justify.c

```
/* Formats a file of text */

#include <string.h>
#include "line.h"
#include "word.h"

#define MAX_WORD_LEN 20

int main(void)
{
  char word[MAX_WORD_LEN+2];
  int word_len;

  clear_line();
  for (;;) {
    read_word(word, MAX_WORD_LEN+1);
    word_len = strlen(word);
    if (word_len == 0) {
      flush_line();
      return 0;
    }
    if (word_len > MAX_WORD_LEN)
      word[MAX_WORD_LEN] = '*';
    if (word_len + 1 > space_remaining()) {
      write_line();
      clear_line();
    }
    add_word(word);
  }
}
```

Including both `line.h` and `word.h` gives the compiler access to the function prototypes in both files as it compiles `justify.c`.

`main` uses a trick to handle words that exceed 20 characters. When it calls `read_word`, `main` tells it to truncate any word that exceeds 21 characters. After `read_word` returns, `main` checks whether `word` contains a string that's longer than 20 characters. If so, the word that was read must have been at least 21 characters long (before truncation), so `main` replaces the word's 21st character by an asterisk.

Now it's time to write `word.c`. Although the `word.h` header file has a prototype for only one function, `read_word`, we can put additional functions in `word.c` if we need to. As it turns out, `read_word` is easier to write if we add a small "helper" function, `read_char`. We'll assign `read_char` the task of reading a single character and, if it's a new-line character or tab, converting it to a space. Having `read_word` call `read_char` instead of `getchar` solves the problem of treating new-line characters and tabs as spaces.

Here's the `word.c` file:

word.c

```
#include <stdio.h>
#include "word.h"
```