The CKsum application is one of my first personal c++ projects. Its purpose is to calculate the MD5 sum of every file In a directory, record it, and then compare it to another instance of the directory to confirm it copied correctly using the least amount of non-standard functions.

The entire project took about 4 days to complete with all its options and features.

Day 1 (4hrs): The first day I put together a basic program path; The user would choose whether to check two files now, check one file and output the data in a txt file, or compare a file to a txt file. Then the program collects the directory path and assigns it to a filesystem::path object which is used to create a recursive_directory_iterator. This iterator will then be used in an asynchronous for loop that will feed each path to a function that will use the system() function to calculate and return the MD5 sum to a txt file.

After writing the code I realized that it would be easier and faster to save the path names and results in an unordered_map, and then iterate through each to compare. I also realized that the system() function only returns the statue of the command so after researching a solution I decided to use a C style pipe function to execute the command.

Day 2 (2.5hrs): The second day I started debugging the program. The first issue was that the use of the std::async function which would occasionally throw a xhash file exception with a minimal boost in speed, so I decided to omit it for now. Another issue was that directory paths with spaces wouldn't get processed correctly so I used std::getline() instead of cin. However the getline() function would get skipped due to the cin stream operators in the main menu selector; To fix this I just put cin.ignore() before the getline(). However, fixing this problem let to a new one where creating the second iterator would always lead to an exception even if the code was exactly the same as the first; So I decided to step in the filesystem header and cpp file, recoding the changes in the relevant variables.

Day 3 (4.5hrs): The third day I continued looking through the files and found out that the exception was thrown because the path failed to generate a handle even with all the same data and options. However after looking closer I realized that the path of the second directory didn't have a drive letter; which was due to using cin.ignore() a second time. After fixing this the program ran smoothly on small directories but failed to line up the same files against each other in larger directories. This was a quick fix, I just iterated one map instead of both and used the path of the first map to get the sum of the second and then compare the two sums.

At this point the main path of the program was done so I started working on the path that would take a txt file for a previous check to compare to, and a path to generate this file. To reduce the amount of redundant code the directory iterator loop was put in a function of its own; and the rest of the time was spent figuring out the best custom file structure, isolating the file name and sum code. Along with figuring out how to read it properly.

Overall, the project took about 11 hours including research, writing the code, and an additional 45 minutes to write this report.

CKsum version 2.0

Later on, I decided to add additional features to the project such as a GUI, simplifying the code, and attempting to use the async function again; Along with the ability to check different directory versions.

Day 1 (7hrs): The first day I would research ways to add a GUI only using c++; Deciding it would be best to use the WxWidgets library in this case. Then I created a new file, and constructed a basic window for the program while learning the library. This took longer that expected due to the cluttered documentation and lack of examples, but I was able to use visual studios tools to parse through member functions to build a compact window that could display instructions for each option in an intuitive way.

Day 2 (7hrs): The second day I added the essential code of the original program, editing the syntax to what was appropriate to that library. I decided it would be best to put the code in a separate class to better organize the GUI elements from the Computing elements. The main problem today was that visual studio suddenly stopped recognizing any elements, and returned errors for some, once the old code was pasted in. To fix this I went restarted the program, went through the properties, and restarted the computer but the problem persisted. Eventually I discovered commenting that reference wrapper lines and writing the types without auto fixed the issue. After finding the fix I was able to easily integrate the code and have the app fully working. The only issue now is that, although the code works perfectly, the pipes _popen command in the old code would open a CMD window for every file it checked which could be irritating to the user.

Day 3 (8hrs): The third day I researched a way to fix the pipe command and found that if I use C's WinAPI I can set the CMD window to open minimized; However this required a deep understanding of C style programing and pipes, and researching this took most of the time. After finally finishing this the ReadFile() function would hang and the program would crash. Unfortunately Visual Studio would not let me step in to the function, so all I was able to determine was that the function did successfully return a few lines from the pipe; However there were lines or random corrupted data inserted at the end of each read pass. At this point I was unable to determine a way to fix this so I decided to revert to the previous pipe method and solve this problem at a later date

Day 4 (2hrs) The fourth day I began cleaning up the old code and implemented better algorithm structure throughout the program while adding comments.

In conclusion this upgrade took about 24 hours and 45 minutes to report.