Introduction to Software Engineering Final Assessment

20551410 - Jason Nguyen

Wednesday 4:00PM - 6:00PM

Building 314, Room 218

Introduciton

For the Final Assessment of ISAD1000, we have been tasked with create a program that can perform multiple conversions. We are being evaluated not by our coding skills, but rather our coding pratices which were picked up throughout the semester. It has been designed to test the basics of **software testing, modularity, version control and ethics**. The were taught and tested through the semester with weekly signoff, however this assessment is about to test our ability to incorporate these technics is actuality.

In my code I have decided to implement the all four of the **Category 1** functionalities and then **Category 2's** time conversion. My code has implemented a system that initially askes if the user would like manually input their values, or to allow the program to read from a file pointed to by the user within a looped menu to avoid any misinputs. When the user makes their choice they are asked to confirm whether what they have entered was correct, and if it was not then they will get an opportunity to type in their string once more. Afterwards the program brings them to another menu displaying the choices of conversion they'd like to follow through with is displayed. The options displayed are:

1. Convert given string to upper case or lower case. . . . . .(1a)
2. Identify whether numeric values are in a given string . . . . . .(1b)
3. Identify whether a given string is a valid number . . . . . .(1c)
4. Remove numeric values in a given string then convert to upper or lower case . . . .(1d)
5. Convert time. . . . . .(2c)

Aferwards, the user can choose any conversion they'd like and the program will run the conversion and print out the result, along with saving the result to a text document. After each conversion the user is brought back to the same menu where they can choose another conversion or to hit **0.** to exit and quit the program.

The code has been designed to where I have created multiply functions that rely on one another. Newer functions can call upon old functions to use their outputs. While this does increase coupling, it does promote the program to be more generic and versatile. The program is designed to use arrays and lists, for both automatic and manual data entry, so no matter what method of inputting data, the program still runs the same exact code without needing modification. There is a menu class which will be ran on startup, however it only gets the input type which conversion the user would like to run. When it comes to the actual calculations, the functions have been put in another class which the menu file calls and uses.

As for testing, white and black-box testing have been performed throughout the code. Testing actually helped discover a few flaws that were ocerlooked, so thanks to the testing the functionality of the code has been even greater improved.

Module Description

My program currently has five main functions. All of **Category 1 and Category 2's (C)** conversions. However, there more methods contained in the program exist to aid in the opperation of the system. They were designed to enable the repetition of code without worrying about data redundacy. Each of the main modules was heavily described in the the Preliminary Description documents, however smaller functions haven't been.

1A. Converting a String to Uppercase or Lowercase

This function is intended to be used to take a string in and then convert the letter to uppercase or lowercase based on the user's choice. The program should ignore all non-letter characters and only perform the capitilization on the letters of the English alphabet. It doesn't support special letters with tones, as programming for each case would cause the program to contain a lot of if-else or cases, which would hinder the readability of the program. A rough description of how the program works is quoted and adjusted from the Preliminary Description:

> - This method will run a menu loop that asks the user to choose between converting to uppercase and lowercase.
> - Afterwards the program will run a for loop that will get the i'th element of the list, and set that as a char array.
> - This array will then passed into "charArrayToAsciiArray" to which it'll then be set to an ascii array.
> - This ascii array will then be passed into a function called "arrayToUpper" or "arrayToLower" depending on the option chosen by the user.
>   - The program will check if the int is within the range of being either lowercase or uppercase and convert them to the right case by adding or subtracting 32 for each respective cases.
>   - It'll loop through the entire array and set the current element of the passed array.
>   - Then it'll run asciiArrayToString method and set the output to "returnString"
>   - Afterwards it'll return this string.
> - The returned value will then be saved into a "convertedString" variable.
> - And the program will print both the original and "convertedString" variable out to the user and save it into a next row of a new file.
> - The program will then loop through and then afterwards end itself and save that new file.

