### **Command Line Management System**

*2.0 System Requirements*

* Create a menu system that allows the user to select any operation
* Create individual working functions assigned to each operation
* Allow the user to carry out as many operations as desired without the program exiting
* Create a working Change Log that stores all past information on changes to all files
* Add an additional two additional operations

*2.1 System Design*

**2.1.1 How to Specify what Operation to choose**

I decided I would create a simple menu system, that assigns each operation to a number. The user is presented with a choice of these numbers, and whatever input is entered, the program will run the function assigned to that number. I also decided that it would be rather inconvenient for the program to exit immediately after one operation has been completed, as what if the user had intended to complete several actions in one boot up. Therefore I decided I would make the program loop back every time an operation is complete, so the user can perform as many operations as they desire without having to reload the program. However, I needed to make a way for the user to actually exit the program, so I had to add an option that would terminate the program, so it would not be stuck in an infinite loop.

**2.1.2 Justifications for Usefulness for Additional Operations**

*2.1.2.1 Merging Files*

One important use of this feature could be in group projects, where more often than not, several documents, files, data and so on will be submitted from every angle. An example of this could be if the project was a programming project write up itself, where one person submits the System Requirements and the Design, and where the other submits the Testing and Evaluation. Merging these files into a one singular file with one notable format is a lot more convenient and productive than having information scattered all over the place, where the files could be difficult to locate.

A more professional use higher up in industry could be when developers are simultaneously working towards the same project, but write individual code. One developer may need the other's code in his program, so rather than copying and pasting, merging could be used. A further extension of this function I could add in the future would be the possibility to merge several files into one, say 4 or 5 rather than only 2.

Another example is storing documents of notes separately for different lectures. Merging them would mean the user would have less files to navigate to, reducing the time spent trying to find a document. A non-educational example, that would have personally been of use, would be when an estate agent sends a form out to each of the potential tenants before a rental agreement. These forms can be merged into one document, and sent to the estate agent, saving time for both parties. Merging also takes less space and reduces file size, as there will be only one memory location holding a file compared to two.

*2.1.2.2 Comparing Files*

Comparing data from two smaller files manually can be acceptable, but when it comes to files containing hundreds or even thousands of lines of data, then manual comparison can affect productivity heavily. The data to be compared in this situation for example could be logging data from an experiment, and checking if the two experiments yield the same data. I could see myself using this feature personally, as in my Foundations of Accounting optional module, I am required to write yearly cash flows. I could compare cash flows between 2 years and see if there has been any differences in the companies assets, or whether the value of the assets has remained the same.

Yet another use could be for marking multiple test tests or quizzes. The program will identify the row and column where the difference is located, so wherever the difference is, this is where the error in the quiz is. Deduct the number of differences from the number of questions, and this is the final mark. All these tasks done manually could be tedious, and also prone to human error, whereas an automated program will present all statistical differences in a clean readable manner with no mistakes.

This tool could be used as a code correction tool in programming projects like this one itself. If a small change is made to a program leading to a failure in compilation of the program, or even logic errors, the errors can be hard to locate. However, if a version of the code is saved beforehand, the two files can be compared, and the exact location of the altercation can be found, and the error can be reverted. If different developers are working on the same project, a developer can use this as a version control software, and find out what the impact of the new code is to the overall program and find out the effect of removing it, by finding the location of the changes. A possible extension I could add is allow the user to also add append a name after each file change, so if a developer goes to compare they can view who exactly it was who made the modifications, and can consult them directly if they have any potential queries.

*2.2 Implementation*

**2.1.1 How to Specify what Operation to choose**

The majority of the implementation explanation is explained in the code comments, but in general I ensured that the user was brought back the menu system after each operation completion using a do while loop. As long as the user doesn't enter the option to exit the program when prompted to at the menu screen, the main driver code will continuously run asking the user to choose another option. I also added an if statement to scan for integers only to be accepted. If a non-integer value was to be entered, the program normally would infinitely loop due to the do...while loop, but due to this if statement exiting the program if a non-integer value is entered, the infinite loop does not occur. If a integer not defined in the switch case is entered, then the program will repeatedly run asking for a correct input to be entered.

**2.1.2 fgetc() vs getc()**

The reason I decided to use fgetc() rather than getc() is because the former cannot be implemented as a macro. This means that fgetc() is guaranteed to be a function, so I am able to take it’s address if needed, and pass this address as an argument to another function. However the disadvantage of this is that calling fgetc() takes slightly longer, as it takes slightly longer to call a function.

