School of Computing and Information Systems

comp10002 Foundations of Algorithms Semester 2, 2019

Assignment 1

In this project you will demonstrate your understanding of arrays, strings, and functions. You may also use typedefs and structs if you wish (see Chapter 8) – and will probably find the program easier to assemble if you do – but you are not required to use them in order to obtain full marks. You should not make any use of malloc() (Chapter 10) or file operations (Chapter 11) in this project.

Text Formatting

In this assignment we are going to journey back to the 1970s, and implement a simple text formatting tool based entirely on fixed-width terminal fonts and character-based output. The formatting "commands" will be indicated by input lines that start with a period, ".", with the command indicated by the letter in the next character position. For example, the characters ".p" in the first two positions of a line are (in this simple language) the signal to start a new paragraph in the output.

Stage 1 – Filling Lines (8/15 marks)

The first version of your program should:

- read text input from stdin (you may assume that input lines will be at most 999 characters long);
- completely discard the content of all lines that commence with a period character;
- replace all instances of (multiple) whitespace characters (blanks, tabs, newlines) by single blanks;
- re-insert newline characters in a greedy line-by-line manner, so that each output line emitted is as long as possible, but not longer than 50 characters;
- with the exception that if there is an input token/word that is greater than 50 characters, it goes on an output line by itself, and that line can be greater than 50 characters long.

The lines in the output should be arranged so that the left margin has a default of 4 initial blanks, and hence that the first non-blank character in each line is in column 4 (counting the columns from zero), and the last character in any line (except as noted in connection with very long tokens) is never beyond column 53 in each output line. For example, if the file test0.txt contains

```
one two three four five six seven eight

nine
ten eleven twelve thirteen fourteen
                        fifteen            sixteen
                        seventeen
 eighteen nineteen twenty.

sixteen
```

```
    .b
    Enough of the numbers already, what about:
    101, 102, 103, 104, 105, 106, 107, 108, and 109?
    Plus 110, 111, 112, 113, 114 and all of 115 116 117 118 and 119.
```

the output should be

```
    0----5---10---15---20---25---30---35---40---45---50---55---60
    mac: ass1-soln < test0.txt
        one two three four five six seven eight nine ten
        eleven twelve thirteen fourteen fifteen sixteen
        seventeen eighteen nineteen twenty. Enough of the
        numbers already, what about: 101, 102, 103, 104,
        105, 106, 107, 108, and 109? Plus 110, 111, 112,
        113, 114 and all of 115 116 117 118 and 119.
    mac:
    0----5---10---15---20---25---30---35---40---45---50---55---60
```

where the two "rulers" are *not* part of the output and are provided here to help you count character positions. (But, hint hint, they *were* part of the output while the sample solution was being debugged.) See the FAQ page linked from the LMS for more example input files and the output that is expected. Note that the values 4 and 50 should be held in variables assigned via initial #define starting points – you will need to be able to change their values in Stage 2.

The getword() function that was discussed in class may look like a tempting starting point. But you need to be able to examine the first character of each line without necessarily consuming it, and so a main loop that reads and processes lines is probably a better bet. You can then break that line up into tokens as required. You should also pay close attention to the item on the FAQ page that discusses the issues that may arise in connection with newline characters.

Note also that while it is possible to achieve the limited functionality required in this stage with a program that is little more than a "while (getchar())" loop, you will only receive 8/8 for a Stage 1 submission if your program shows that you have planned it in a way that allows the functionality required for Stage 2 and Stage 3 to be added without needing to completely rewrite it.

Stage 2 – Processing Commands (13/15 marks)

Once you have the Stage 1 program operational (and submitted, so that it is "on the record"), extend your program so that it recognizes and acts upon the following commands, where the "." that indicates a formatting command always appears in the first position of an input line, straight after a newline character:

- .b – break the current line, so that the next input token starts at the beginning of the next line;
- .p – leave a blank line and start a new paragraph without altering the margins, with the next input

  token starting the new paragraph;

- .l *nn* – alter the left margin from its current value (default initial value of 4) to the new value *nn*, and start a new paragraph;

- .w *nn* – alter the width of each line from its current value (default initial value of 50) to the new value *nn*, and start a new paragraph.

Stage 3 – Adding Structure (15/15 marks)

Now add two further commands:

- .c – take the remaining contents of this line and center them within the current output width. If the remaining contents cannot fit within the current output width, then it should be placed so that it starts at the left margin and overflows beyond the current width. When there is an odd number of spaces to be assigned, the rounded-down half should be at the beginning of the line, and the rounded-up half at the end of the line.
- .h *nn* – take the remainder of the contents of this line and use them as a section heading at the level indicated by *nn*. The Section/subsection/subsubsection number should be printed, and then the complete heading on a single line, even if it is longer that the current width. Headings at level *nn* reset the numbering for headings level *nn* + 1, *nn* + 2, and so on; and (to avoid non-computing people from being confused) all headings start their counting from one. At most five levels of headings may occur, counting from *nn* = 1 (the top-level heading) to *nn* = 5. All headings are always preceded and followed by a paragraph boundary, and level-1 headings are also preceded by a full-width line of "-" characters.