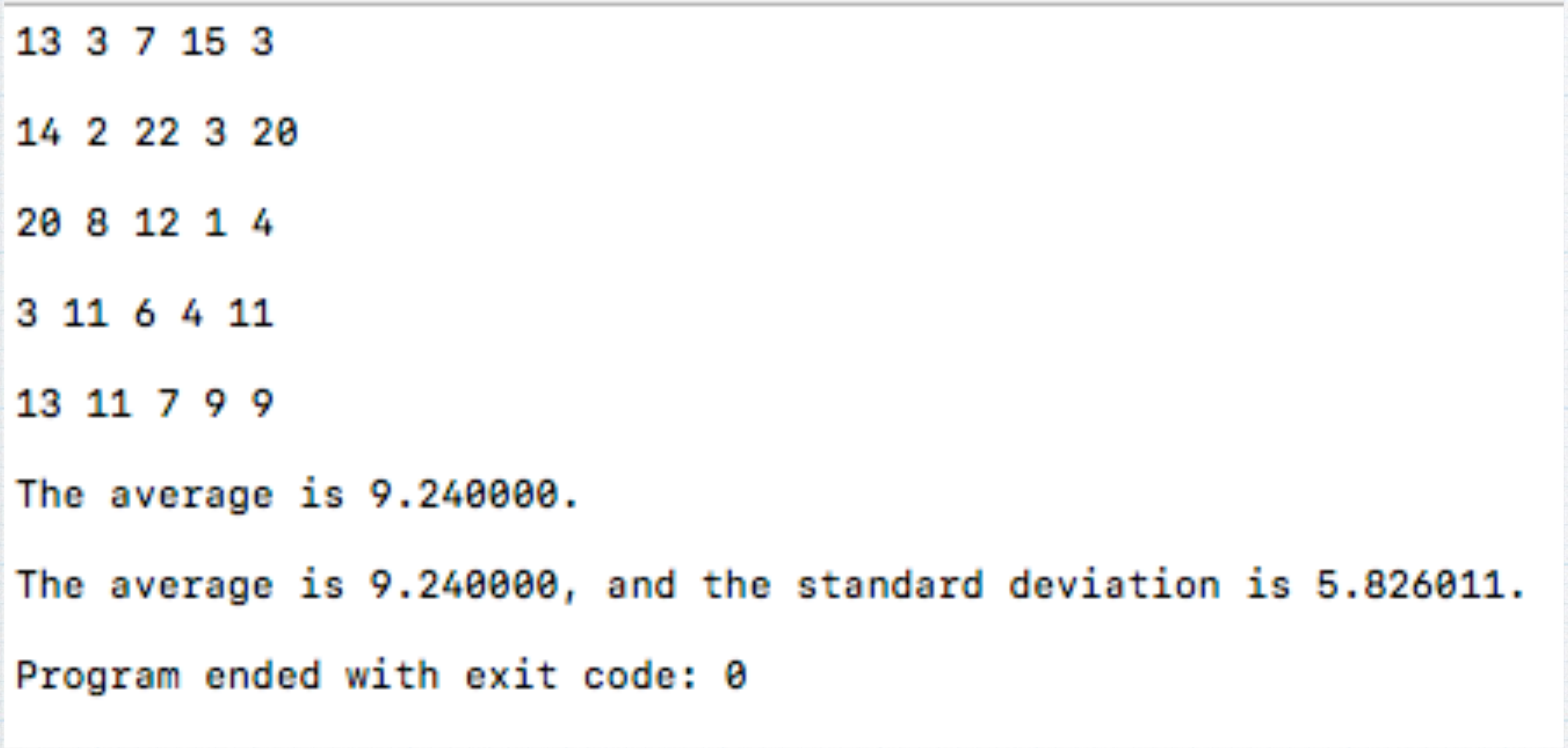
4) Calculate standard deviation

Code file is on the GitHub.