Source - Advanced Programming in Unix Environment

**2.1.3 scanf() vs gets() vs fgets()**

Both scanf() and gets() will read input until newline or EOF is encountered, but the difference is scanf() will not read whitespace. This could be seen as a disadvantage, but I decided that the lack of whitespace in file names with scanf() provides a more succinct, cleaner experience for the user, and reduces any uncertainty over exact position of whitespace. Another advantage of scanf() is that the user does not need to be aware of the size of the stack, however I am aware that the input string buffer can be overflowed, but it is almost guaranteed that the user enters a name than scanf() can handle, in order for conveniency. However, for some functions, such as for appending lines or reading a line, I used fgets() to read one line of text at a given time from the file. This because I wanted the content to have a default buffer size. in order to prevent buffer overflow attacks. This buffer size is ‘sizeof newData’. This buffer was allocated dynamically, rather than by stack, in order to further prevent stack smashing.

fgets(newData, sizeof newData, stdin);

**2.1.4 ChangeLog.txt**

My implementation of Change log can be seen as potentially confusing. For each option chosen to run an operation, the function for that specific operation is called. However, I needed to find a way to write the resultant number of lines after each operation, for that file. Due to the fact I had already written a numberOfLines() function, which was an existing option, I decided it would be efficient to just call this function immediately after a normal operation function. In this way, the user re-enters the name of file, and the resultant number of lines is outputted, and printed to ChangeLog.txt.

A simple file name function was created, and is called subsequently after every number of lines method, which asks the user to enter the same file name as first entered when performing an operation. This is to assign a file name to the resultant number of lines in Change log. I realise this is not the most effective solution, but it does work and get the job done. However, a small problem is that if the user enters a different file name , but one that does exist when prompted to, to store line number, then the ChangeLog will show an incorrect action

**2.1.5 char filename[255]**

I decided to store user input in a char array. It was purely a personal choice keeping the maximum number of characters as 255, as extended ACSCII is a 8-bit character set, meaning 2^8 = 256 characters can be referenced. As counting starts from 0, the max ASCII char code is 255, hence why I chose it. I wanted my system to be in somewhat resemblance to professional software ; for example on Windows, given c:\section1\section2 , section1 and section2 should be both 255 or less characters.

**2.1.6 Error Handling**

I decided to handle errors of when names of files being entered already exist, or when files are not found, by first opening the file in read mode, using fopen() which returns a pointer to the structure FILE. If a connection is successful, the function returns a pointer to the file, but if no connection is found, then NULL is returned. I then had to use exit(0) to make the program exit if a NULL was returned, else a segmentation fault would occur due to program trying to continue with no pointer to a file returned.

