CS 162C++
Flattening Arrays

Flattening arrays is another term for representing a multi-dimensional array as a one-dimensional array.

Why Bother

When you pass a multi-dimensional array to a function, you must specify the size of each dimension after the first. That is, for two-dimensional arrays, you can vary the number of rows, but the number of columns is fixed. For three-dimensional arrays, you can vary the number of rows, but the number of columns and height is fixed.

Why

The compiler must convert your multi-dimensional array to a single dimension since that is all that it has in memory. Thus, it must convert a reference of the type array2[row][col] to a reference of the type array1[index]. If you want to write functions that allow for both arbitrary row and column lengths, you need to fake it with a one-dimensional array.

If you have a two-dimensional array array2 that has 4 rows and 5 columns, then it would appear as follows:

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4

When you convert this to linear memory, effectively each row is laid out in order so that it appears as:

0,0	0,1	0,2	0,3	0,4	0,4	1,0	1,1	1,2	1,3	1,4	2,0	2,1	2,2	2,3	2,4	3,0	3,1	3,2	3,3	3,4

Alternatively, you could consider this as a one-dimensional array of length twenty:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

To convert the row/column location to the index location you use the formula

index = columnPosition + rowPosition * NumberOfColumns

Example

If your program calculated the multiplication table up to 7 * 8 as shown here:

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    const int ROW = 8;
    const int COL = 7;
    int values[ROW][COL];

        // fill with multiplication table
    for(int i = 0; i < ROW; i++)
        for (int j = 0; j < COL; j++)
            values[i][j] = (i+1) * (j+1);

        // display
    for(int i = 0; i < ROW; i++)
    {
        for(int j = 0; j < COL; j++)
            cout << setw(5) << values[i][j];
        cout << endl;
    }
    cout << endl;

    return 0;
}
```

Then you could replace it with this program using a one-dimensional array:

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    const int ROW = 8;
    const int COL = 7;
    int values[ROW * COL];

        // fill with multiplication table
    for(int i = 0; i < ROW; i++)
        for (int j = 0; j < COL; j++)
            values[j + i * COL] = (i+1) * (j+1);

        // display
    for(int i = 0; i < ROW; i++)
    {
        for(int j = 0; j < COL; j++)
            cout << setw(5) << values[j + i * COL];
        cout << endl;
    }
    cout << endl;

    return 0;
}
```

All that was replaced was the definition of the array as values[ROW][COL] -> values[ROW * COL] and references of values[i][j] -> values[j + I * COL].