Invoking the `new` operator secures a block of memory large enough to hold `num_elements` elements of type `type_name`, with `pointer_name` pointing to the first element. As you're about to see, you can use `pointer_name` in many of the same ways you can use an array name.

**Using a Dynamic Array**

After you create a dynamic array, how do you use it? First, think about the problem conceptually. The statement

```cpp
int * psome = new int [10]; // get a block of 10 ints
```

creates a pointer `psome` that points to the first element of a block of ten `int` values. Think of it as a finger pointing to that element. Suppose an `int` occupies four bytes. Then, by moving your finger four bytes in the correct direction, you can point to the second element. Altogether, there are ten elements, which is the range over which you can move your finger. Thus, the `new` statement supplies you with all the information you need to identify every element in the block.

Now think about the problem practically. How do you access one of these elements? The first element is no problem. Because `psome` points to the first element of the array, `*psome` is the value of the first element. That leaves nine more elements to access. The simplest way may surprise you if you haven't worked with C: Just use the pointer as if it were an array name. That is, you can use `psome[0]` instead of `*psome` for the first element, `psome[1]` for the second element, and so on. It turns out to be very simple to use a pointer to access a dynamic array, even if it may not immediately be obvious why the method works. The reason you can do this is that C and C++ handle arrays internally by using pointers anyway. This near equivalence of arrays and pointers is one of the beauties of C and C++. We'll elaborate on this equivalence in a moment. First, **Listing 4.13** shows how you can use `new` to create a dynamic array and then use array notation to access the elements. It also points out a fundamental difference between a pointer and a true array name.

**Listing 4.13** arraynew.cpp

```cpp
// arraynew.cpp _ using the new operator for arrays
```