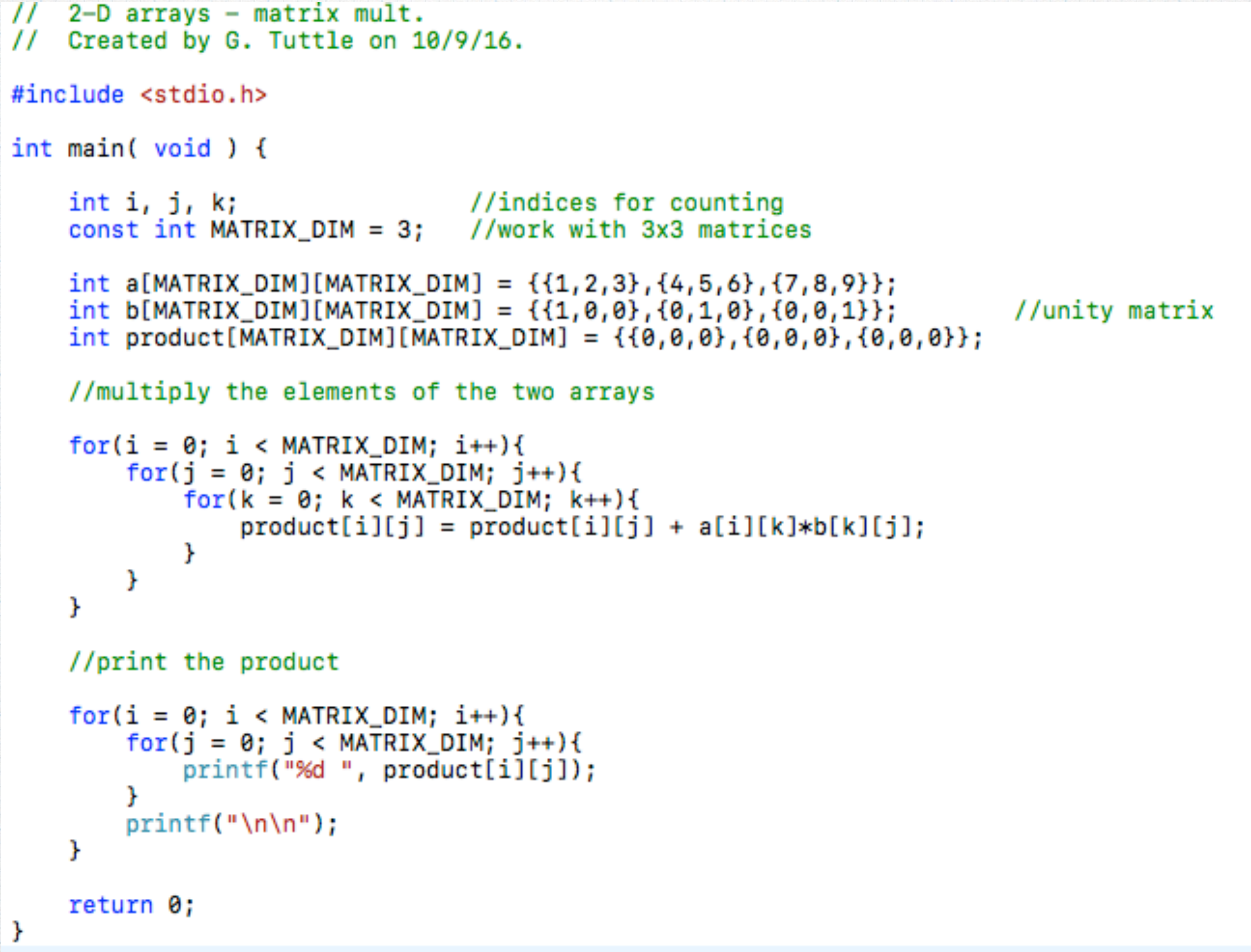


```c
//  2-D arrays - average and standard deviation
//  Created by G. Tuttle on 10/9/16.

#include <stdio.h>
#include <stdlib.h>        //needed for random number function
#include <time.h>          //needed to seed randon number function
#include <math.h>          //needed for square root function

int main( void ) {

    int i, j;              //indices for counting
    int array2D[5][5];     // a 5x5 2-D array
    int total = 0;
    double average = 0, variance = 0, stddev;

    //fill the array with random numbers between 1 and 25;

    srand( (int)time(0) );

    for(i = 0; i < 5; i++){
        for(j = 0; j < 5; j++){
            array2D[i][j] = rand()%25 + 1;
        }
    }

    //print the elements of the array

    for(i = 0; i < 5; i++){
        for(j = 0; j < 5; j++){
            printf("%d ", array2D[i][j]);
        }
        printf( "\n\n");
    }

    //calculate the average value

    for(i = 0; i < 5; i++){
        for(j = 0; j < 5; j++){
            total = total + array2D[i][j];
        }
    }
    average = total/25.0;

    printf( "The average is %lf.\n\n", average);

    //Now calculate the standard deviation

    for(i = 0; i < 5; i++){
        for(j = 0; j < 5; j++){
            variance = variance + (array2D[i][j] - average)*(array2D[i][j] - average);
        }
    }

    stddev = sqrt(variance/25.0);

    printf( "The average is %lf, and the standard deviation is %lf.\n\n", average, stddev);

    return 0;
}
```

