Unit 12: Writing Large Programs

Although some C programs are small enough to be put in a single file, most are not. Programs that consist of more than one file are the rule rather than the exception. In this unit we cover the separation of a project into different files, usually header files and source files. Header files contain information to be shared among source files, and source files contain definitions of functions and external variables. Relevant for this unit are the following Book Sections:

- Section 15.1: discusses source files
- Section 15.2: covers header files
- Section 15.3: describes how to divide a program into source files and header files
- Section 15.4: shows how to "build" a program that consists of more than one file and how to rebuild a program after part of it has changed. Here Makefiles are introduced as well, as we shall use them with Linux.

Project u12: Text Justification

The justify.c program justifies lines by inserting extra spaces between words. The way the `write_line` function currently works, the words closer to the end of a line tend to have slightly wider gaps between them than the words at the beginning. For example, the words closer to the end might have three spaces between them, while the words closer to the beginning might be seperated by only two spaces. The justify program consists of the files justify.c, word.h, word.c, line.h, line.c, and Makefile. The program can be compiled with 'make' or 'make justify'.
Implement the following two modifications of the justify.c program:

- Improve the program by having `write_line` alternate between putting the larger gaps at the beginning of the line and putting them at the end of the line.
- Modify the program by having the `read_word` function (instead of main) store the '*'-character at the end of a word that's been truncated.
- Modify the Makefile to compile `u12_justifymod.c`, link `u12_justifymod.o`, and produce the executable `u12_justifymod`. By running 'make' the executable `u12_justifymod` must be created.

Here are examples of the same text, first justified with the provided justify.c program:

```
Vettel got all the hard work done in the first corner as  he
made a fantastic start and beat Webber on the inside of turn
one. After holding off the advances of  his  team-mate  over
the next two laps,  the  Red  Bulls  almost  adopted  cruise
control to finish comfortably  ahead  of  third-placed  Nico
Rosberg in the Mercedes.
```

and justified with the new alternating method of spacing:

```
Vettel  got all the hard work done in the first corner as he
made a fantastic start and beat Webber on the inside of turn
one.  After  holding  off the advances of his team-mate over
the next two laps,  the  Red  Bulls  almost  adopted  cruise
control  to  finish  comfortably  ahead of third-placed Nico
Rosberg in the Mercedes.
```

The difference can be seen in the very first line after the first word. The first word “Vettel” is followed by one space with the original justification, and right before the last word of the same line we have two spaces. Whereas with the new method, we have two spaces following the word “Vettel” whereas we have only one space at the end of the line right before “he”. The 2nd, 4th, etc. lines are equal, only the 1st, 3rd, etc. lines are (possibly) different. The lines do not differ if we always have exactly the same number of space between all words of the same line, otherwise we see the difference in justification.
Hint: In the implementation use a variable to alter between lines of (even/odd) number, and perform the reverse form of justification on every other line, starting with the very first one. A straight-forward implemetnation is to store the differently justified line in an auxiliary array and print it afterwards, whereas the original justification remains unchanged for every other line.
Refer to the provided test-cases and compare the output of your program with the provided output of the test-cases. Note that both modifications are represented by the test-cases. Your program should behave exactly like the test-cases before you hand it in. Hand in a zip file containing all the files of the project (ensure that the zip file contains only the files, but *not* a directory).