

obviously a number of improvements needed, ranging from minor tweaks to major enhancements (such as saving the reminders in a file when the program terminates). We'll discuss several improvements in the programming projects at the end of this chapter and in later chapters.

13.6 String Idioms

Functions that manipulate strings are a particularly rich source of idioms. In this section, we'll explore some of the most famous idioms by using them to write the `strlen` and `strcat` functions. You'll never have to write these functions, of course, since they're part of the standard library, but you may have to write functions that are similar.

The concise style I'll use in this section is popular with many C programmers. You should master this style even if you don't plan to use it in your own programs, since you're likely to encounter it in code written by others.

One last note before we get started. If you want to try out any of the versions of `strlen` and `strcat` in this section, be sure to alter the name of the function (changing `strlen` to `my_strlen`, for example). As Section 21.1 explains, we're not allowed to write a function that has the same name as a standard library function, even when we don't include the header to which the function belongs. In fact, all names that begin with `str` and a lower-case letter are reserved (to allow functions to be added to the `<string.h>` header in future versions of the C standard).

Searching for the End of a String

Many string operations require searching for the end of a string. The `strlen` function is a prime example. The following version of `strlen` searches its string argument to find the end, using a variable to keep track of the string's length:

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

    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}
```

As the pointer `s` moves across the string from left to right, the variable `n` keeps track of how many characters have been seen so far. When `s` finally points to a null character, `n` contains the length of the string.

Let's see if we can condense the function. First, we'll move the initialization of `n` to its declaration: