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”.

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

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)

<table>
<thead>
<tr>
<th>address</th>
<th>array element</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>array2D[0][0]</td>
</tr>
<tr>
<td>00001</td>
<td>array2D[0][1]</td>
</tr>
<tr>
<td>00010</td>
<td>array2D[0][2]</td>
</tr>
<tr>
<td>00011</td>
<td>array2D[1][0]</td>
</tr>
<tr>
<td>00100</td>
<td>array2D[1][1]</td>
</tr>
<tr>
<td>00101</td>
<td>array2D[1][2]</td>
</tr>
<tr>
<td>00110</td>
<td>array2D[2][0]</td>
</tr>
<tr>
<td>00111</td>
<td>array2D[2][1]</td>
</tr>
<tr>
<td>01000</td>
<td>array2D[2][2]</td>
</tr>
<tr>
<td>01001</td>
<td></td>
</tr>
<tr>
<td>01010</td>
<td></td>
</tr>
<tr>
<td>01011</td>
<td></td>
</tr>
</tbody>
</table>

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.

```c
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.

```c
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.

```
#include <stdio.h>
#include <stdlib.h> // needed for random number function
#include <time.h>  // needed to seed random 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 %.1f.\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 %.1f, and the standard deviation is %.1f.\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
Code file is on the GitHub.
Example with \( [b] = \) unity matrix.

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

Program ended with exit code: 0

Example with \( [b] = \) negative of unity matrix.

\[
\begin{bmatrix}
-1 & -2 & -3 \\
-4 & -5 & -6 \\
-7 & -8 & -9
\end{bmatrix}
\]

Program ended with exit code: 0

Example with \( [b] = [a] \).

\[
\begin{bmatrix}
30 & 36 & 42 \\
66 & 81 & 96 \\
102 & 126 & 150
\end{bmatrix}
\]

Program ended with exit code: 0