There are many functions in `<string.h>`; I'll cover a few of the most basic. In subsequent examples, assume that `str1` and `str2` are character arrays used as strings.

The `strcpy` (String Copy) Function

The `strcpy` function has the following prototype in `<string.h>`:

```
char *strcpy(char *s1, const char *s2);
```

`strcpy` copies the string `s2` into the string `s1`. (To be precise, we should say "`strcpy` copies the string pointed to by `s2` into the array pointed to by `s1`.") That is, `strcpy` copies characters from `s2` to `s1` up to (and including) the first null character in `s2`. `strcpy` returns `s1` (a pointer to the destination string). The string pointed to by `s2` isn't modified, so it's declared `const`.

The existence of `strcpy` compensates for the fact that we can't use the assignment operator to copy strings. For example, suppose that we want to store the string `"abcd"` in `str2`. We can't use the assignment

```
str2 = "abcd";          /*** WRONG ***/
```

because `str2` is an array name and can't appear on the left side of an assignment. Instead, we can call `strcpy`:

```
strcpy(str2, "abcd");   /* str2 now contains "abcd" */
```

Similarly, we can't assign `str2` to `str1` directly, but we can call `strcpy`:

```
strcpy(str1, str2);     /* str1 now contains "abcd" */
```

Most of the time, we'll discard the value that `strcpy` returns. On occasion, though, it can be useful to call `strcpy` as part of a larger expression in order to use its return value. For example, we could chain together a series of `strcpy` calls:

```
strcpy(str1, strcpy(str2, "abcd"));
  /* both str1 and str2 now contain "abcd" */
```

⚠ In the call `strcpy(str1, str2)`, `strcpy` has no way to check that the string pointed to by `str2` will actually fit in the array pointed to by `str1`. Suppose that `str1` points to an array of length *n*. If the string that `str2` points to has no more than *n* – 1 characters, then the copy will succeed. But if `str2` points to a longer string, undefined behavior occurs. (Since `strcpy` always copies up to the first null character, it will continue copying past the end of the array that `str1` points to.)

`strncpy` function ➤*23.6*

Calling the `strncpy` function is a safer, albeit slower, way to copy a string. `strncpy` is similar to `strcpy` but has a third argument that limits the number of characters that will be copied. To copy `str2` into `str1`, we could use the following call of `strncpy`: