space characters before it begins storing characters. Not only will this solve the new-line problem, but it also allows us to avoid storing any blanks that precede the part name.

Since read_line is unrelated to the other functions in inventory.c, and since it's potentially reusable in other programs, I've decided to separate it from inventory.c. The prototype for read_line will go in the readline.h header file:

readline.h

```
#ifndef READLINE_H
#define READLINE_H

/****************************************************************
 * read_line: Skips leading white-space characters, then        *
 *            reads the remainder of the input line and         *
 *            stores it in str. Truncates the line if its       *
 *            length exceeds n. Returns the number of           *
 *            characters stored.                                *
 ****************************************************************/
int read_line(char str[], int n);

#endif
```

We'll put the definition of read_line in the readline.c file:

readline.c

```
#include <ctype.h>
#include <stdio.h>
#include "readline.h"

int read_line(char str[], int n)
{
  int ch, i = 0;

  while (isspace(ch = getchar()))
    ;
  while (ch != '\n' && ch != EOF) {
    if (i < n)
      str[i++] = ch;
    ch = getchar();
  }
  str[i] = '\0';
  return i;
}
```

The expression

```
isspace(ch = getchar())
```

controls the first while statement. This expression calls getchar to read a character, stores the character into ch, and then uses the isspace function to test whether ch is a white-space character. If not, the loop terminates with ch containing a character that's not white space. Section 15.3 explains why ch has type int instead of char and why it's good to test for EOF.