Assignment # 3:
SE480
Fall 2012

Assigned: September 13th, 2012
Due: September 30th 2012 , 11.59pm

This assignment focuses on the pipe-and-filter architecture. You will be required to construct and implement a pipe-and-filter architectural for a text processing problem.

You may program in java, C++, or C#. If you really have another preference – check with me first.

All assignments must be submitted as a zip file to D2L. All code must have an executable file and before submission you must hardcode your program (sorry) to read the input data files (which I will provide) from C:\SE480\DataFiles. In other words I expect your executable file to run without bugs on my computer against the datafiles. (The reason for this is that the datafiles are quite large – in order to make the performance analysis meaningful, and I do NOT want you to upload datafiles with your submission). You must also submit source code.

All submissions must include a README file explaining how to run your program.

This assignment has several parts:

Part 1.

Design a pipe-and-filter architecture which reads in a text file, removes stop words (see explanation below), removes all non-alphabetical text, stems words into their root form (from now on referred to as terms), computes the frequency of each term, and prints out the 20 most commonly occurring terms (not including stop words) in descending order of frequency. If there are ties, then print the tied terms in alphabetical order.

The challenge is to complete the processing of a single file as fast as possible, while considering other architectural goals such as maintainability, reuse, and understandability. You can use multi-processing, pipelining (in the pipe-and-filter) architecture.

SOME parts of the solution include:

- **Data Source**: Input files are provided. Your program must be able to run on all four of them without breaking. (i) Declaration of Independence (Test1.txt), (ii) Complete text of Alice in Wonderland (Test2.txt), (iii) A miniature test case (I developed this when I was coding the solution myself), and finally (iv) the text of the King James Version of the Bible. Some of the data may be messy so you will have to clean it up and remove non-alpha characters.

- **Stopword removal**. We will be using the "very long list" available here: http://www.ranks.nl/resources/stopwords.html. For your convenience I have saved the words into the stopwords.txt file.

- **<u>Porter's stemming algorithm</u>**. You will need to apply a standard stemming algorithm in order to stem all words to a morphological root (for example: jumping, jumps, jumped → jump). Code is provided for many standard languages including C# and Java (not sure about python). You can reuse the provided code as is.

- **<u>Data Sink</u>.** Outputs the 20 most frequently occurring terms. In the case of a tie, continue outputting all terms tied for the position in alphabetical order.

Note: I have deliberately NOT listed all possible filters here. This is for you to consider.

For part 1 of your assignment, you must submit a fully working solution and supporting Component and Connector diagram. The C&C diagram must use UML 2.0 as discussed in class **<u>either in week 3</u>**. For more specific details, please see the grading rubric below.

Also note that you are absolutely expected to implement this assignment using elegant, standard programming practices. This means that your code should contain basic documentation (describing each class and method). You are not expected to document line by line!!! You WILL also lose points if you use poor programming practices. In other words you should bring everything you learned from SE450 to this programming assignment.

Although I HATE to encourage this – due to time constraints, and the focus of this particular class on architecture, <u>I am NOT requiring you to write automated unit tests</u>. However, I am going on record to note that I consider writing unit tests a critical and essential component of good programming practices.

NOTE: It is acceptable to download and reuse code that you find on the internet for individual filters i.e. Porter's stemming algorithm, data structure for the pipe etc. ALL reused code must be marked as such. It is NOT acceptable to borrow code written by your class mate, or to download the entire code (if you should find it) for the homework. Each person is responsible for developing their own application.

Part 1 Grading:

Section pt ranges	15-20 pts	10-14 pts	1-9 pts	0 pts
Component and Connector diagram in UML 2.0 20 points	• Component and connector diagram modeled correctly using UML 2.0 (Note: you may utilize any of the correct standard UML 2.0 modeling techniques shown in class). • Diagram modeled using a proper UML modeling tool (i.e. EA, Visio etc).	• Some non-minor notational mistakes in the UML diagram. • Diagram is correct but modeled without use of a tool (or hand-drawn).	• Significant flaws in the Component and Connector diagram. • Missing elements. Unable to see Pipe and Filter architecture in diagram. • Wrong level of abstraction (too high or too low)	• Missing
Section pt ranges	20pts	14-19 pts	1-13 pts	0 pts
System functionality 20 pts	• System correctly parses the three provided test files + one additional mystery file (not larger than Alice) and produces	• Produces approximately correct output with minor errors.	• Significant difficulties running the program. References data in incorrect directory. Runs with significant	• Program does not run and/or compile.

	correct output.		errors. Produces incorrect output etc.	
Section pt ranges	15-20 pts	10-14 pts	1-9 pts	
Pipe and Filter Implementation 20pts	• Filters are active (i.e. each runs on its own thread and synchronizes through the pipe as a buffer). (Note: you may have a hybrid model as long as the major work is done along an active pipeline of filters)	• Problems with buffer synchronization. • Problems with pipeline setup and/or teardown. • Some filters fail to stop running.	• Solution was implemented using passive filters AND data was shared via shared memory instead of through pipes.	
Section pt ranges	15-20 pts	10-14 pts	1-9 pts	0 pts
Code Quality 20 pts	• Code is well written and elegant. • Sufficient documentation (not overkill). • Good coding practices are followed (i.e. naming conventions, dynamic dispatch, programming to an interface, etc. etc) • All "borrowed" code is marked as such (i.e. PORTER etc)	• Medium-level problems in code quality (messy, undocumented, inelegant, unstructured)	• Very weak code	• Missing

Part 2:

In the second part of the assignment you will analyze the extensibility and response-time of your solution.

