notation instead to declare \texttt{str} in the function heading:

\begin{verbatim}
int c_in_str(const char str[], char ch) // also okay
\end{verbatim}

However, using pointer notation reminds you that the argument doesn't have to be the name of an array but can be some other form of pointer.

The function itself demonstrates a standard way to process the characters in a string:

\begin{verbatim}
while (*str)
{
  statements
  str++;
}
\end{verbatim}

Initially, \texttt{str} points to the first character in the string, so \texttt{*str} represents the first character itself. For example, immediately after the first function call, \texttt{*str} has the value \texttt{m}, the first character in minimum. As long as the character is not the null character (\texttt{\0}), \texttt{*str} is nonzero, so the loop continues. At the end of each loop the expression \texttt{str++} increments the pointer by one byte so that it points to the next character in the string. Eventually, \texttt{str} points to the terminating null character, making \texttt{*str} equal to \texttt{0}, which is the numeric code for the null character. That condition terminates the loop. (Why are string-processing functions ruthless? Because they stop at nothing.)

### Functions That Return Strings

Now suppose you want to write a function that returns a string. Well, a function can’t do that. But it can return the address of a string, and that’s even better. Listing 7.10, for example, defines a function called \texttt{buildstr()} that returns a pointer. This function takes two arguments: a character and a number. Using \texttt{new}, the function creates a string whose length equals the number, and then it initializes each element to the character. Then, it returns a pointer to the new string.

\textbf{Listing 7.10 strgback.cpp}

// strgback.cpp -- a function returning a pointer to char