```
#define N 10
...
int a[N], i, sum = 0, *p = a;
...
for (i = 0; i < N; i++)
  sum += p[i];
```

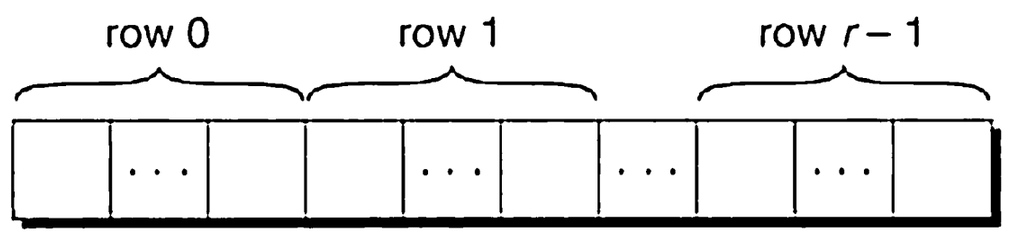
The compiler treats `p[i]` as `*(p+i)`, which is a perfectly legal use of pointer arithmetic. Although the ability to subscript a pointer may seem to be little more than a curiosity, we'll see in Section 17.3 that it's actually quite useful.

12.4 Pointers and Multidimensional Arrays

Just as pointers can point to elements of one-dimensional arrays, they can also point to elements of multidimensional arrays. In this section, we'll explore common techniques for using pointers to process the elements of multidimensional arrays. For simplicity, I'll stick to two-dimensional arrays, but everything we'll do applies equally to higher-dimensional arrays.

Processing the Elements of a Multidimensional Array

We saw in Section 8.2 that C stores two-dimensional arrays in row-major order; in other words, the elements of row 0 come first, followed by the elements of row 1, and so forth. An array with r rows would have the following appearance:



We can take advantage of this layout when working with pointers. If we make a pointer `p` point to the first element in a two-dimensional array (the element in row 0, column 0), we can visit every element in the array by incrementing `p` repeatedly.

As an example, let's look at the problem of initializing all elements of a two-dimensional array to zero. Suppose that the array has been declared as follows:

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

The obvious technique would be to use nested `for` loops:

```
int row, col;
...
for (row = 0; row < NUM_ROWS; row++)
  for (col = 0; col < NUM_COLS; col++)
    a[row][col] = 0;
```