This declaration states that p is a pointer variable capable of pointing to *objects* of type int. I'm using the term *object* instead of *variable* since—as we'll see in Chapter 17—p might point to an area of memory that doesn't belong to a variable. (Be aware that "object" will have a different meaning when we discuss program design in Chapter 19.)

abstract objects ➤ 19.1

Pointer variables can appear in declarations along with other variables:

```
int i, j, a[10], b[20], *p, *q;
```

In this example, i and j are ordinary integer variables, a and b are arrays of integers, and p and q are pointers to integer objects.

C requires that every pointer variable point only to objects of a particular type (the *referenced type*):

```
int *p;       /* points only to integers   */
double *q;    /* points only to doubles    */
char *r;      /* points only to characters */
```

There are no restrictions on what the referenced type may be. In fact, a pointer variable can even point to another pointer.

pointers to pointers ➤ 17.6

11.2 The Address and Indirection Operators

C provides a pair of operators designed specifically for use with pointers. To find the address of a variable, we use the & (address) operator. If x is a variable, then &x is the address of x in memory. To gain access to the object that a pointer points to, we use the * (*indirection*) operator. If p is a pointer, then *p represents the object to which p currently points.

The Address Operator

Declaring a pointer variable sets aside space for a pointer but doesn't make it point to an object:

```
int *p;   /* points nowhere in particular */
```

It's crucial to initialize p before we use it. One way to initialize a pointer variable is to assign it the address of some variable—or, more generally, lvalue—using the & operator:

lvalues ➤ 4.2

```
int i, *p;
...
p = &i;
```

By assigning the address of i to the variable p, this statement makes p point to i: