```
int read_char(void)
{
  int ch = getchar();

  if (ch == '\n' || ch == '\t')
    return ' ';
  return ch;
}

void read_word(char *word, int len)
{
  int ch, pos = 0;

  while ((ch = read_char()) == ' ')
    ;
  while (ch != ' ' && ch != EOF) {
    if (pos < len)
      word[pos++] = ch;
    ch = read_char();
  }
  word[pos] = '\0';
}
```

Before we discuss read_word, a couple of comments are in order concerning the use of getchar in the read_char function. First, getchar returns an int value instead of a char value; that's why the variable ch in read_char is declared to have type int and why the return type of read_char is int. Also, getchar returns the value EOF when it's unable to continue reading (usually because it has reached the end of the input file).

EOF macro ➤22.4

read_word consists of two loops. The first loop skips over spaces, stopping at the first nonblank character. (EOF isn't a blank, so the loop stops if it reaches the end of the input file.) The second loop reads characters until encountering a space or EOF. The body of the loop stores the characters in word until reaching the len limit. After that, the loop continues reading characters but doesn't store them. The final statement in read_word ends the word with a null character, thereby making it a string. If read_word encounters EOF before finding a nonblank character, pos will be 0 at the end, making word an empty string.

The only file left is line.c, which supplies definitions of the functions declared in the line.h file. line.c will also need variables to keep track of the state of the line buffer. One variable, line, will store the characters in the current line. Strictly speaking, line is the only variable we need. For speed and convenience, however, we'll use two other variables: line_len (the number of characters in the current line) and num_words (the number of words in the current line).

Here's the line.c file:

line.c

```
#include <stdio.h>
#include <string.h>
#include "line.h"
```