```
 enough   to  displace         assembly   language,
   yet sufficiently   abstract   and fluent       to describe
  algorithms   and     interactions    in a   wide    variety
of   environments.
                     --      Dennis     M.         Ritchie
```

To run the program from a UNIX or Windows prompt, we'd enter the command

```
justify <quote
```

input redirection ➤22.1

The < symbol informs the operating system that `justify` will read from the file `quote` instead of accepting input from the keyboard. This feature, supported by UNIX, Windows, and other operating systems, is called ***input redirection***. When given the `quote` file as input, the `justify` program will produce the following output:

```
C is quirky,  flawed,  and  an  enormous  success.  Although
accidents of history surely helped, it evidently satisfied a
need for a system implementation language  efficient  enough
to displace assembly language, yet sufficiently abstract and
fluent to describe algorithms and  interactions  in  a  wide
variety of environments. -- Dennis M. Ritchie
```

output redirection ➤22.1

The output of `justify` will normally appear on the screen, but we can save it in a file by using ***output redirection***:

```
justify <quote >newquote
```

The output of `justify` will go into the file `newquote`.

In general, `justify`'s output should be identical to its input, except that extra spaces and blank lines are deleted, and lines are filled and justified. "Filling" a line means adding words until one more word would cause the line to overflow. "Justifying" a line means adding extra spaces between words so that each line has exactly the same length (60 characters). Justification must be done so that the space between words in a line is equal (or as nearly equal as possible). The last line of the output won't be justified.

We'll assume that no word is longer than 20 characters. (A punctuation mark is considered part of the word to which it is adjacent.) That's a bit restrictive, of course, but once the program is written and debugged we can easily increase this limit to the point that it would virtually never be exceeded. If the program encounters a longer word, it must ignore all characters after the first 20, replacing them with a single asterisk. For example, the word

```
antidisestablishmentarianism
```

would be printed as

```
antidisestablishment*
```