Kristian Snyder

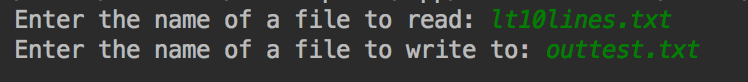
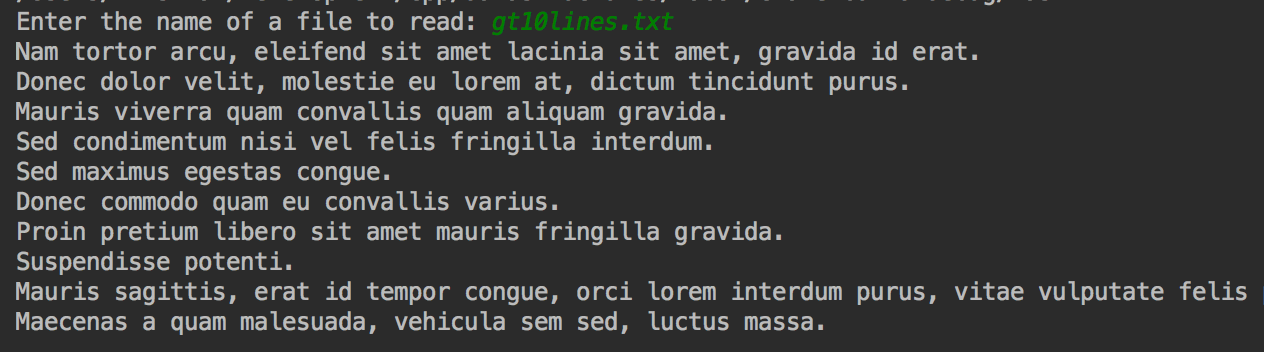
Data Structures CS2021 Section 001

Prof. Charles Zimmer, Prof. Karthik Vadambacheri

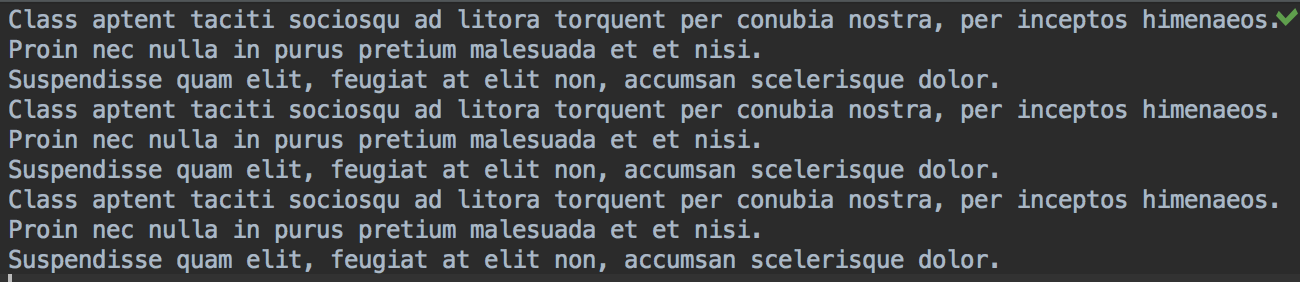
Lab 02

**Central Concepts:**

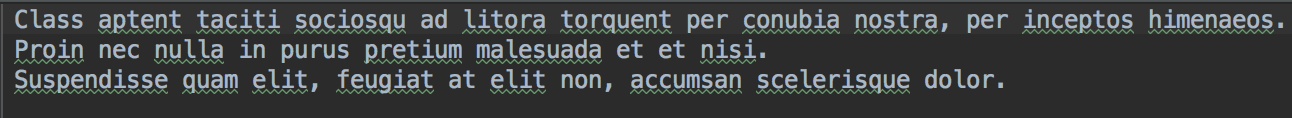
Two concepts with relatively common use were explored in this lab: file access and class design and implementation.

Knowing both how to manipulate files in a safe and sane manner is important when writing many types of applications to prevent conflicts, unintentional writes, or large amounts of disk I/O that can reduce the life of the user’s hard drive. Basic error checking goes a long way to preventing obscure errors, as file I/O is usually buried deep within an application and many other operations are heavily dependent on it. However, other considerations, such as file formats, are also important to efficient and simplistic file access. The less time spent getting a file format to work, the better. Any work after the initial revision should be in performance, not dealing with how fragile the file is.

Task 1: File with more than 10 lines



Task 2: Result of three file writes with three lines each

Class design and implementation is extremely significant when working on teams or large projects. Knowing how to correctly split interfaces and implementations across files and thinking carefully about what methods, member variables, and other properties of the class to include or omit is what makes a codebase maintainable and extensible as the requirements grow or shrink. Additionally, it’s easier and simpler to use a well-written class wherein the applications of it are obvious. Taking the time to think carefully about what operations can be performed inside of the class repeatedly, how the class will be used, and any tradeoffs that are required from the implementation are important considerations in any project.

Task 2: Result of one file write of three lines

Task 2: Writing three lines to a file that does not exist

**File Access Flags:**

The flags I used were ios::in and ios::app. For each file, I was either writing to it or reading from it, never both. My personal preference was to use fstream when reading from a file and apply the flag instead of using ifstream as I found it easier to see the purpose of the file stream. For writing, I used ofstream and explicitly set the write mode to append. For each, I didn’t consider any other flags as it was apparent that all files read from wouldn’t be written to and it was stated in the instructions to append to the file if it existed, which the append flag performs exactly. I feel that these were the most specific flags possible and limited functionality to precisely what was desired.

**Testing:**

For each task, I considered the test cases briefly before I started and later as I was performing file operations in a more detailed manner. My initial pass at them was to look for any cases that I thought might need individual consideration – such as an empty file in task 3. I wouldn’t want to just run through the program if there weren’t any products read in, so I needed to change control in that case. However, usually the tasks involved reading in a file with either too much or too little data than would be expected. I found that accounting for the two extremes usually provided correct functionality for the typical case anyway. In the case of having to write to *or* append to a file in task 2, the append flag covered both for me.

I can imagine test cases in the future where I’ll have to make many special considerations for different circumstances, however. These will require more of a planning phase and constant consideration to determine whether the operations I’m writing will be safe in all cases.