Software Development Project

Project 1, Lines of Code Counting

Report By: Nicholas Doherty

Course: Software Engineering, 3034, 01

Project Dates: 10/1/17 – 10/10/2017

Date Submitted: 10/10/2017

# Introduction:

The Lines of Code (LOC) counter is meant to receive a project source file and count the active lines of code contained within the file. The purpose of such an instrument is to give a programmer an idea of the size of their program and thus how long it might take to compile. As a result, a programmer can make a sound decision as to whether or not they need to trim the fat from their program or if they are clear to add more without overburdening a PC.

# Estimates/Plans:

Since the project simply opens a file, reads it, and reports back the amount of valid Lines of Code (LOC), the project should go relatively quickly. The code will be simple enough, probably taking only somewhere around 3 hours to code and debug as it will consist only of a main function to do the back-end work, a count function to tally things, a selector function to select a file, and an HMI module to decide continued exectution. The Interface design will probably only take 2 hours to complete and implement. The project plan will be to use a sprint model to set aside a day in order to code and debug; with another day test individual test cases before submitting the project before the deadline of Tuesday, October 10, 2017.

# Requirements Analysis:

The final product will open to main, where the user will be able to input the file path to the desired file(s). Once the user does so and closes the window, the program will open each selected file and pass it to a function that then counts all active LOC contained therein. It is important to note that a single LOC is counted as a valid line of non-whitespace characters that contribute to the overall function of a program. This excludes any and all comments or blank lines. As scope brackets define the code contained within functions or loops, a LOC might therefore include nothing more than a scope bracket. Upon completion, the function will return the integer indicating the total LOC. The program will then output the name of the file and the resultant LOC before closing the file and repeating the process. At the end, the program will output the total number of files counted and then close.

# Design:

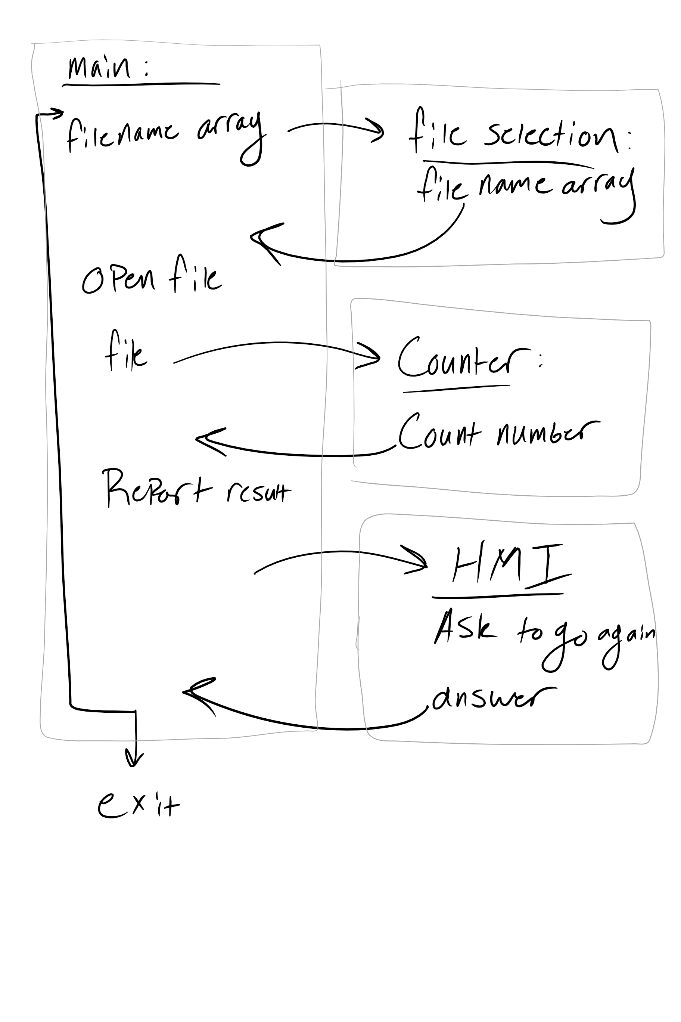
The LOC counter will consist of a main function that takes user input and outputs information to the user as well as a LOC counting function that contains the code definitions of what is and is not a valid LOC. Upon booting the program, the main function will accept the input describing which, if any, source files are to be counted and then opens the file for counting, passes the contents to the counter, and then closes the file when finished and outputs the results. The counter function will not manage the files or accept user input. In fact, the user will never know that the counter is a separate function. The counter function will be a back-end function that accepts the contents of the file and checks them against its definitions of what counts as code. It will then return the results and conclude its business.

Figure : HLD

Main Inputs:

Input files

Main outputs:

File names

Count results

File Selector Inputs:

Filenames array

File Selector Outputs:

Filenames array

Counter inputs:

File contents

Counter outputs:

Line count

# Implementation:

The code will be handled in sections. Each section will be devoted to creating each module required for the project. Once everything has been coded separately, all modules will be put together and tested. From that point on, debugging will ensure that all parts mesh together into a cohesive project.

# Testing:

The testing on the project was done as the project was put together, making it easy to debug components as issues arose.

chooseFile was difficult in that the API handling file selection changed the format in which file names were returned based on the amount of files selected. One file was returned combined with the path to the file; multiple files first had the path, then a null character followed by the first file name, another null character before the next file name, and so on. In the end, the function had to test for multiple files before either simply accepting the return string as a whole, or sending it to a parser to get the individual files.