1B. Identifying Whether or Not a String Contains Any Numbers

This function is inteded to be used in order to take in a string, and then check if there are any numbers within the string. After checking each character in the string th eprogram will display a message of whether there was or wasn't a number within the string. Afterward it should write the result out to a file. It'll accomplish this by checking the ascii value of the char against the range of ascii values between 48 & 57 inclusive, as those are the ascii values of the number 0-9. A runthrough of how the program is quote and adjusted form the Preliminary Description below:

> - The method will run a for loop that will get the i'th element of the list, and set that as a char array.
> - This array will then passed into "charArrayToAsciiArray" to which it'll then be set to an ascii array.
> - The ascii array is then sent into a function called "checkNumericsExist" and the output assigned to "doNumericsExist".
>   - The function will loop through each element in the array.
>   - And check them against the range of 48 to 57 inclusive.
>   - If there is a number within the range, a return boolean is set to true, otherwise it remains false.
> - Afterwards, the main method will run an if statement check if the variable "doNumericsExist" is true or false.
> - Depending on which state the boolean is, it'll print out saying that there was or wasn't a number in the given string. Lastly it'll save the print output into a file.

1C. Identifying Whether or Not a Given String is a Valid Number

---

The purpose of this function is to identify is the string that is give is a valid number. It checks is there or only numbers contained within a string that is provided, otherwise it'll tell the user that the string proposed was no valid. It does this by checking if the passed ascii values array has any values which are not within the range of 48 to 57 represent the numbers 0-9. If there are any outside this range it'll return a string state that the initial string provided was invalid. A rundown of the code is taken from the preliminary description and shown below:

> - The program will run a for loop that will get the i'th element of the list, and set that as a char array.
> - Afterwards, inside this loop the array is passed into a function called charArrayToAsciiArray to convert the char array to an ascii array.
> - This array is then sent to "arrayToNumbers" and the return value to a list.
>   - Inside "arrayToNumbers" it'll loop based on the length of the array.
>   - Inside this loop it'll set "inputArray" to the i'th element and compare using an if statement.
>   - If "inputArray" value is between 48 and 57 inclusive, it'll set outputString to outputString + the respective number.
>   - Otherwise, it'll set "stringIsInvalid" to true, and set "nonNumberString" to the "noNumberString" + the value of the current element.
>   - At the end it'll check if "stringIsInvalid" is true.

        - If so, the returnList element 0 is set to " was invalid. (Commas and Periods are not included)".
        - Otherwise the returnList element 0 is set to " was the valid number: " + outputString
        - Plus it'll set the returnList element 1 to "nonNumberString".
        - And returnList element 2 to "outputString".
    - Afterwards the function will set "stringToPrint" to the 0'th element of the returned list.
    - Lastly, the program will print out the result and save it to a text file.

1D. Removing Numbers from a String and Changing the Case

This function is designed to take in a string and then remove any numbers that are in it and then change the capitalization of the string through user input. First it'll take in the string and convert it to an ascii array, and then check if there are any ascii values that represent numbers. If so it'll remove those values and then ask the user if they'd like to go to lowercase of upppercase. This function reuses code from 1A and 1C, and a breakdown of the code is taken from the Preliminary Description and shown below:

- This method will run a menu loop that asks the user to choose between converting to uppercase and lowercase.
- Afterwards, the program will run a loop that'll get the i'th element of the passed in list, and set it to a char array.
- This char array is then converted to an ascii array.
- The ascii array then sent to "arrayToNumbers" and the 1st element set to "nonNumberString".
- "nonNumberString" is then changed to a char array then to an ascii value array.
- Afterwards, the new ascii array is then passed into "arrayToUpper" or "arrayToLower" depending on the the choice the user made.
- Finally it is printout out and then saved into a text file.

2C. Conveting a Number Between Hours, Minutes and Seconds

This code is used to convert a given amount of time to a different unit of time. It can convert seconds, to minutes or hours, vice versa in any combination. You could go directly from hours to seconds, rather than hours to minutes. Doing it this way made it a lot easier without having to think about too much extra. It works by taking in the the units the user in coming from and then which units they'd like to convert to. An overview of the code was taken from the Preliminary Decription, adjusted and displayed below:

> - The function will start by running a new menu funciton called "whichTimeConversion".
> - This'll get the units in which the user wants to convert to and from.
> - It'll set the units to "convertTo" and "convertFrom".
> - Afterwards, it'll run a loop that will get the i'th element of the passed in list and set it to a char array.
> - This char array is then converted to an ascii array.
> - Then "timeNumber" is set to the parsed integer form of the 2nd element of "arrayToNumbers" passing in the ascii array.
> - Afterwards a switch case is used to check "convertFrom" and then run if statements inside each case.
>   - It checks the "convertTo" value and then does multiplication or division dependingly.
>   - It all also does a mod check to get the remainder and then set each of the "convertedHours", "convertedMinutes", "convertedSeconds" variables.
> - These variables are then put into a string with the original unit to convert from.
> - And then printed to the terminal and saved in a text file.

