

Homework 7 - Final Project

Computer Science I (20-CS-121) Summer 2007

Monkeys and Typewriters

Assigned: Monday, 16 July, 2007

Due: Tuesday, 24 July, 2007 at NOON

1 Before You Even Look at the Following...

Take a deep breath and have confidence in what you have done thus far. The following assignment, although it may seem difficult at first, has an extremely simple design *and* solution. The more one spends thinking about this (which I try to force you to do through submission of design documents), the easier the solution becomes. In fact, with perhaps a few days worth of thought and note taking, one should be able to design and implement a solution in one good afternoon.

This is *not* an assignment where you want to skip the steps of good software design! (I specifically picked this problem because of this!) Before you even begin to design anything, be **absolutely sure** you understand **exactly** what the problem is. Then make sure **you** can perform what's asked, with paper & pencil. After that, **and only after that**, start designing a solution/program to do *what you already can do*. Once your design is complete, start implementing.

Do *NOT* try to sit down and write this program *all at once*, even if you have a complete, well-thought-out design! Each function you write should be written incrementally with a driver/test program to make sure each function does *exactly* what you designed it to do.

2 Problem Description

You've probably heard the saying that if you sit 1,000,000 monkeys in front of 1,000,000 typewriters, eventually one of them will type out the complete works of Shakespeare (assuming we never run out of paper...or monkeys). You are tasked with writing a program that does *something* like this, which will generate random text with the goal of (re)producing some well known literary work (or something close).

Consider the following modification to the problem described above...Rather than generating text randomly (or pseudo/monkey randomly), we'll assume that the monkeys have "some experience", and generate text "in the style of" some previous text they've read.

Specifically, you'll write a program that reads in a text file, character-by-character, and computes the frequency of each character read *in the context of the previous 2 characters*. Once the file has been completely read and all frequencies have been calculated, you'll use this information to output a piece of text, again character-by-character, employing these frequencies. (Treat the first character to be output as preceded by two spaces, and the second as preceded by a space and then the first character generated.) So, if your program outputs the two letters "tr", and if you've calculated from the original text that 50% of the time this was followed by "a", 25% of the time it was followed by "e", 15% by "i", and 10% by "o", you would then generate one of "a", "e", "i", or "o" based on these probabilities.

Note that your program should account for characters "a" - "z", "A" - "Z", "0" - "9", standard punctuation (spaces, periods, commas, colons, semicolons, single & double quotes, and parentheses), and new-line characters.

3 Details of the Assignment and Provided Code

This assignment will be done in pieces, starting with some design documents outlining your solution as you drill successively deeper into the problem. After that, you'll submit your complete design documents and program source code on the last day of class (24 Tuesday 2007).

Note that I'll also be providing hints along the way, in both the form of design and code. Specifically, I will provide the following (probably on Wednesday):

- Code for reading in a file character-by-character, which I will cover extensively.
- A general design for how to calculate and represent the frequencies mentioned above.
- Code to calculate the next letter to be output, given the previous two letters output and the frequencies you calculated (well...not quite, but I'll give you something pretty close!)

I want you, however, to first think about the problem and come up with your own understanding of the problem and a *preliminary* design before I provide any such hints.

Note that since this program will obviously make use of `rand()`, you should seed the random number generator and report whatever seed you use.

4 Due Dates

The following items are due on the dates specified. . .

1. **Wednesday, 18 July 2007:** A high-level design document that outlines the steps required to solve this problem. I'm not looking for extreme detail here, but simply an overview "at 30,000 feet" of what your solution will do and how it does it. *Due at the start of class on Wednesday!*
2. **Friday, 20 July 2007:** A *more detailed* design document outlining *all of the details* of your solution, including how you plan to represent all of the information required, how you plan to extract the frequencies from the input text file, and how you plan to generate the output text using the frequencies. *Due at the start of class on Friday!*
3. **Tuesday, 24 July 2007 by NOON:** Complete copies of your program source code and design documents, both electronically and on paper (in my mailbox).

5 How to Submit Your Assignment

Your program *must*. . .

- Compile without errors or warnings
- Employ a good design, *which you clearly document*
- Make *extensive* use of a good C++ style (as discussed in class)
- Perform the specified task

In addition, you *must* submit a design document, outlining your solution to this problem as you follow the steps of good software design. You must include a breakdown of your solution in pseudo-code. Once you are finished, you are to either

1. Email a copy of your source code and design documents to me at ryan.flannery@gmail.com
2. Submit a copy of your source code and design documents to me through BlackBoard's Digital Drop-box

Either one must be done *before noon on Tuesday* and put a print-out of your source code and design docs in my mailbox, **STAPLED!**, before that time. (You could, of course, submit the print out in class on Monday if you finish early.)

6 How Your Work will be Graded

Remember that this assignment is worth 70 points, or 7% of your final grade. Your submission will be graded upon the normal spread for programs, as outlined in the syllabus (with a *small* fudge-factor). . .

- 20% (or 14 points / 1.4% of your final grade) on style. This includes *all* of the good coding style guidelines we covered in class. Functions should be relatively short, and have a single, clear purpose. *Global variables should be avoided like the plague.* Good naming and commenting are critical.
- 20% (or 14 points / 1.4% of your final grade) on design. These points will come from what you submit on Wednesday, Friday, with your final submission, *and how well your code implements the design you outlined!* (I.e. your design should be evident from your code!)

- 60% (or 42 points / 4.2% of your final grade) on correctness. Whatever you submit, I should be able to compile and run, and it *should* perform the task specified. If you can't quite finish the assignment, submit as much as you have, clearly documenting where your problems are. What you submit should *still* compile, run, and do as much as possible.
- **Note** that the deadline is, in fact, a *deadline!* Late submissions (be they digital or paper) will lose 10 points for every hour late.