*2.3 Testing*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | Test | Expected Outcome | Outcome | Result |
| 1 | Program correctly compiles with no errors and menu system is outputted to screen | Menu will appear, allowing user to choose an option | Menu appears, allowing user to choose an option | Pass |
| 2 | For a valid input corresponding to a option in the menus, the program runs. For example, enter input of 3 | Outputs to user they have chosen option 3. Runs the function to copy a file, as this is option 3 | Outputs to user they have chosen option 3. Runs the function to copy a file, as this is option 3 | Pass |
| 3 | For a invalid input not in the menu options, e.g a value greater than 13. Test input is the number 14 | No operation will be chosen, as it is an invalid input and the menu will reappear asking the user to chose an option yet again | No operation will be chosen, as it is an invalid input and the menu will reappear asking the user to chose an option yet again | Pass |
| 4 | For another invalid input that is not an integer, e.g a string, Test input is hello | Program will be exited | Program is exited | Pass |
| 5 | User can select the option that allows them to intendedly exit the program | Program will be exited | Program is exited | Pass |
| 6 | For option 1, a new file with a new name can be created. Test input for new file name is 'new123' | Outputs to user that file is created, and file is created is with name 'new123' | Outputs to user that file is created, and file is created is with name 'new123' | Pass |
| 7 | For option 1, enter same name of file to write the number of lines to ChangeLog.txt. Test input for file name is 'new123' | Will output the created file 'new123' has 0 lines. Will write that this file has 0 lines to ChangeLog.txt | Will output the created file 'new123' has 0 lines. Will write that this file has 0 lines to ChangeLog.txt | Pass |
| 8 | For option 1, enter invalid name of file that does not exist to write number of lines to ChangeLog.txt. Test input for file name is 'hello123' | Output that File does not exist. The fact that the original file was created called 'new123' and that the follow up function received an invalid file name input, means nothing will be written to ChangeLog.txt in this instance | Will output that File does not exist. The fact that the original file was created called 'new123' and that the follow up function received an invalid file name input, means nothing will be written to ChangeLog.txt in this instance | Pass |
| 9 | For option 1, if file name entered already exists, file is not created, and existing content is not overwritten | Program should output file is not created. | Program outputs file is not created. | Pass |
| 10 | For option 2, if filename entered corresponds to existing file, file should be deleted. | Program outputs file deleted and file is removed | Program outputs file deleted and file is removed | Pass |
| 11 | For option 2, if filename entered does not correspond to existing file, no file Is deleted | Program outputs ‘Invalid File Name’ and program is exited | Program outputs ‘Invalid File Name’ and program is exited | Pass |
| 12 | For option 5, to append a line, can 2 lines be appended. | Allows user to enter 2 lines of input | Allows user to enter 2 lines of input | Pass |
| 13 | For option 6, desired line should be deleted from specific file | Allows user to enter the line number of which they want to remove, and the file is displayed to screen with updated contents. | User is allowed to enter the line number of which they want to remove, and the file is displayed to screen with updated contents. | Pass |
| 14 | After any option, enter correct file name again to store file name in ChangeLog.txt | Nothing is outputted, but filename will be written to ChangeLog.txt (along with number of lines) | Nothing was outputted, but filename will be written to ChangeLog.txt (along with number of lines) | Pass |
| 15 | For option 8, content can be inserted at a line with existing content in, if desired index is less than line length | Line Insertion Complete. Outputs updated content of file | Line Insertion Complete. Outputs updated content of file | Pass |
| 16 | For option 8, content should not be inserted if the inputted line number is 1. | Outputs contents of file, but new content won’t actually be inserted, as content on line 1 cannot be inserted | Outputs contents of file, but new content won’t actually be inserted, as content on line 1 cannot be inserted | Pass (but this flaw can be worked upon) |
| 17 | For option 8, content should not be inserted if the inputted line number would mean content goes on a completely new line | Outputs contents of file, but new content won’t actually be inserted, as content on new line cannot be inserted | Outputs contents of file, but new content won’t actually be inserted, as content on new line cannot be inserted | Pass (but this flaw can be worked upon) |
| 18 | For option 9, if correct file name is entered, is number of lines outputted. | File ‘x’ has ‘y’ number of lines | File ‘x’ has ‘y’ number of lines | Pass |
| 19 | For option 9, if invalid file name is entered, number of lines should not be outputted. | Outputs that file does not exist | Outputs that file does not exist | Pass |
| 20 | For option 12, is ChangeLog.txt printed to screen | Outputs contents of the Change Log | Outputs contents of the Change Log | Pass |
| 21 | For option 11, if either of the file names entered is invalid, no comparison should occur | Outputs File cannot be opened for comparison, and program is exited. | Outputs File cannot be opened for comparison, and program is exited. | Pass |
| 22 | For option 11, if both file names are entered are valid, output the differences | The total number of differences along with line number and difference position are outputted | The total number of differences along with line number and difference position are outputted | Pass |
| 23 | For option 10, if either of the file names entered is invalid, no merging should occur | Outputs File cannot be opened, and program is exited. | Outputs File cannot be opened, and program is exited. | Pass |
| 24 | For option 10, if if both file names are entered are valid, both files are merged and the original files are deleted. | Program asks you to enter name for new merged file. After entering name, files should be merged into one singular file, and deleted | Program asks you to enter name for new merged file. After entering name, files should be merged into one singular file, and deleted | Pass |

*2.4 Evaluation*

Overall, my program does meet the criteria asked for in the question, so it can be considered a success. On the other hand there are some design improvements I am aware that could be made, as repeatedly typing out the same file name over and over can be tedious. I could also develop the error handling further, if an incorrect file name is entered, rather than exiting the system, the user is taken back to the menu. Another small change is to the user insert a line on line 1 or on a new line in my insertLine() function, but as the user has been told of these limitations explicitly beforehand, it should not be a major inconvenience or shock.

*2.5 References*

* <https://www.geeksforgeeks.org/why-to-use-fgets-over-scanf-in-c/#:~:text=Advantage%20of%20using%20scanf()%3A,strings%20until%20BufferOverflow%20error%20occur>.
* B. W. Kernighan, D. Ritchie. The C Programming Language (2nd Edition), Prentice Hall, March 1988
* <https://stackoverflow.com/questions/30415276/inserting-a-text-into-a-specific-line>
* <http://www.cplusplus.com/reference/cstdio/>
* <https://beginnersbook.com/2014/01/c-file-io/>
* <https://www.w3resource.com/c-programming-exercises/file-handling/c-file-handling-exercise-10.php>
* C. Programming Language: An Applied Perspective by Lawrence H. Miller
* https://www.tutorialspoint.com/c-program-to-delete-a-file