Modularity

In order to run the production code, you need to build the Menu.java file using `javac Menu.java`. This should automatically build the Function.java aswell, as the Menu file calls upon the Function file. Afterwards the Menu can simply be run using `java Menu`. All these commands should be run inside the Cdoe folder directory. Afterwards the user can just follow the on-screen instructions and make their choices using the keyboard.

Some of the modularity concepts used in my code are, coupling, cohension and redundacy. Code has been programmed in a sense where there is little to no reliance on global variables. All functions create their own local variables, and either get passed or return variables. There should be no instances where global variables are being adjusted within the entire program. Furthermore, variables have been named very clearly and are named based on their function so that upon quick inspection, it is easy to tell what the variable should be. i.e, the boolean wantUpper would indicate that depending on if it's true or false the user want to convert to upper or lowercase. Additionally, all functions and variables have adhered to camal casing practices.

Cohesion and redundacy have been represent through the menu codes. Most of the menus are run using methods that contain loops. This is so that the main code can be presented as more readble while still being able to undestand where the certain print statements are located. Within these menus, they all run while loops that check the user's input and they should all be able to realise when the input is invalid and request the user to try again. A prime example or redudancy avoidance, is the reusage of modules from 1A and 1C for the completion of 1D. The task for 1D requires the program to remove numbers, and change the cases, therefore I just reused the menu from 1A asking which case the user would like to change. In addition, from 1C, additionally functionality was added so that while it's check the ascii values of the non 0-9 numbers, it'd add

them to another array and return that as 2nd element in the already returned list. Writing the code in 1C made more sense than rewriting the same exact checking code except with different outputs. By simply adding an extra few lines of code, it made it reusable and prevention the need for excessive repeated code. Afterwards just converting the arrays to upper and lowercase was easily done by calling the respective functions.

Coding Checklist

1. Is each "/" operator working with the correct datatype?
2. Do all string comparisons use the ".equals()" method?
3. For functions that return a value, is the returned value being used?
4. Are there any variables that have not been initialized?
5. Are the return values the same and as the function type?
6. Are there "tolerance" values for real numbers?
7. Do calls provide the necessary parameters in the correct order?
8. Can a variable's purpose be quick and easily identified by just the name?
9. Are there null values being used?
10. Are arrays and variables atleast set to empty rather than null?
11. Are there any excess global variables?
12. Are if statements less than 3 if's long?
13. Are lines of code larger than the screen and need to go to next line?
14. Are comments on seperate lines?
15. Are all lines of code ended with a semi-coloum, where necessary?
16. Are variables created, and assigned in the right order?
17. Is there any repeated and unneccessary code?

Black-Box Test Cases

Some of the code couldn't be tested as I didn't know any way to test it without hardcoding an input value, since most of the functions take in user input. Test code still remains, however just commented out in the production code. Furhtermore, manual testing was conducted during the writing of the original code to ensure that the function could be built, therefore most tests were run through the terminal while running the program. Nonetheless, there were still some functions that could be tested using the testing techniques.

All test cases have had their test functionality be put into a file called TestFunctions.java, where there is a menu to choose exactly which function is tested. Each function is tested using the assert function. Futhermore, it is assumed that the inputs will be done using a keyboard and that they are correct to avoid crashing the testing code. JUnit was not used as I could not get it work during my workshop, and did not have enough time to spend attempting to get JUnit to work.

1A. Converting a String to Uppercase or Lowercase

Boundary Value Testing was chosen to test this function. It was chosen as the input is an integer from an integer array. The code compares current element of the array against a range of integers that represent the capitalization of each character in a string. Testing these boundaries can verify that the set ranges are working properly and perform the right function based on the calculation being required. However, the code still needs to take in the user input before performing the conversion, as it needs to ask if the user is converting to uppercase or lowercase. There is no need to test these values, as the program will automatically let the user know if the input was invalid.

Since we are only testing the raw, arrayToUpper and arrayToLower, we can use an array with the numbers, -1 & 0, to check the valid ascii boundary, 64 & 65 for the boundary between uppercase "A" and then 90 & 91 for the boundary between uppercase "Z" for arrayToLower. Values between 65 and 90 inclusive, should be registered as valid and change to their lowercase counterpart, while any other values should remain the exact same. Similarly, for arrayToUpper, the values -1 & 0, for ascii range, 96 & 97 for lowercase "a", and 122 & 123 for lowercase "z". Once more, only those within the lowercase bounds should change and nothing else should become different.

arrayToUpper	Test Data	Expected Result
-1 / 0	-1	-1
	0	0
96 / 97	96	96
	97	65
122 / 123	[122, 123]	[122, 91]

| arrayToLower | Test Data | Expected Result | | -1 / 0 | -1 | -1 | | | 0 | 0 | | 64 / 65 | 64 | 64 | | | 65 |97 | | 90 / 91 | 90 | 122 | | | 91 | 91

Input	Expected Output
"1410" + u	"1410"
"1410" + l	"1410"
"Nguyen" + u	"NGUYEN"
"Nguyen" + l	"nguyen"
"Jason NGUYEN" + u	"JASON NGUYEN"
"Jason NGUYEN" + l	"jason nguyen"
"Doctor Strange in the Multiverse of Madness" + u	"DOCTOR STRANGE IN THE MULTIVERSE OF MADNESS"

Input	Expected Output
"Doctor Strange in the Multiverse of Madness" + u	"doctor strange in the multiverse of madness"

1B. Identifying Whether or Not a String Contains Any Numbers

Boundary Value Analysis was chosen to test this function. In this function, the main function called "b_DoNumericsExist" does not return any value, instead only the method call "checkNumericsExist" contains a boolean return value. Therefore, I have choosen to test this module for the functionality testing of 1B. BVA was chosen as the input is an integer array that is from a range of varibles. Since the output can only be valid or invalid, boundary value testing was chosen. The selction was because the integers between 48 to 57 inclusive represent the numbers 0 to 9 meaning that anything within this range is valid and everything outside is invalid.

Invalid	Valid	Invalid
<= 47	48 - 57	>= 58

checkNumericsExist	Test Data	Expected Result
47 / 48	47	False
	48	True
57 / 58	57	True
	58	False

Input	Expected Output
"1410"	"Numbers do exist"
"Nguyen"	"Numbers do NOT exist"
"Jason NGUYEN"	"Numbers do NOT exist"
"Doctor Strange in the Multiverse of Madness"	"Numbers do NOT exist"

1C. Identifying Whether or Not a Given String is a Valid Number

Similarly to 1B, 1C will also be using bounday value analysis. The main function of 1C is called "c_StringIsValidNumber" however it doesn't return any values, instead it takes the returned value of "arrayToNumbers" and uses that, hence in this test case, we will be testing for "arrayToNumbers". Since this function checks the boundaries between 47 & 48 to 57 & 58, and sets the validity only when the integer is out of the range, we can use BVA to test the output.

String = Invalid	Ignore	String = Invalid
<= 47	48 - 57	>= 58

arrayToNumbers	Test Data	Expected Result
47 / 48	47	False
	48	True
	[47, 48]	False
57 / 58	57	True
	58	False
	[57, 58]	False

Input	Expected Output
"1410"	"Valid"
"Nguyen"	"Invalid"
"Jason NGUYEN"	"Invalid"
"Doctor Strange in the Multiverse of Madness"	"Invalid"

1D. Removing Numbers from a String and Changing the Case

For 1D, "d_RemoveNumericsAndConvertCase" does not return any values, but instead prints it directly to the terminal. However it does use the functions of "arrayToUpper" and "arrayToLower", and "arrayToNumbers", from 1A and 1C. Therefore, the only tests that can be run are those for the aforementioned functions, which have been performed in prior test cases. Manual testing and entering inputs can still be performed, however code for such cannot be provided as I don't know how to program inputs without hardcoding and testing them.If manually entering the data was testable, it'd fall under equivalence partioning as we are just testing a range of two outcomes, not many boundaries like before.

arrayToUpper	Test Data	Expected Result
-1 / 0	-1	-1
	0	0
96 / 97	96	96
	97	65

arrayToUpper	Test Data	Expected Result
122 / 123	[122, 123]	[122, 91]

arrayToLower	Test Data	Expected Result
-1 / 0	-1	-1
	0	0
64 / 65	64	64
	65	97
90 / 91	90	122
	91	91

String = Invalid	Ignore	String = Invalid
<= 47	48 - 57	>= 58

arrayToNumbers	Test Data	Expected Result
47 / 48	47	False
	48	True
	[47, 48]	False
57 / 58	57	True
	58	False
	[57, 58]	False

Input	Expected Output
"1410" + u	""
"1410" + l	""
"Nguyen" + u	"NGUYEN"
"Nguyen" + l	"nguyen"
"Jason NGUYEN" + u	"JASON NGUYEN"
"Jason NGUYEN" + l	"jason nguyen"
"Doctor Strange in the Multiverse of Madness" + u	"DOCTOR STRANGE IN THE MULTIVERSE OF MADNESS"
"Doctor Strange in the Multiverse of Madness" + u	"doctor strange in the multiverse of madness"

2C. Conveting a Number Between Hours, Minutes and Seconds

For 2C, converting between time units, I have choosen to go with equivalence partioning as we are test for validity of three values. We aren't able to test "whichTimeConversion" because it, alongwith the main function "d_RemoveNumericsAndConvertCase" do not return any values. By testing certain values we can see if the program recognizes the inputs, however since the program requires an input, I am unable to test the code without any hardcoding.

However, "getFromUnit" can still be tested ask it takes in a char and outputs a string. This can easily be done using an assertion.

getFromUnit	Test Data	Expected Result
Hours	'h'	"Hours"
Minutes	'm'	"Minutes"
Seconds	's'	"Seconds"

Input	Expected Output
All test strings	Invalid, infinite loop waiting for any of the correct chars

White-Box Test Cases

All test cases you have designed as answer for the part 5 of this assessment, assumptions you made if any, brief explanation why you have done the test design in the way you have done

The two cases in which I choosen to perform white-box testing is for my Main Menu and 2C. Time conversions is a function that runs almost entirely self contained, relying on two other functions to convert numbers and char into strings; not performing calculations. Main menu was choosen as it is the first thing that users will interact with, so I'd like to perform as much extensive testing as possible. All the other functions are just additions, however the main menu is where all the other methods are called. Therefore I'd like to focus as much as possible on these functions.

The main menu regularly takes in inputs, therefore I don't know how to write an additional file to change these inputs, instead I have choosen to hardcode certain values trying to test the program. These values can still be found within the code, except they are just commented out. By hardcoding the values, I can identify which exact variable is causing issues in the menu. For the menu I intend to perform both branch and path testing as there are many yet few paths the code can take. Furthermore, there are two branching locations, 1st at the input type menu, and then at the function type menu. Quickly in my head, are 10 possible paths to take in the main of the menu file. 5 paths for each conversion type, along with 2 input types: manually and from a file.

Hardcoding the string list for the listOfStrings variable is simple, however testing each function that can be performed will be strenuous, therefore, I will just put print statements to know that the switch case's case were correct.

As for 2C, the Time Conversions. I have also taken a hardcoding approach, however I have also added the ability to test the "whichTimeConversion" and "getFromUnit" using assert functions. By testing these functions I can confirm the inputs and outputs of the "twoC_TimeConversion" function, as in doing so, I can know precisely what information is getting assigned to each variable. Furthmore, hardcoding variables can been seen commented out in the production code as it would be too troublesome to extract the function into it's separate file. Extracting it to another file is possible, however it adds an additional degree of error, as I coudld forget to update the original code, or perhaps end up changing too much of it to the point where it no longer works. Therefore, the hardcoding will remain within the production code, just commented out to avoid running it.

Input	Expected Output
"1410" + h + h	"1410 Hours, 0 Minutes, 0 Seconds"
"1410" + h + m	"0 Hours, 84600 Minutes, 0 Seconds"
"1410" + h + s	"0 Hours, 0 Minutes, 5076000 Seconds"
"1410" + m + h	"23 Hours, 30 Minutes, 0 Seconds"
"1410" + m + m	"0 Hours, 1410 Minutes, 0 Seconds"
"1410" + m + s	"0 Hours, 0 Minutes, 84600 Seconds"
"1410" + s + h	"0 Hours, 23 Minutes, 30 Seconds"
"1410" + s + m	"0 Hours, 23 Minutes, 30 Seconds"
"1410" + s + s	"0 Hours, 0 Minutes, 1410 Seconds"
All other test strings	"0 Hours, 0 Minutes, 0 Seconds"

Test Implementation and Testing

Description of how to run your test code with correct commands. Results of test execution with test success and failures with short discussion of results from part 6 of this assessment, discussion on whether you have attempted to improve your code and new results, if any. You can use screen shots to support your answer in this part. You are supposed to produce a table which shows following information to help you and the marker to check the work you have done. (EP: Equivalence partitioning, BVA: Boundary value Analysis, BB : Black-box, WB: White-box)

To the TestFunctions.java file the user can type `javac TestFunctions.java` to build the file and then `java -ea TestFunctions` to run the code with assertions on. If the user doesn't add the `-ea` the code will run but will not print out any assetion errors as they haven't been

enabled. When running the code, the user is prompted with all the options they can choose and the program will loop through itself until the user chooses to exit or there happens to be an assertion error.

arrayToUpper & arrayToLower

During the testing of arrayToUpper and arrayToLower, there were no issues with the BVA testing code.

checkNumericsExist

During the testing of checkNumericsExist, there was an assertion error at the 47, 48 boundary. The error was happening on the 47 solo boundary, where the output should have been false but it kept providing an error. So I printed out the output to see that it was true. Going back into the Functions file, I had realized that I had used "<" and ">" rather than "<=" and ">=" symbols. However fixing these still gave an error. Afterwards I had realized that I was setting inputArray elements 0 and 1 at the beginning, but at the boundary test I was only setting the 0th element and not the 1st. Therefore it was keeping this value from before. The test code was fixed and the output was to be expected, with no further errors.

Before: `inputArray[0] = 47;`

After: `inputArray[0] = 47; inputArray[1] = 47;`

arrayToNumbers

During the testing of arrayToNumbers, there was an assertion error. This was not from the production code but rather the test code. Since an array of length 2 was initialized, the retaining values that weren't changed would still be counted by the program and loop through during the check. To fix this, I put the arrays of length 2 at the top, then underneath just set the input array to a new array with the length of only 1 to check if the output was correct. If anytihng, this demonstrates that the code is able to pickup all values of the array and will perform the calculations without skipping a beat.

getFromUnit

During the testing of getFromUnit, there were no issues with the EP testing code.

main

During the testing of Menu.java's main, there were no errors. Hardcoding has been commented out.

twoC_TimeConversion

During the testing of there were errors in the calculations. If the input string contained no numbers, the array would return null and could not be parsed as in int. Therefore, to alleviate this issue, I add a quick check before returning the list in "arrayToNumbers", if the string was empty it'd set the string to "0".

```
if(outputString.equals("") == true)
{ outputString = "0" }
```

Another issue that was encountered was when the same unit for convertTo and convertFrom were selected. This is because within the calculations, it would just check if convertTo was one of the 3 possible units, and then an else statement was used to run for the remaining 2. This was an oversight as I did not expect the From and To variable to ever be the same. To fix this I simply added an `else if` statement that was the to check for the char of the same unit. After changing this the code worked as expect, and all the hardcoding commented out.

Module Name	BB EP Design	BB BVA Design	WB Design	EP Code	BVA Code	WB Testing
arrayToUpper	None	Done	None	None	Done	None
arrayToLower	None	Done	None	None	Done	None
checkNumericsExist	None	Done	None	None	Done	None
arrayToNumbers	None	Done	None	None	Done	None
getFromUnit	Done	None	None	Done	None	None
main	None	None	Done	None	None	Done
twoC_TimeConversion	None	None	Done	None	None	Done

Version Control

For version control I have used **git** to keep track of my files. Through the creation of an online github repository, the entire assignment can be accessed online. Through the github repository, it

allows for access from other locations along with making changes and to download and track progress on the project.

