**How to Perform an Hours Audit Using the Auto Hours Auditor Program**

Prepared by: Quoc-Liem Vu, Assistant Director of Alumni Administration, COPE Health Scholars, Swedish Medical Center

Last revised: April 2022

**Summary**

This instruction manual teaches alumni coordinators (or other helpers) to perform an hours audit manually and automatically using the Auto Hours Auditor Program. In addition, this document contains troubleshooting information and appendices to understand the code further. Appendix A has a quick explanation of the organization of files, Appendix B has an advanced explanation of how the code works and Appendix C has some information on the added Batch Auto Hours Auditor Program. Finally, this document will use my hours as an example.

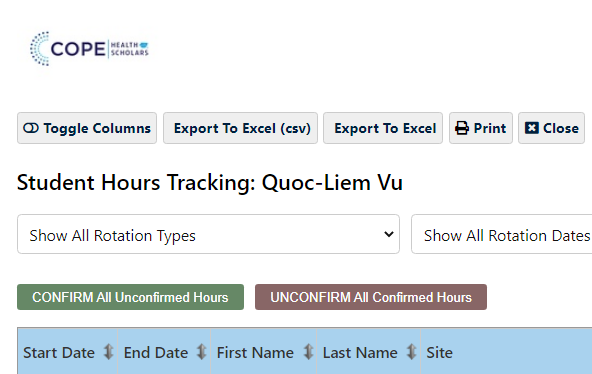
**Helpful Information**

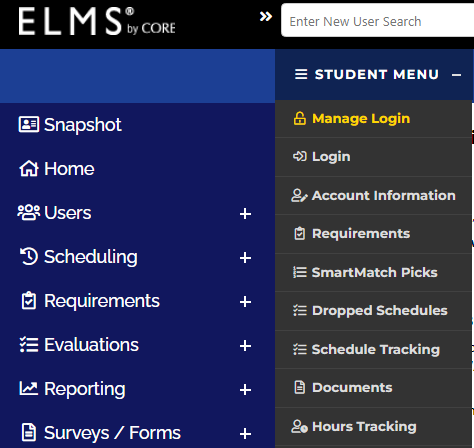
* A computer/laptop with an internet connection is required to perform an hours audit. If not available, computers are available for use in the COPE Health Scholars Office in the Heath Building at the Swedish Medical Center First Hill Campus.
* No special computer knowledge is required to perform the audits. However, if the program requires heavy revision, Appendix B will provide a high-level overview of the program’s function and is meant for people with knowledge of Java programming.
* Scholar information is confidential, make sure to not leak information and apply similar HIPAA practices.
* If an error occurs during any process and there is no information on fixing it, consult the ADAA for help.

**Original Method: Performing a Manual Hours Audit**

**Step 1: Gathering Scholar Data**

The first step in an hours audit is to gather the data. Usually, the ADAA will divvy up which scholars to audit in the graduation tracker file (Found in X: GlobalShare/Health Scholar/Healthcare Systems/Swedish Seattle/1.0 Administration/Alumni/Graduation trackers/). Outlined below are the steps to perform this.

1. *Log in to your admin CORE account.*
2. *On the above search bar, look for the health scholars account.*
3. *Under the dropdown “STUDENT MENU,” click the “Hours Tracking” tab.*
4. *Click the “Export to Excel” tab.*

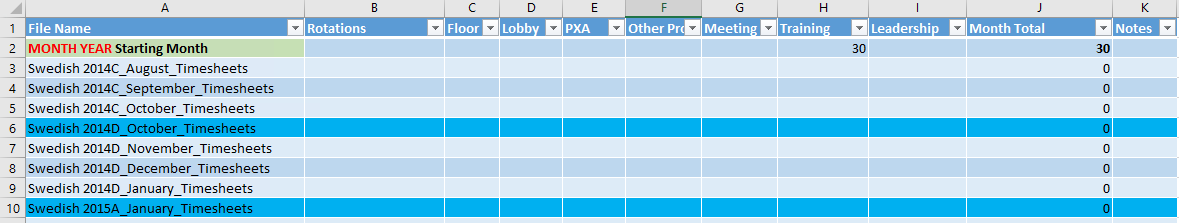


**Figure 1.** CORE website with highlighted locations for the “Hour Tracking” tab and the correct “Export To Excel” button.

