find_largest and store_zeros are more versatile than you might expect. Consider find_largest, which we originally designed to find the largest element of a one-dimensional array. We can just as easily use find_largest to determine the largest element in row i of the two-dimensional array a:

```
largest = find_largest(a[i], NUM_COLS);
```

Processing the Columns of a Multidimensional Array

Processing the elements in a *column* of a two-dimensional array isn't as easy, because arrays are stored by row, not by column. Here's a loop that clears column i of the array a:

```
int a[NUM_ROWS][NUM_COLS], (*p)[NUM_COLS], i;
...
for (p = &a[0]; p < &a[NUM_ROWS]; p++)
  (*p)[i] = 0;
```

I've declared p to be a pointer to an array of length NUM_COLS whose elements are integers. The parentheses around *p in (*p)[NUM_COLS] are required; without them, the compiler would treat p as an array of pointers instead of a pointer to an array. The expression p++ advances p to the beginning of the next row. In the expression (*p)[i], *p represents an entire row of a, so (*p)[i] selects the element in column i of that row. The parentheses in (*p)[i] are essential, because the compiler would interpret *p[i] as *(p[i]).

Using the Name of a Multidimensional Array as a Pointer

Just as the name of a one-dimensional array can be used as a pointer, so can the name of *any* array, regardless of how many dimensions it has. Some care is required, though. Consider the following array:

```
int a[NUM_ROWS][NUM_COLS];
```

a is *not* a pointer to a[0][0]; instead, it's a pointer to a[0]. This makes more sense if we look at it from the standpoint of C, which regards a not as a two-dimensional array but as a one-dimensional array whose elements are one-dimensional arrays. When used as a pointer, a has type int (*)[NUM_COLS] (pointer to an integer array of length NUM_COLS).

Knowing that a points to a[0] is useful for simplifying loops that process the elements of a two-dimensional array. For example, instead of writing

```
for (p = &a[0]; p < &a[NUM_ROWS]; p++)
  (*p)[i] = 0;
```

to clear column i of the array a, we can write

```
for (p = a; p < a + NUM_ROWS; p++)
  (*p)[i] = 0;
```