Homework 3 - MPCS 51040

(last modified: November 11, 2014)

Issued: November 11, 2014

1 General Instructions

1.1 Compiling

Your code must compile with `gcc -std=c11 -Wall -pedantic`. There should be no warnings or errors when compiling with gcc-4.8 (as installed on `linux.cs.uchicago.edu`). You can add `-Werror` to force the compiler to treat any warning as an error, and stop compilation immeditately.

1.2 Handing in

To hand in your homework, you need to commit all requested files (with correct filenames!) to your personal subversion repository. You should have access to a repository named yourcnetid-mpcs51040-aut-14.

Make a subdirectory called 'homework/hw3' and place your files under that directory. Don't forget to commit your files! You can check on `http://phoenixforge.cs.uchicago.edu` to make sure all files were committed to the repository correctly.

If you can cannot login to `http://phoenixforge.cs.uchicago.edu` or have issues accessing your repository, please contact `techstaff@cs.uchicago.edu`. Please cc `dries@uchicago.edu`.

The deadline for this homework is 5:30pm, November 25, 2014. To grade the homework, the contents of your repository at exactly the deadline will be considered. Late policy: Changes made after the deadline will not be taken into account. No extensions are given.

1.3 Code samples

This document, and any file you might need to complete the homework can be found the subversion repository `https://wooldridge.cs.uchicago.edu/svn/mpcs51040-aut-14`.

1.4 Grading

Your code will be graded based on the following points (in order of descending importance):

- Correctness of the C code: there should be no compiler errors or warnings when compiling as described in 1.1. There should be no memory leaks or other problems (such as those detected by valgrind).
- Correctness of the solution. Your code should implement the required functionality, as specified in this document.
- Code documentation. Properly documented code will help understand and grade your work.
- Code quality: your code should be easy to read and follow accepted good practices (avoid code duplication, use functions to structure your program, ...). This includes writing portable code (which will work on both 32 bit and 64 bit systems).
- Efficiency: your code should not use more resources (time or space) than needed.

Some items might be marked as 'optional' or 'extra credit'. When correctly completing these tasks, the points obtained could go towards mistakes made elsewhere in the homework, possibly raising your grade.

2 Assignment

2.1 Problem Description

The goal of this homework is to become familiar with creating custom data structures based on linked lists.

2.2 Task 1

In a first step, you will need to read the input file specified on the command line and store it in memory for processing. A specific structure in memory is required, based on how we will be analyzing the text later on.

In particular, your data structure needs to allow efficiently traversing the text by word, sentence and paragraph. For the purpose of this homework, words are seperated by whitespace or punctuation tokens, sentences by '?', '!' or '.' and paragraphs by one or more blank lines.

You do not need to worry about how to read the text or how to split it into components. The code provided already does this.

You need to modify the provided program so that it:

- Stores the text into a special purpose data structure in memory (further described below).
- Provides correct answers, using the data stored in your in-memory data structure, for the questions outlined below.

You are allowed to modify the program in any way you want except for the routine parsing the text. In addition, you cannot modify the output of the program.

Your datastructure (60 points) needs to be able to:

- Iterate over the text by word, sentence or paragraph in forward direction without having to skip over data that has not been requested. For example, when iterating over the document by paragraph, each node in the list should point to the beginning of a paragraph. There should be no need to iterate over sentences or words in order to count the number of paragraphs (and likewise for sentence).
- Data should not be duplicated in memory. Each word should only be stored once. This means that you should not make separate linked lists storing full paragraphs or sentences.
- It should be possible to, starting from a paragraph, list the sentences making up that paragraph. Likewise, for a given sentence, it should be possible to list the words of that paragraph.
- Order (of sentences, words and paragraphs) needs to be preserved.
- While your data structure does not need to handle any other information than words, sentences and paragraphs, the code for your data structure should be in a separate file (called `ts.c` and `ts.h`), as we have done for `arb_int` and the linked list designed during class. In particular, operations to add information to the data structure, iterate over the data, and to destroy the data structure should be in the `ts.c` and `ts.h` files.

  The code in the main program will use these functions to answer the questions below.

  Questions:

1. (10 points) How many words, sentences and paragraphs are there in the document? Find the answer by walking your data structure by word, sentence and paragraph.
2. (25 points) How many times does a consecutive word, sentence or paragraph start with the same letter? For example, in the following text:

```
This is a test. There will be another test.

Or maybe not.
```

There is one instance of a word starting with the same letter (test - There), one instance of a sentence starting with the same letter and no paragraphs starting with the same letter.

Case does not matter.

3. (10 points) What is the minimum, maximum and average number of:
   - Words in a sentence? (for the whole document)
   - Sentences in a paragraph?
4. (20 points) [extra credit] Remove every other sentence, paragraph and word and recalculate the answers to the questions above. For example, the following text:

   ```
   This is an example sentence. And also a paragraph.

   Well, the paragraph starts here. This is a short paragraph.
   It used to be at least.

   This is the last paragraph. Which had 3 sentences. I wonder how many will remain.
   ```

   becomes (punctuation retained for clarity):

   ```
   This an sentence.

   This the paragraph. I how will.
   ```

Before the program ends, memory needs to be freed so that there are no memory leaks. As you know, you can check for memory leaks using the 'valgrind' tool.

2.3 Task 2 (25 points)

Create a file called `README.TXT` which briefly describes how you implemented the requested functionality. Please be clear and complete; In case of accidental mistakes or bugs in your program, this file can help the person grading your work understand your intentions. **Make sure that you commit the homework to the exact directory requested in this document, using the exact filenames.**

Do not underestimate the importance of the README file!