**Step 2: Performing an Audit**

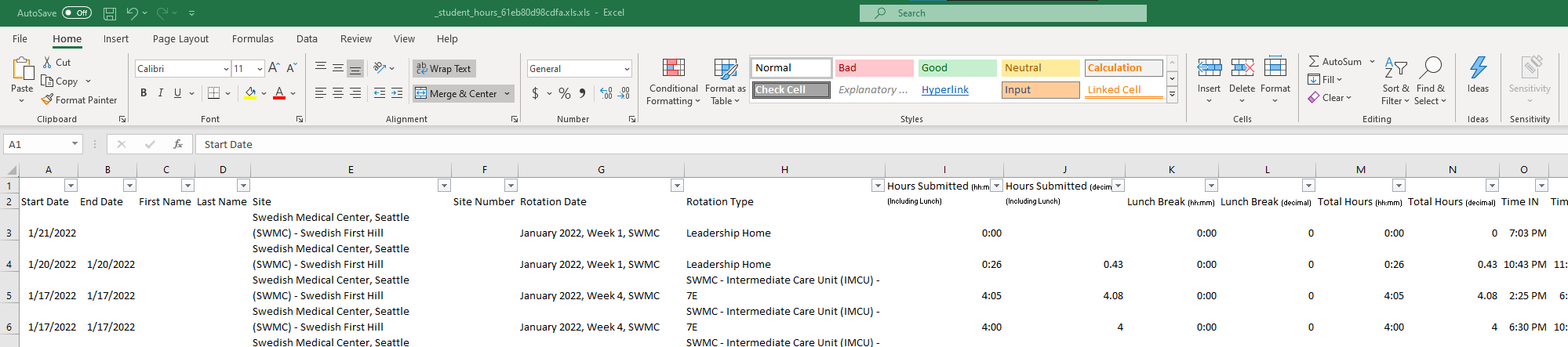
After getting the scholars’ data, now it is time to create the hours audit sheet.

1. *Get the template hours audit sheet from the “Manuals and Templates” folder.*
2. *Edit the “File Name” tab to match the scholar’s rotation dates.* *Make sure to highlight the starting month row in green, the months are alternating blues, and the start of a new rotation is a deep blue.*



**Figure 2.** The beginning part of the template hours audit sheet.

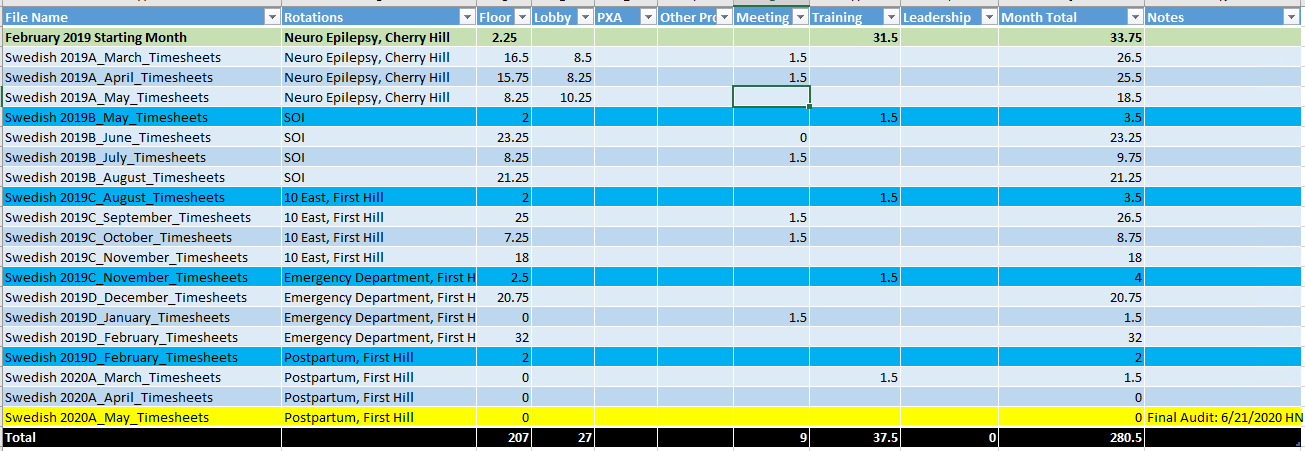
1. *In the scholar’s hours data file you got from CORE, select the “Sort and Filter” button.* Figure 3 shows this button highlighted in red.
2. *Filter the rotation dates by months starting from the beginning.* Figure 3 shows this highlighted in blue.
3. *Optionally, you can further filter the rotation type.* Figure 3 shows this highlighted in green.



