The compiler sets aside eight characters for `date4`, enough to store the characters in `"June 14"` plus a null character. (The fact that the length of `date4` isn't specified doesn't mean that the array's length can be changed later. Once the program is compiled, the length of `date4` is fixed at eight.) Omitting the length of a string variable is especially useful if the initializer is long, since computing the length by hand is error-prone.

Character Arrays versus Character Pointers

Let's compare the declaration

```
char date[] = "June 14";
```

which declares `date` to be an *array*, with the similar-looking

```
char *date = "June 14";
```

which declares `date` to be a *pointer*. Thanks to the close relationship between arrays and pointers, we can use either version of `date` as a string. In particular, any function expecting to be passed a character array or character pointer will accept either version of `date` as an argument.

However, we must be careful not to make the mistake of thinking that the two versions of `date` are interchangeable. There are significant differences between the two:

- In the array version, the characters stored in `date` can be modified, like the elements of any array. In the pointer version, `date` points to a string literal, and we saw in Section 13.1 that string literals shouldn't be modified.
- In the array version, `date` is an array name. In the pointer version, `date` is a variable that can be made to point to other strings during program execution.

If we need a string that can be modified, it's our responsibility to set up an array of characters in which to store the string; declaring a pointer variable isn't enough. The declaration

```
char *p;
```

causes the compiler to set aside enough memory for a pointer variable; unfortunately, it doesn't allocate space for a string. (And how could it? We haven't indicated how long the string would be.) Before we can use `p` as a string, it must point to an array of characters. One possibility is to make `p` point to a string variable:

```
char str[STR_LEN+1], *p;

p = str;
```

`p` now points to the first character of `str`, so we can use `p` as a string. Another possibility is to make `p` point to a dynamically allocated string.

dynamically allocated strings ➤ 17.2