Example output.



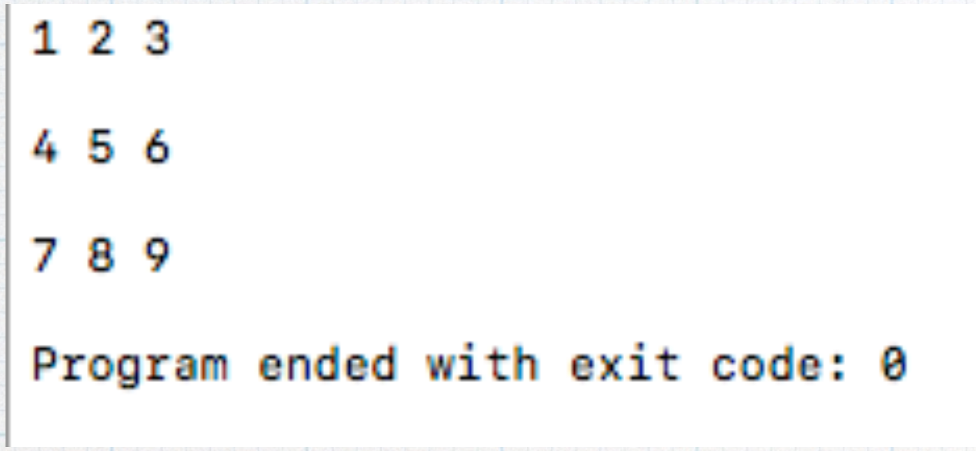
```
13 3 7 15 3

14 2 22 3 20

20 8 12 1 4

3 11 6 4 11

13 11 7 9 9

The average is 9.240000.

The average is 9.240000, and the standard deviation is 5.826011.

Program ended with exit code: 0
```



```c
//  2-D arrays - matrix mult.
//  Created by G. Tuttle on 10/9/16.

#include <stdio.h>

int main( void ) {

    int i, j, k;                //indices for counting
    const int MATRIX_DIM = 3;   //work with 3x3 matrices

    int a[MATRIX_DIM][MATRIX_DIM] = {{1,2,3},{4,5,6},{7,8,9}};
    int b[MATRIX_DIM][MATRIX_DIM] = {{1,0,0},{0,1,0},{0,0,1}};        //unity matrix
    int product[MATRIX_DIM][MATRIX_DIM] = {{0,0,0},{0,0,0},{0,0,0}};

    //multiply the elements of the two arrays

    for(i = 0; i < MATRIX_DIM; i++){
        for(j = 0; j < MATRIX_DIM; j++){
            for(k = 0; k < MATRIX_DIM; k++){
                product[i][j] = product[i][j] + a[i][k]*b[k][j];
            }
        }
    }

    //print the product

    for(i = 0; i < MATRIX_DIM; i++){
        for(j = 0; j < MATRIX_DIM; j++){
            printf("%d ", product[i][j]);
        }
        printf("\n\n");
    }

    return 0;
}
```