**Figure 3.** An example hours data sheet with important buttons highlighted. The red square shows where to find the sort and filter button, the blue square shows where to sort by date, the green square shows where to sort by type, and the purple square shows which hours column to use for summation.

1. *Click and drag over the squares in the “Total Hours (decimal)” column of the same type of hours, and on the bottom right, excel will tell you the sum of those hours.* Figure 3 shows this highlighted in purple.
2. *Insert that sum in the correct column and month in the template sheet.*
3. *Fill in the “Rotations” column with the assigned department for that month as found in the Rotation Type column.* Figure 3 shows this button highlighted in green.
4. *After filling in every month for the scholar, highlight the final month row in yellow.*
5. *In the Notes column of the yellow highlighted row, fill in with the template “Final Audit: MM/DD/YYYY YOUR\_INITIALS”.*

After completing the above process, your document should look like Figure 4.



**Figure 4.**  Example of a completed manual audit.

**Step 3: Finishing Up**

After creating the hours audit sheet, now it is time to do the finishing steps.

1. *Rename the hours audit sheet in the form “LASTNAME\_FIRSTNAME STARTINGROTATION Hours Summary”.*
2. *In the X: GlobalShare/Health Scholar/Healthcare Systems/Swedish Seattle/1.0 Administration/Alumni/Alumni Cohorts/2021D roster/2021D Graduate Hours Audits folder, create a folder with the template “LASTNAME, FIRSTNAME”*. Note: the example file location uses 2021D.
3. *Place your hours audit excel file into the created folder.*
4. *Update the Graduation Tracker with the hours.*

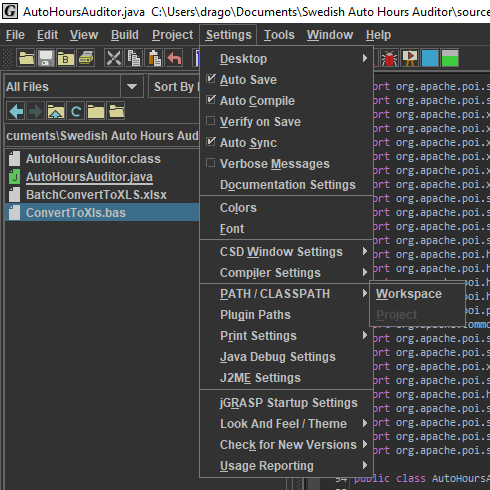
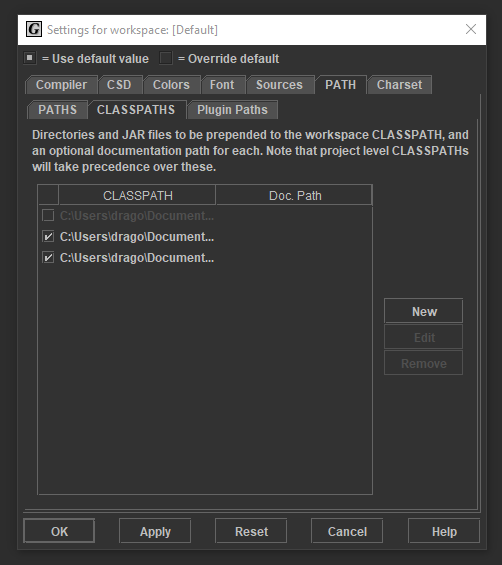
After finishing with your assigned scholars, make sure to send the initial approval/denial email. You can find the email templates and this manual in the “Manuals and Templates” folder.

**New Method: Performing an Hours Audit Using the Auto Hours Auditor**

**Getting Started: Downloading and Setting up a Java IDE**

Before using the Auto Hours Auditor, you will need a program that can run Java programs. Java IDEs are programs that can run other java programs; this guide uses jGrasp for the examples, but this can work for any Java IDE.

1. *Download a Java IDE.* Below is a link for jGrasp. Make sure to click the latest .exe download.
   1. <https://spider.eng.auburn.edu/user-cgi/grasp/grasp.pl?;dl=download_jgrasp.html>
2. *Get the “Auto Hours Auditor.zip” from the ADAA and extract it into your computer.*
3. *From the jGrasp program go to Settings >> PATH / CLASSPATH >> Workspace*
4. *A window will open from the PATH tab; click the CLASSPATHS subtab.* Figure 5 shows this process.



