

bers are still stored in an array; we're simply using a different technique to keep track of where we are in the array.

Note that the second argument to `scanf` is `p`, not `&p`. Since `p` points to an array element, it's a satisfactory argument for `scanf`; `&p`, on the other hand, would be a pointer to a pointer to an array element.

Array Arguments (Revisited)

When passed to a function, an array name is always treated as a pointer. Consider the following function, which returns the largest element in an array of integers:

```
int find_largest(int a[], int n)
{
    int i, max;

    max = a[0];
    for (i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];
    return max;
}
```

Suppose that we call `find_largest` as follows:

```
largest = find_largest(b, N);
```

This call causes a pointer to the first element of `b` to be assigned to `a`; the array itself isn't copied.

The fact that an array argument is treated as a pointer has some important consequences:

- When an ordinary variable is passed to a function, its value is copied; any changes to the corresponding parameter don't affect the variable. In contrast, an array used as an argument isn't protected against change, since no copy is made of the array itself. For example, the following function (which we first saw in Section 9.3) modifies an array by storing zero into each of its elements:

```
void store_zeros(int a[], int n)
{
    int i;

    for (i = 0; i < n; i++)
        a[i] = 0;
}
```

To indicate that an array parameter won't be changed, we can include the word `const` in its declaration:

```
int find_largest(const int a[], int n)
{
    ...
}
```