Main was simple to test in that it simply had to call a few functions and then display the output from countLOC. Testing consisted of making sure that the output was displayed correctly and the do-while loop containing execution worked properly.

The HMI module had to be rewritten from the original c version, but once that was done, it was a very simple function to write. Rather than accepting user input in the form of letters or numbers, it displayed a box with a yes or no button. Minimal testing was required to make sure the two buttons worked.

countLOC on the other hand required a lot of testing. Each case had to be handled. The approach was to define what was NOT a LOC and then work backwards from there. Some logic, like blank lines or line comments were easy to test, as they have relatively simple logic. The block comments required the most thought, as there was so much that could be done with them. In the end, testing was done as each new logical rule was introduced. It was tedious, but the logic is sound. As a result, the program handles a LOC before a block comment begins and after a block comment ends.

# Conclusion/Project Retrospective Analysis:

The project took more lines of code than expected. The biggest parts were the file selection and countLOC functions. They required all sorts of if statements describing what counted as what or how things should be formatted. Everything else operated as expected and was thus easy to code and/or required very little in the way of LOC.

The biggest lessons to take from the project would be to take more time considering just how much effort needs to be put into something before it will be completed.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Estimated: | Measured: | Analyze: %Error |
| Time | 17.85 Hrs | 14 Hrs | 21 % |
| Size | 250 LOC | 464 LOC | 46 % |

Actual Project Time and Program Size vs. Estimates

# Appendices:

Time Estimates:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Total Size Estimate: | Total Time Estimate: | |  |  |  |  |  |  |
|  | 250 | 17.85 |  |  |  |  |  |  |  |
| Units: | LOC | Hours |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Sub Phase: | Estimated Size: | Estimated Time: (hours) | |  |  |  |  |  |  |
| Preliminary Analysis | (N/A) | 0.75 |  |  |  |  |  |  |  |
| Planning and Estimation | (N/A) | 1.25 |  |  |  |  |  |  |  |
| Requirements Analysis | (N/A) | 1.00 |  |  |  |  |  |  |  |
| Requirements Specification | (N/A) | 2.00 |  |  |  |  |  |  |  |
| Decomposition | 50 | 3.00 | "Baseline" code |  |  |  |  |  |  |
| Interface Design | 150 | 2.00 | "Interface" code | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | Estimated Time: (hours) | | **[Note that Total Time is automatically calculated.]** | | | |  |
|  | Estimated Size: | Total Time: | **Iteration Plan** | **Detail Design** | **Test Design** | **Code** | **Unit Test** | Integration Test | Iteration Retrospective |
| LOC Check Function | 50 | 5.85 | 0.30 | 0.50 | 0.75 | 0.30 | 1.00 | 1.50 | 1.50 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Functional Test | (N/A) | 1.00 |  |  |  |  |  |  |  |
|  | (N/A) | 1.00 |  |  |  |  |  |  |  |

Time Log:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date:** | **Start Time:** | **Stop Time:** | **Duration:** | **Feature/Component Name:** | **Development Phase:** | **Notes/Description:** |
| 10/1/2017 | 8:06 PM | 9:18 PM | 1:12:00 |  | Project Plan |  |
| 10/2/2017 | 12:15 PM | 1:25 PM | 1:10:00 |  | Project Plan |  |
| 10/2/2017 | 8:41 PM | 9:41 PM | 1:00:00 |  | Project Plan |  |
| 10/8/2017 | 4:26 PM | 5:02 PM | 0:36:00 |  | HLD |  |
| 10/8/2017 | 5:02 PM | 5:54 PM | 0:52:00 | Main | Code | Coding the greeting and passing of variables |
| 10/8/2017 | 7:48 PM | 9:53 PM | 2:05:00 | Main | Code |  |
| 10/9/2017 | 12:01 PM | 2:31 PM | 2:30:00 | File Selection Function | Code |  |
| 10/9/2017 | 2:31 PM | 2:45 PM | 0:14:00 | LOC Counter Function | Code |  |
| 10/9/2017 | 4:48 PM | 5:54 PM | 1:06:00 | HMI Module | Code | Coding an improved HMI Module |
| 10/9/2017 | 6:54 PM | 8:29 PM | 1:35:00 | countLOC | Functional Test | Retooling LOC Definitions |
| 10/9/2017 | 11:58 PM | 11:59 PM | 0:01:00 | Project Report | Project Retrospective | Filling out the project Report |

Defect Log:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Defect ID # | **Date:** | **Start Time:** | **Stop Time:** | **Duration:** | **Feature/Component Name:** | **Phase Injected/Introduced; Origin (Where?):** | **Defect Type (What?):** | **Defect Mode (Why?):** | **Phase Discovered** | **Defect Status** | **Change Defect** |
| 1 | 10/8/2017 | 12:53 PM | 2:31 PM | 1:38:00 | chooseFiles | Test | Test: Test Software | Incorrect | Unit Test | Closed | X |
| 2 | 10/9/2017 | 8:29 PM | 11:57 PM | 3:28:00 | countLOC | Test | Test: Test Specification error | Incorrect | Unit Test | Closed | X |