**Figure 5.** jGrasp program with highlighted tabs to find the CLASSPATH.

1. *Click on the “New” button. A new window will pop up, and then for the “Directory Path or JAR File” block, click on the “Browse” button. Another popup will occur and navigate to the APIs folder and for each folder (there should be 2). Click on them and click the “Choose” button on the bottom of the newest popup.*

**Step 1: Gathering Scholar Data**

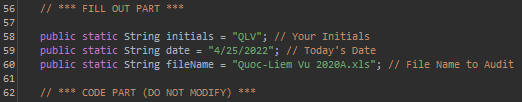
The steps to gather the scholar data are the same as the manual method, except that you must grab the data for ALL your assigned scholars first, as well as some additional steps.

1. *After downloading one file, rename it to the form “FIRSTNAME LASTNAME STARTROTATION”.* Their start rotation is found on their CORE page under “Class of: “
   1. *If the Scholar is a* ***JHS****, make sure to name the file in the form “FIRSTNAME LASTNAME JHS STARTROTATION”.*
2. *After downloading all the hours audit files, place them in the “In Progress Audits” folder.*
3. *Open each of the audits, click “Enable Editing” then in the “Save As” tab, resave the file into a “.xls” file. A warning pop up will occur; click continue.* The .xls version from CORE is outdated, so resaving it will change it to an updated “.xls” format.

**Step 2: Performing an Audit**

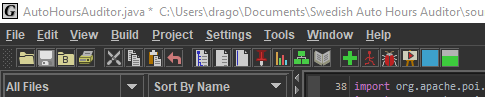
After gathering all the hours data, now it is time to perform the audits.

1. *Under the “\*\*\* FILL OUT PART \*\*\*” fill in your initials, the current date, and the file name.*



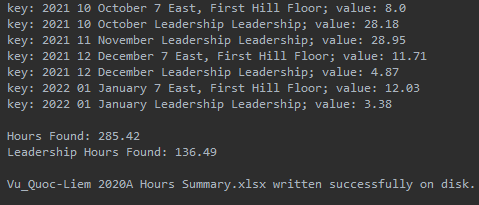
**Figure 6.** Area of the code to fill out.

1. *Click on the “Run” button, which will start the program.* Figure 7 highlights the button.



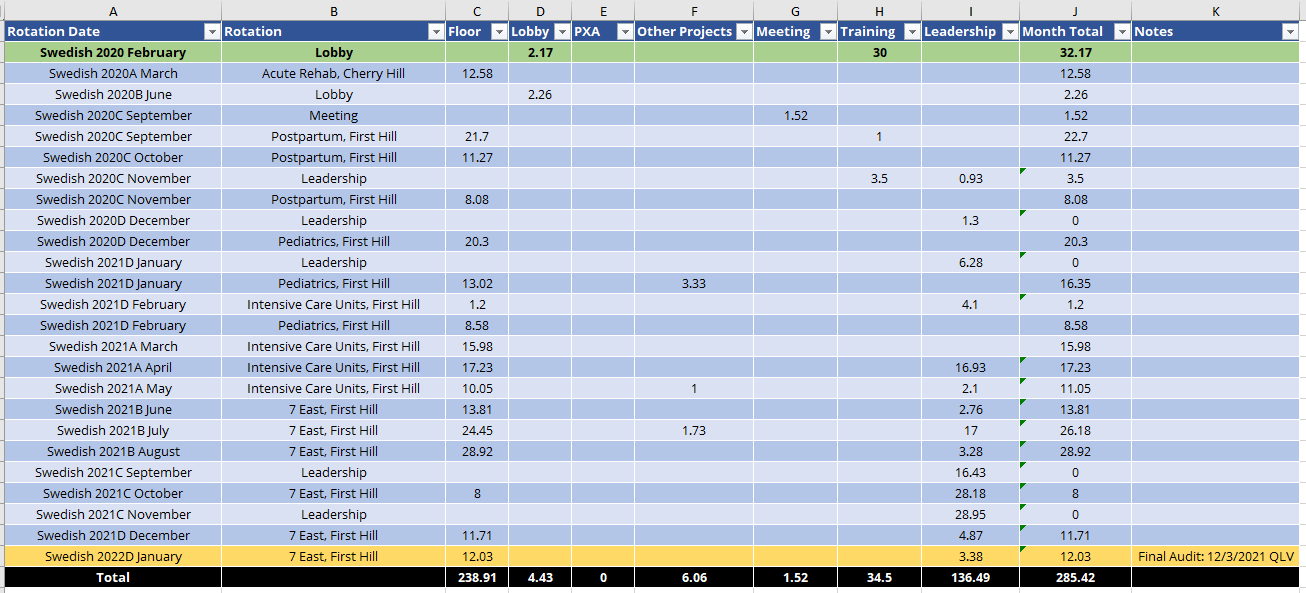
**Figure 7.**  jGrasp program with highlighted “Run” button.

