```
char *p;

p = "abc";
```

This assignment doesn't copy the characters in `"abc"`; it merely makes `p` point to the first character of the string.

C allows pointers to be subscripted, so we can subscript string literals:

```
char ch;

ch = "abc"[1];
```

The new value of `ch` will be the letter `b`. The other possible subscripts are 0 (which would select the letter `a`), 2 (the letter `c`), and 3 (the null character). This property of string literals isn't used that much, but occasionally it's handy. Consider the following function, which converts a number between 0 and 15 into a character that represents the equivalent hex digit:

```
char digit_to_hex_char(int digit)
{
  return "0123456789ABCDEF"[digit];
}
```

Attempting to modify a string literal causes undefined behavior:

```
char *p = "abc";

*p = 'd';    /*** WRONG ***/
```

Q&A A program that tries to change a string literal may crash or behave erratically.

String Literals versus Character Constants

A string literal containing a single character isn't the same as a character constant. The string literal `"a"` is represented by a *pointer* to a memory location that contains the character `a` (followed by a null character). The character constant `'a'` is represented by an *integer* (the numerical code for the character).

Don't ever use a character when a string is required (or vice versa). The call

```
printf("\n");
```

is legal, because `printf` expects a pointer as its first argument. The following call isn't legal, however:

```
printf('\n');    /*** WRONG ***/
```
