**Software Engineering 265**
**Software Development Methods**
**Summer 2020**

*Assignment 1*

Due: Monday, June 15, 11:55 pm by "git push"
(Late submissions **not** accepted)

## Programming environment

For this assignment you must ensure your work executes correctly on the virtual machines you installed as part of Assignment #0 (which I have taken to calling `Senjhalla`). This is our "reference platform". This same environment will also be used by the course instructor and the rest of the teaching team when evaluating submitted work from students.

Any programming done outside of this reference platform might not work correctly. Such work may receive a failing grade.

## 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 work.)

## Objectives of this assignment

- Understand a problem description, along with the role played by sample input and output for providing such a description.
- Use the C programming language to write the first implementation of a text formatter named `senjify1` (and do this without using dynamic memory).
- Use `git` 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: `senjify1.c`**

You are to write a C program that inputs lines from `stdin`, reads formatting options contained within that stream of characters, and then outputs to `stdout` the text where each line output has the appropriate width, indenting, etc. as indicated by the formatting commands. Here is one file `/home/zastre/seng265/assign1/in04.txt` contained on the SENG servers.

```
{{ 30 }}
This particular document has
   words all over             the place
with odd
            formatting.
In fact there
                  is a lot of horizontal
white space       which may be      eliminated
      as
         part
             of
                processing
the original file. But do notice there there is
no vertical
white
          space
that
         is,
no
          blank
lines
         within
the
         text.
```

This file contains a single formatting option "`{{ 30 }}`" indicating the following text must be formatted such that each line contains at most 30 characters. The formatting program also eliminates unnecessary white space between words (i.e., regardless of inter-word spacing within the input file, words with the same output line are separated by a single space).

The resulting expected output looks like the following (and you can find this on the SENG servers at `/home/zastre/seng265/assign1/out04.txt`):

```
This particular document has
words all over the place with
odd formatting. In fact there
is a lot of horizontal white
space which may be eliminated
as part of processing the
original file. But do notice
there there is no vertical
white space that is, no blank
lines within the text.
```

Assuming you have copied the tests inputs and outputs from the SENG servers (i.e., from `/home/zastre/seng265/assign1`) into `Senjhalla`, and that this subdirectory is named `tests/` and is located within the directory you are using for writing `senjify1`, then the input would be transformed into the output via the following command:

```
% cat tests/in04.txt | ./senjify1 > testout04.txt
```

and the file `testout04.txt` produced by re-directing the output from the program appears here in the same directory as `senjify1`. To compare the output produced by `senjify1` 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 testout04.txt tests/out04.txt
```

In you wish, you can combine both the formatting and the testing into one longer command by using only Unix pipes:

```
% cat tests/in04.txt | ./senjify1 | diff – tests/out04.txt
```

Note that `diff` is much (much!) more reliable than using only your eyes to look for discrepancies. In fact, discrepancies such as additional blank spaces at the ends of lines in your output are invisible to you without the use of `diff`.

For this first assignment there are only five formatting commands:

- `{{` *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). For example, `{{ 50 }}` would set the formatted line width to 50 characters wide. If this command does not appear in the input file, then the input text is not transformed in the output.

- `{{ >`*indent*`}}`: Each line following the command will be indented by *indent* spaces from the left-hand margin. Note this indentation must be included within the current line width. For example, `{{ >3 }}` would set the line indent to three spaces. If this command does not appear in the input file, then the value of *indent* is 0 (zero).
- `{{ on }}` and `{{ off }}`: These are used to turn formatting on and off. If `{{ on }}` appears then all text below the command up to the next `{{ off }}` command is formatted given the line width and line indent. If `{{ off }}` appears then all text below the command up to the next `{{ on }}` command is output with no formatting. The `{{ off }}` command ***does not reset*** the line width or line indent settings. If no line width is provided when formatting is turned on, then the default line width is 80.
- `{{ ! }}`: This toggles the formatting mode. If it was `on` then the formatting mode is turned off; if it was `off` then the formatting mode is turned on.

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

- Commands are always located at the start of a line; there will only ever be one command on such a line. Any command that is not at the start of a line is to be considered as regular text.
- Processing of a file begins with the formatting mode set to `off`. In this case all indenting subsequent commands are ignored until there is an `on` command.
- If no line width has been provided when formatting is turned on, then the default line width is 80.
- If a `{{` *width* `}}` command appears, the formatting mode is automatically turned to `on`.
- For this first assignment, you can assume all test files have line widths much greater than left margins (*e.g.*, there will never be a combination such as `{{ 30 }}` and `{{ >40 }}`).
- There is no limit to the number of `{{ on }}`, `{{ off }}`, and `{{ ! }}` commands that can appear within an input file.
- All input to `senjify1` is from `stdin`, and all output is to `stdout`. ***You must not hardcode filenames in your submitted code!***

**Exercises for this assignment**

1. Write your program the `a1` directory within your local repo for SENG 265. Amongst other tasks you will need to:
   - read text input from a `stdin`, line by line;
   - write output to `stdout`;
   - extract substrings from lines produced when reading a file;
   - create and use arrays in a non-trivial array.

- use the `-std=c99` flag when compiling to ensure your code is compliant with the 1999 C programming language standard.

2. 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 500 lines.

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

4. Use the test files to guide your implementation effort. Start with simple cases (such as those given in this writeup). In general, lower-numbered tests are simpler than higher-numbered tests. Refrain from writing the program all at once, and budget time to anticipate for when "things go wrong".

5. For this assignment you can assume all test inputs will be well-formed (i.e., the teaching team will not test your submission for handling of errors in the input). Later assignments may error-handling as part of the assignment.

6. Use `git add` and `git commit` appropriately. While you are not required to use `git push` during the development of your program, you **must** use `git push` in order to submit your assignment.

**What you must submit**

- A single C source file named ~~`sengjify1.c`~~ `senjify1.c` within your git repository (and in the `a1` directory) containing a solution to Assignment #1. Ensure your work is **committed** to your local repository **and pushed** to the remote **before the due date/time**. (You may keep extra files that you have used during development within the repository.)
- No dynamic memory-allocation routines are permitted for Assignment #1 (i.e., do not use anything in the `malloc()` family of functions).

**Evaluation**

Our grading scheme is relatively simple.

- "A" grade: A submission completing the requirements of the assignment. Submitted code is well-structured and very clearly written. Global variables are kept to a minimum. `senjify1` runs without any problems; that is, all tests pass and therefore no extraneous output is produced.

- “B” grade: A submission completing the requirements of the assignment. `senjify1` runs without any problems; that is, all tests pass and therefore no extraneous output is produced. The program is clearly written.
- “C” grade: A submission completing most of the requirements of the assignment. `senjify1` runs with some problems.
- “D” grade: A serious attempt at completing requirements for the assignment. `senjify1` runs with quite a few problems; even through the program runs, it may be that no tests pass.
- “F” grade: Either no submission given, or submission represents very little work.