Escape Sequences in String Literals

escape sequences ➤7.3

String literals may contain the same escape sequences as character constants. We've used character escapes in `printf` and `scanf` format strings for some time. For example, we've seen that each `\n` character in the string

```
"Candy\nIs dandy\nBut liquor\nIs quicker.\n  --Ogden Nash\n"
```

causes the cursor to advance to the next line:

```
Candy
Is dandy
But liquor
Is quicker.
  --Ogden Nash
```

Although octal and hexadecimal escapes are also legal in string literals, they're not as common as character escapes.

Be careful when using octal and hexadecimal escape sequences in string literals. An octal escape ends after three digits or with the first non-octal character. For example, the string `"\1234"` contains two characters (`\123` and `4`), and the string `"\189"` contains three characters (`\1`, `8`, and `9`). A hexadecimal escape, on the other hand, isn't limited to three digits; it doesn't end until the first non-hex character. Consider what happens if a string contains the escape `\xfc`, which represents the character *ü* in the Latin1 character set, a common extension of ASCII. The string `"Z\xfcrich"` ("Zürich") has six characters (`Z`, `\xfc`, `r`, `i`, `c`, and `h`), but the string `"\xfcber"` (a failed attempt at "über") has only two (`\xfcbe` and `r`). Most compilers will object to the latter string, since hex escapes are usually limited to the range `\x0`–`\xff`.

Q&A

Continuing a String Literal

If we find that a string literal is too long to fit conveniently on a single line, C allows us to continue it on the next line, provided that we end the first line with a backslash character (`\`). No other characters may follow `\` on the same line, other than the (invisible) new-line character at the end:

```
printf("When you come to a fork in the road, take it.  \
--Yogi Berra");
```

In general, the `\` character can be used to join two or more lines of a program into a single line (a process that the C standard refers to as "splicing"). We'll see more examples of splicing in Section 14.3.

The `\` technique has one drawback: the string must continue at the beginning of the next line, thereby wrecking the program's indented structure. There's a better way to deal with long string literals, thanks to the following rule: when two or more string literals are adjacent (separated only by white space), the compiler will