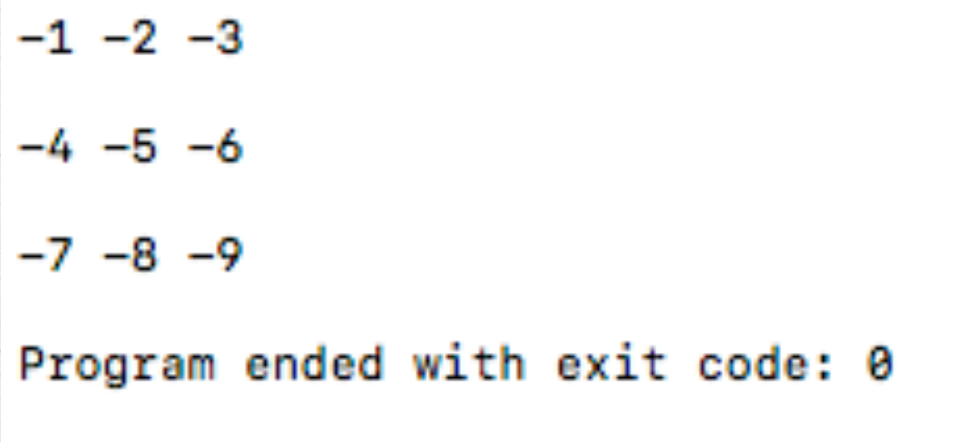
Code file is on the GitHub.

Example with [b] = unity matrix.



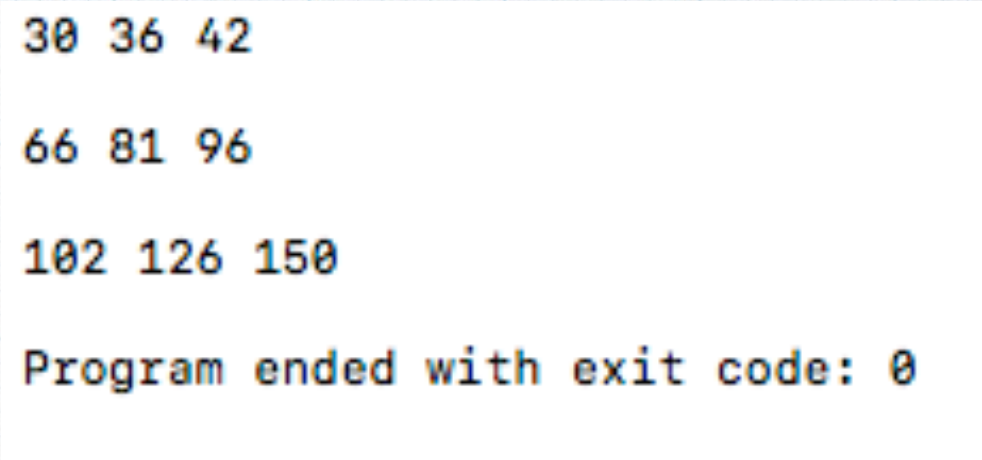
```
1 2 3

4 5 6

7 8 9

Program ended with exit code: 0
```

Example with [b] = negative of unity matrix.



```
-1 -2 -3

-4 -5 -6

-7 -8 -9

Program ended with exit code: 0
```

Example with [b] = [a].



```
30 36 42

66 81 96

102 126 150

Program ended with exit code: 0
```