1. **Extensibility:** Explain how your solution would support the following extensibility goal: *The customer wishes to redesign the system to handle text files written in languages other than English. (Each file is in one language). The design time modification must take less than 1 day. The ultimate solution must be configurable automatically at runtime.*
   (Hint: Expected answer will include about a paragraph of text).

Section pt ranges	8-10 pts	5-7 pts	1-4 pts	0 pts
Maintainability Explanation 10 points	• Solid explanation of how the design supports the extensibility goal.	• Incomplete justification or problems in the argument	• Major problems in justification.	• Missing

2. **Response Time:** Using the "King James Version" text file, instrument your code (or if available, use a profiler) and report on the response time (under conditions of no contention) of your application. Do not include infrastructure setup time (i.e. loading stopwords), but do include everything related to loading, processing, and saving results from "KJV". Also evaluate the individual response times of each filter. Note that this is going to be a little tricky, because the runtime of each filter is impacted by the performance of its incoming buffer. You will have to figure out how to get around this problem. Some possible solutions include (i) determining the time needed to process a single element (i.e. read from incoming pipe, process, and output to outgoing pipe), or (ii) set up a driver component.

Discuss your results in at least one paragraph. Identify bottle necks (pipes, filters, etc).

Section pt ranges	15-20 pts	10-14 pts	1-9 pts	0 pts
Response time computed 20 pts	• Overall response time reported for Alice File. • Well justified and well supported analysis of individual runtimes for each filter, and pipe transmissions. • Evidence showing these are actual measures (screen shots, sections of instrumented code etc.	• Overall response time reported. • Individual filters – buffer times not reported. • Flaws in technique used to estimate performance.	• Major problems in response time estimation and measurements.	• Missing
Section pt ranges	10 pts	5-9 pts	1-4 pts	0 pts
Discussion of results 10 pts	• Performance results are analyzed and discussed in a meaningful way, clearly showing an understanding of the performance issues.	• Some analysis, but misses some of the key issues.	• Weak analysis and/or discussion	• Missing • Very poor quality

3. **Based on your answer from #2 "Response Time"** redesign your solution in UML to improve response time WHILE balancing the need for maintainability and reuse. Present your solution as a Component and Connector diagram. Explain why you believe your new solution will outperform your original one. Hints: your new solution might include combining/splitting filters, modifying the architectural design to use shared memory, introducing multiple instances of selected filters etc.

Section pt ranges	15-20 pts	10-14 pts	1-9 pts	0 pts
Redesign with C&C diagram 20 pts	• Correct use of UML 2.0 notation in C&C diagram. • Sensible improvements in the design to improve performance while maintaining extensibility goals.	• Some modifications but likely to have limited impact on performance and/or negatively impact extensibility	• Major weaknesses in proposed architectural solution.	• Missing
Section pt ranges	10 pts	5-9 pts	1-4 pts	0 pts
Discussion of results 10 pts	• Performance results are analyzed and discussed in a meaningful way, clearly showing an understanding of the performance issues.	• Some analysis, but misses some of the key issues.	• Weak analysis and/or discussion	• Missing • Very poor quality

Bonus points for implementing your new solution and delivering a comparative graph to show how it improves performance over the original solution. (Amount of bonus points depends on the extent of the modifications), and the achieved benefits.

Note: You can still earn bonus points if your new solution underperforms as long as your logic for the new design was well motivated, and you analyze the reasons that it ultimately underperformed (Maximum points 10).

NOTE: It is not possible to earn more than 100% of the score of all assignments summed together.

Section pt ranges	10 pts	Fewer points
BONUS POINTS for implementing new solution 10 pts	• Signfiicant modifications (i.e. concurrent processing on certain filters, redesign of filters, comparison of shared memory.) • Good analysis of results (for better or worse)	• Minimal changes in the configuration. • Weak analysis of performance improvements or degradations.

Submission

Turn in to D2L the following:

1. Source code, executable file, and README file in a single zip file. README file must tell me how to run the program. Remember it must assume data is in C:\se480\DataFiles
2. A pdf file (preferably – but I will take MS Word) with the remaining parts of the homework. i.e. READABLE images of your architectural design etc etc. You may submit in landscape mode if it is easier to fit (fairly wide pipe-and-filter) diagrams that way.

Philosophy if the code doesn't run: If I cannot run your code I will contact you to try to sort this out. If it is your error in packaging the code etc then you will lose points. However, whoever's fault it is, I will work with you to attempt to resolve the issue.

Office Hours

If you have problems getting your code to run you can (i) make use of the tutors http://facweb.cs.depaul.edu/tutors/tutor_search.asp or (ii) call me or come to my office hours. Note: I am NOT going to do your programming for you (I already completed this assignment once for myself!!!); however I'm willing to discuss homework with you. I strongly recommend that you program incrementally and DO NOT PROCEED to add more functionality until you get each of the building blocks working. I AM NOT going to help debug **big-bang failures**.