```
strncpy(str1, str2, sizeof(str1));
```

As long as `str1` is large enough to hold the string stored in `str2` (including the null character), the copy will be done correctly. `strncpy` itself isn't without danger, though. For one thing, it will leave the string in `str1` without a terminating null character if the length of the string stored in `str2` is greater than or equal to the size of the `str1` array. Here's a safer way to use `strncpy`:

```
strncpy(str1, str2, sizeof(str1) - 1);
str1[sizeof(str1)-1] = '\0';
```

The second statement guarantees that `str1` is always null-terminated, even if `strncpy` fails to copy a null character from `str2`.

## The `strlen` (String Length) Function

The `strlen` function has the following prototype:

```
size_t strlen(const char *s);
```

size_t type ➤7.6

`size_t`, which is defined in the C library, is a `typedef` name that represents one of C's unsigned integer types. Unless we're dealing with extremely long strings, this technicality need not concern us—we can simply treat the return value of `strlen` as an integer.

`strlen` returns the length of a string `s`: the number of characters in `s` up to, but not including, the first null character. Here are a few examples:

```
int len;

len = strlen("abc");    /* len is now 3 */
len = strlen("");       /* len is now 0 */
strcpy(str1, "abc");
len = strlen(str1);     /* len is now 3 */
```

The last example illustrates an important point. When given an array as its argument, `strlen` doesn't measure the length of the array itself; instead, it returns the length of the string stored in the array.

## The `strcat` (String Concatenation) Function

The `strcat` function has the following prototype:

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

`strcat` appends the contents of the string `s2` to the end of the string `s1`; it returns `s1` (a pointer to the resulting string).

Here are some examples of `strcat` in action:

```
strcpy(str1, "abc");
strcat(str1, "def");   /* str1 now contains "abcdef" */
```