Software Engineering 265
Software Development Methods
Spring 2013

Assignment 1

Due: Tuesday, February 5th, 9:00 am by submission via Subversion
(no late submissions accepted)

Programming environment

For this assignment you must ensure your work executes correctly on the Linux machines in ELW B215. You are welcome to work on your own laptops and desktops; if you do this, give yourself a few days before the due date to iron out any bugs in the C program you have uploaded to the BSEng machines. (Bugs in this kind of programming tend to be platform specific, and something that works perfectly at home may end up crashing on a different hardware configuration.)

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Zastre).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools will be used to examine submitted programs.)

Objectives of this assignment

- Understand a problem description, along with the role used by sample input and output for providing such a description.
- Use the C programming language to write the first implementation of a SENG 265 text formatter named "s265fmt" (and do this without using dynamic memory).
- Use Subversion to manage changes in your source code and annotate the evolution of your solution with "messages" provided during commits.
- Test your code against the ten provided test cases.

This assignment: "s265fmt.c"

You are to write a C program that inputs lines of text files, reads formatting options contained within those text files, and then outputs the text where each line has the appropriate width, indenting, etc. as indicated by the formatting options. For example, here is one such file contained in the "tests" subdirectory at /home/zastre/seng265/assign1/tests ("in04.txt"):

```
?pgwdth 30
While there    are enough characters   here to
fill
   at least one line, there is
plenty
of
            white space that needs to be
eliminated
from the original
         text file.
```

This particular file contains one formatting option ("?pgwdth 30") which indicates that the following text must be formatted such that each line contains at most 30 characters. The formatting program also concatenates the lines of a paragraph together in order to eliminate unnecessary white space. The resulting output (stored in "out04.txt") looks like this:

```
While there are enough
characters here to fill at
least one line, there is
plenty of white space that
needs to be eliminated from
the original text file.
```

With your completed "s265fmt" program, the input would be transformed into the output via the following UNIX command:

```
% ./s265fmt /home/zastre/seng265/assign1/tests/in04.txt > ./out04.txt
```

where the file "out04.txt" would be placed in your current directory. To compare the output produced by s265fmt with what is expected, you can use the Unix "diff" command as shown below (assuming you're still in the same directory as when you executed the command above):

```
% diff -b /home/zastre/seng265/assign1/tests/out04.txt ./out04.txt
```

Note that "diff" is much more reliable than your own eyes!

For this first assignment there are only four formatting commands:

- ?pgwdth *width*: Each line following the command will be formatted such that there is never more than *width* characters in each line. The original whitespace in the input text need not necessarily be preserved (i.e., single

spaces are used to separate words in the output). If this command does not appear in the input file, then the input text is not transformed in the output.
- ?mrgn *left*: Each line following the command will be indented *left* spaces from the left-hand margin. Note that this indentation must be included in the page width. If this command does not appear in the input file, then the value of *left* is 0 (zero).
- ?fmt [off | on]: This is used to turn formatting on and off. If the command appears with "off", then all text below the command up to the next "?fmt" command is output without being formatted. If the command appears with "on", then all text below the command up to the next "?fmt" command is output with as many words as will fill the given page width. (It may be that there is no "?fmt" command following, in which case the choice of "on" or "off" is used for the remaining input text.)

There is some default behavior expected. (Some details from the previous bullet points are repeated below.)

- If no "?pgwdth" command appears, the default mode is "?fmt off". In this case all "?mrgn" commands are ignored.
- If a "?pgwdth" command appears, the default mode is "?fmt on".
- For this first assignment, there will only ever be one "?pgwdth" or "?mrgn" command in an input file, and these will appear at the top of the file.
- For this first assignment, you can assume that all test files will have page widths that are much greater than left margins (e.g., there will not be a combination such no as "?pgwdth 30" and "?mrgn 40").
- There is no limit to the number of "?fmt" commands that can appear within an input file.

Exercises for this assignment

1. If you have not already done so, ensure your Subversion project is checked out from the repository. Within the project create an "assign1" subdirectory. Ensure all directories and program files you create are placed under Subversion control. (You need not add the test directory to subversion control unless you wish to do so.)

2. Write your program. Amongst other tasks you will need to:
   - read text input from a file, line by line
   - write output to the terminal
   - extract substrings from lines produced when reading a file
   - create and use arrays in a non-trivial array.
   - use the "-ansi" flag when compiling to ensure your code is ANSI compliant.

3. Do not use “malloc”, “calloc” or any of the dynamic memory functions. For this assignment you can assume that the longest input line will have 132 characters, and no input file will have more than 300 lines.

4. Keep all of your code in one file for this assignment. In later assignments we will use separable compilation features of C.

5. Use the test files to guide your implementation effort. Start with simple cases (such as those given in this writeup). Refrain from writing the program all at once, and budget time to anticipate when “things go wrong”.

6. For this assignment you can assume all test inputs will be well-formed (i.e., our teaching assistant will not test your submission for handling of input or for arguments containing errors). Later assignments will specify error-handling as part of the assignment.

What you must submit

- A single C source file named “s265fmt.c” within your subversion repository containing a solution to Assignment #1.
- **No dynamic memory-allocation routines are to be used for Assignment #1.**

Evaluation

Students will demonstrate their work to a member of the course’s teaching team. Sign-up sheets for demos will be provided a few days before the due-date; each demo will require from 10 to 15 minutes.

Our grading scheme is relatively simple.

- “A” grade: An exceptional submission demonstrating creativity and initiative. “s265fmt” runs without any problems. The program is clearly written and uses functions appropriately (i.e., is well structured).
- “B” grade: A submission completing the requirements of the assignment. “s265fmt” runs without any problems. The program is clearly written.
- “C” grade: A submission completing most of the requirements of the assignment. “s265fmt” runs with some problems.
- “D” grade: A serious attempt at completing requirements for the assignment. “s265fmt” runs with quite a few problems.
- “F” grade: Either no submission given, or submission represents very little work.