1. *The program will output all the hours found and summarize total floor hours found and leadership hours found; use this information for the “Graduation Tracker”.* Figure 8 shows an example output. There is error checking built into the program, and you will be notified of errors in the output message. The program will still output a file, but make sure to fix it! Check the Troubleshooting section for help.



**Figure 8.**  An example successful run through output.

The file will appear in the “Completed Audits” folder. Figure 9 shows an example of what the completed one looks like.



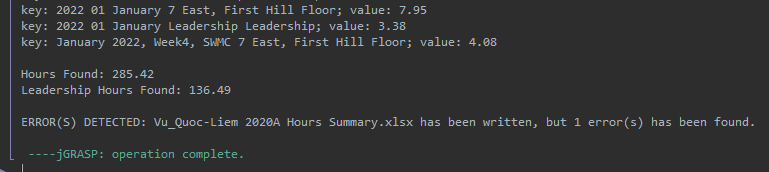
**Figure 9.**  Example of the Completed Audit. *Note: a redesign was used for the automated audits.*

**Step 3: Finishing Up**

After completing all of the audits, send them to your COPE email, so you can put the files in the Swedish cloud servers using the remote desktop. Then, perform the same finishing up steps used in the manual audits section.

**Troubleshooting**

The program has built-in error checking to tell you what went wrong. Here is how to fix the common mistakes that the program catches. The details of the error can be found by looking through the output. In the example shown in Figure 10, the program encountered one error. Therefore, I should see one descriptive error message.



**Figure 10.**  An example run through output with an error.

**ERROR: Typo OR Transfer Scholar found in data**

In a typo error, simply look through the excel document and fix the typo in labeling the rotation type. This error also occurs if a scholar from another location transfers to Swedish and their hours are in CORE. In that case, manually add them to the audit sheet.



**Figure 11.**  Example of the typo error message in the run through output.

**ERROR: Detected Old Hours**

In an old hours error, the scholar shifted before COPE shifted into CORE. This error is becoming increasingly rarer since most health scholars are newer and will eventually fade out. In this event, ask the ADAA where you can find the old hours and manually add them to the audit sheet.

**ERROR: Unknown Rotation Type**

In an unknown rotation type error, there could be a typo in the excel sheet. In that case, edit the excel document to fix the error. Another reason could be that the Auto Hours Auditor program did not include a newly opened COPE department. In that case, inform the ADAA, and they will send an updated Auto Hours Auditor program.

**ERROR: Unmatched Hours Found**

In an unmatched hours error, there could be many things gone wrong. The best solution is to look through the hours data and the program’s output audit and find inconsistencies. Also, it would help if you informed the ADAA to fix the bug.

**Conclusion**

Congratulations on finishing a round of audits! To summarize, hours auditing can be broken down into three parts: gathering the data, processing the data, and finishing up. The data gathering and finishing up part are similar for both hours auditing modes, but with minor differences. The longest part is the processing section, but the Auto Hours Auditor program will cut that part down from hours to minutes. If you would like more information, Appendix A describes the file structure and Appendix B provides a high depth description of the program. If you have any problems and issues, make sure to use this document as a reference or contact your ADAA.

**Appendix A: Organization of the Files**

Understanding the organization of files could help demystify what is happening but is unnecessary.

* *Auto Hours Auditor*: The “master” folder that holds all the other folders.
* *APIs* (Application Programming Interface): Contains the Apache POI, which allows us to use and manipulate Microsoft files like excel. As well as the Apache Commons Mathematics POI, which is required to use with the Apache POI for functioning.
* *Completed Audits*: A helper folder to place the completed hours audits.
* *In Progress Audits*: A helper folder to place the extracted data from CORE.
* *source*: Contains the Auto Hours Auditor programs for use.

**Appendix B: Layout of the Program (Advanced)**

This section will provide a high depth overview of the program’s function meant for people with knowledge of Java programming. If heavy revisions to the program’s functioning are required in the future, this will be a helpful reference in understanding the code. In addition, the code has lots of comments for clarity so this document will serve as a more detailed reference.

The code works procedurally and is broken into three parts: importing and extracting the data, processing the data, and exporting the hours audit.