Throughout the course of this assignment, I have been incrementally updating the local repository alongwith pushing the changes to the online repository aswell. Everytime I commit a file, I will write the file that I update, and then on the next line with an indent, I'll do a hyphen followed by any changes I've made. Most of the time it's just grammatical errors that I've fixed, which extended to file size, however thanks to this, I am able to track my progress. The only file that doesn't have any comments is the .DS_Store files, which are macOS files that just track the directory stored for each folder and it's files. Therefore interacting the files in any way will cause these files to update. The class files and a test.txt file are ignored as they are just temporary files that can be removed without any detrimental issues to the entire project. While the test.txt file can be removed, if done so there will be no file to test the program's ability to read a file.

After completing the assignment, I will export the git log into a text file and store it within the Documents folder. It should be named "Git Log.txt"

Ethics

Ethics and professionalism are very important when it comes to usage and operation of my code. While the code is up on github for anyone to view and download, it should not be directly copied and pasted, and used without understanding. Most of the code is very situational is built in a way that each function utilizes each other. To take the code and then attempt to produce new code without a solid grasp on the operation could result in the loss of time and money. The code produces a lot of strings, instead of integers to avoid input mismatch errors, along with working in ascii, therefore if someone were to just copy the code and insert what they thought to be regular integers where there are ascii values, it could potentially cause large calculation errors when it comes to the time conversion program. This could not only harm those who take the could but could also harm me, as if the person who takes my code, points towards me saying that the issues were solely my responsibility, I could suffer from financial, social, and reputational damage as everyone would have a distrust in me and my programming abilities. Therefore it is highly important that we follow ethical and professional practices.

In these ethical guidelines, it discusses how we need to act with honesty and maintain technical competence. For my take on how to avoid ethical and professional issues in my software, I'd like to focus on these two points.To show honesty in my work I would propose to plan ahead and begin working on the assignment as soon as possible, to avoid leavin this project to the last minute. This is due to the fact, that leaving such a large assignment to the end nearing its due date, will put more time pressure onto myself, possibly promoting unethical acts, such as stealing other work, and overlooking important fucntionalities. Furthermore, as for the maintenence of techincal competency, I would suggest leaving comments in my work to show an understanding and explanation for my code. By commenting I wish to show that I do have technical competence over my own work and that I wish to make it easy for another software engineer to read and follow along with my work. By showing a technical understand of my own code it helps me understand what I have written and gives me opportunities to react to any oversights that could negatively

impact others. This will also show that it is all my original and non stolen work. Hopefully, by leaving comments and planning ahead, we can avoid any ethical issues, such as pilferring of code and dishonesty. In doing so, we can avoid any unneccessary time pressure from building up and causing one from acting unprofessionally and unethically.

Discussion

Reflection of your own work, ways to improve your work and any other thing you wish to present.

Honestly, I didn't manage to fully program my code, as per described in the preliminary description. In my description I wrote that for 1C I would be able to distinguish between decimal values, and values with commas, however during the programming stage I realised that the code I had designed was not as flawless that I thought. After realising these flaws and that it would take a lot more effort to code than I would like, I decided to exclude this as it would be very difficult to code and very difficult to test it.

Although there was some functionality that was lost I'm actually still happy with all the other programs, as they run to my expectation. Testing was rather difficult and I'm upset that I wasn't able to test all the functions, since most of them did not return any values. In the future I would plan more towards this and only have a limited amount of functions or if possible, only one (being the main) that doesn't return values. Tetsing the black-box cases was done, however not extensively, since most of the testing was only done with either boundary value analysis or equivalence partioning, and not both. Next time I would attempt to try and have enough test cases where I could both testing methods for my black-box testing.

Overall, I believe I greatly underestimated the difficulty of this porject, however I managed to adapt and complete the work up to my personal standards. However, completing this has only heightened my future expectations and standards, and broadened my understanding of software design and engineering.