509 characters long. (Yes, you read that right—509. Don't ask.) C99 increases the minimum to 4095 characters.

Q: Why aren't string literals called "string constants"?

A: Because they're not necessarily constant. Since string literals are accessed through pointers, there's nothing to prevent a program from attempting to modify the characters in a string literal.

Q: How do we write a string literal that represents "über" if `"\xfcber"` doesn't work? [p. 278]

A: The secret is to write two adjacent string literals and let the compiler join them into one. In this example, writing `"\xfc" "ber"` will give us a string literal that represents the word "über."

Q: Modifying a string literal seems harmless enough. Why does it cause undefined behavior? [p. 280]

A: Some compilers try to reduce memory requirements by storing single copies of identical string literals. Consider the following example:

```
char *p = "abc", *q = "abc";
```

A compiler might choose to store `"abc"` just once, making both `p` and `q` point to it. If we were to change `"abc"` through the pointer `p`, the string that `q` points to would also be affected. Needless to say, this could lead to some annoying bugs. Another potential problem is that string literals might be stored in a "read-only" area of memory; a program that attempts to modify such a literal will simply crash.

Q: Should every array of characters include room for a null character?

A: Not necessarily, since not every array of characters is used as a string. Including room for the null character (and actually putting one into the array) is necessary only if you're planning to pass it to a function that requires a null-terminated string.

You do *not* need a null character if you'll only be performing operations on individual characters. For example, a program might have an array of characters that it will use to translate from one character set to another:

```
char translation_table[128];
```

The only operation that the program will perform on this array is subscripting. (The value of `translation_table[ch]` will be the translated version of the character `ch`.) We would not consider `translation_table` to be a string: it need not contain a null character, and no string operations will be performed on it.

Q: If `printf` and `scanf` expect their first argument to have type `char *`, does that mean that the argument can be a string *variable* instead of a string *literal*?