2D Arrays & Functions

Pass 2D arrays to functions by address, just like 1D arrays. The following function prints the contents of a `ROWS × COLS` **2D** array of `double`:

```
void print(const double m[ROWS][COLS])
{
    for (size_t row = 0; row < ROWS; ++row)
    {
        for (size_t col = 0; col < COLS; col++)
            cout << setw(5) << m[row][col];
        cout << endl;   // end of each row
    }
}
```

Of course, **this function is really quite limited** since **it can only be used** to process an array that is **exactly** `ROWS x COLS` elements in size.

You can make it a little more flexible by **omitting the first dimension**, and then passing the number of rows as a parameter. **You cannot**, however, leave off the number of columns. That must be a constant.

```
void print(const double m[][COLS], size_t nRows)
{
    for (size_t row = 0; row < nRows; ++row)
    {
      ...
    }
}
```

This inflexibility is one of the reasons that the built-in **2D** arrays are so limiting in C/C++.

An expression that uses just one subscript with a **2D** array **represents a single row** within the **2D** array. This row is itself a **1D** array. Thus, if `a2d` is a **2D** array and `i` is an integer, then the expression `a2d[i]` is a **1D** array representing row `i`.