Combining the * and ++ Operators

C programmers often combine the * (indirection) and ++ operators in statements that process array elements. Consider the simple case of storing a value into an array element and then advancing to the next element. Using array subscripting, we might write

```
a[i++] = j;
```

If p is pointing to an array element, the corresponding statement would be

```
*p++ = j;
```

Because the postfix version of ++ takes precedence over *, the compiler sees this as

```
*(p++) = j;
```

The value of p++ is p. (Since we're using the postfix version of ++, p won't be incremented until after the expression has been evaluated.) Thus, the value of *(p++) will be *p—the object to which p is pointing.

Of course, *p++ isn't the only legal combination of * and ++. We could write (*p)++, for example, which returns the value of the object that p points to, and then increments that object (p itself is unchanged). If you find this confusing, the following table may help:

Expression	*Meaning*
*p++ or *(p++)	Value of expression is *p before increment; increment p later
(*p)++	Value of expression is *p before increment; increment *p later
*++p or *(++p)	Increment p first; value of expression is *p after increment
++*p or ++(*p)	Increment *p first; value of expression is *p after increment

All four combinations appear in programs, although some are far more common than others. The one we'll see most frequently is *p++, which is handy in loops. Instead of writing

```
for (p = &a[0]; p < &a[N]; p++)
  sum += *p;
```

to sum the elements of the array a, we could write

```
p = &a[0];
while (p < &a[N])
  sum += *p++;
```

The * and -- operators mix in the same way as * and ++. For an application that combines * and --, let's return to the stack example of Section 10.2. The original version of the stack relied on an integer variable named top to keep track of the "top-of-stack" position in the contents array. Let's replace top by a pointer variable that points initially to element 0 of the contents array:

```
int *top_ptr = &contents[0];
```