**Part 1: Importing and Extracting the Data**

In part 1, the code uses Apache POI’s functions and uses the map data structure (keys and values) to store the data from the .csv hours file. We initialize the column and row counter integers to know which column we are in and ignore the data file's first title row. Also, the strings to be concatenated to form the key and the double value are initialized. Finally, A “problem” integer is initialized to notify the user of the number of errors found throughout the program.

1. A while loop is used to parse through the data and initializes a row and cell iterator, a column counter, and a Boolean JHS data error. The JHS data error becomes true when JHS data is mixed with HS data and aims to separate them.
2. Another while loop is used to go through a row and uses if-statements for each separate column. Most of the error checking is in each column and can be found in the if-statements. Each if-statement reads what is written in the column, splits the information into parts, then forms a useful string. The comments explain the details of each column.
3. At the end of a row iteration, the key is formed by concatenating the rotation date, rotation type, and hours type strings and paired with a key of the hours associated with that row information.
4. After every row is read, we check if the .csv hours file contained the 20/30-hour training from the start of training. If we do not find this information, the program manually inserts it.

**Part 2: Processing the Data**

In part 2, the program processes the data and creates the excel file to be outputted. In this section, the formation of the header row, hours rows, and the final row are separated by comments for clarity.

1. The program initializes two double variables, each representing the total floor and leadership hours.
2. A for-loop is used to read and print each key and value in the data map and sums the hours for each type. At the end of the for-loop, the program prints out the hour information.
3. The new excel workbook, sheet, and colors used for the file are initialized.
4. **In creating the header row,** the program uses an array list data structure that stores all the column titles.
   1. A cell style is created for the header. Cell styles contain font, border, color, and formatting information.
   2. Afterward, the columns are filled in with the titles. Finally, the column widths are adjusted to fit the title lengths for a cleaner look.
5. **In creating the hours rows,** the program creates a new font and three cell styles: two for the alternating blue color look and the final yellow highlighted column.
   1. The number of rows to create and the counter for the rows are initialized. Two strings representing the rotation key and the hours type are initialized. An Apache POI evaluator is created that solves formulas is initialized (Used for the final row summing up the above hours). A month locate integer is initialized to place hours of the same month in the same row and a Boolean repeat month variable is initialized to assist the month locate function.
   2. A for-each-loop is initialized that goes through every key in the data map. The loop begins with splitting the key into a string array to separate all the information. Next, the program checks if the key’s month is already placed in the excel file. Then the rotation date and hours type are stored by reading the previous string array. Finally, the hours data is read by reading the value associated with the key.
   3. The next step is creating an if/else statement for if the month is a repeat or not. If the month is not a repeat, another for-loop initializes to place data in each column. If the month is a repeat, the program locates the repeated month row and depending on what is unique about the hours type in the key, inserts the information.
   4. Finally, the final total hours row is summed up after each run and the iterator values are updated.
6. **In creating the total hours row,** the program begins by creating a new cell style and font.
   1. The program begins by using a for-loop to get the location of the second to last row.
   2. Afterward, a for-loop is used to reset the cell style to the highlighted yellow and signs the document with the user’s initials and date.
   3. Next, a string array list is used to store the formulas to calculate the total hours for each column and the extraneous labelling.
   4. A for-loop is used to iterate over each cell to fill in the information and set the cell style. An if/else statement is used to skip the first, second, and last column as they are labels, while the other columns are calculated.
   5. The last cell that contains the sum for all the hours is evaluated to place its value. An error check is used here to see if all the total hours are accounted for and not missed by the code.
7. After creating the rows, the program adds the rotation letters to the months.
   1. First, the program obtains the starting rotation letter from when the scholar started and the start month from the data. A string array of every month is initialized.
   2. A for-loop is used to find the start month and stores that as a number. (1-12)
   3. A char array is initialized for the rotation letters (A-D) and is used in a for-loop to find the starting rotation letter and creates a numerical pointer to that letter.
   4. A string array is initialized to hold the letters associated with each month. Also, a month pointer, a reset pointer, and a letter counter are initialized as integer variables. A Boolean reset variable is initialized in the event of the first reset. The first reset occurs if the starting month is not January, so the pointer resets to get the months beforehand.
   5. A for-loop is used to insert the letters to the month-letter starting from the scholars’ first month using the letter pointer. Next, an if-statement is used to check if the program has reached December, and if so, reset the pointer and replace the Boolean reset variable with true. Finally, we insert the associated letter to the associated month in the month-letter array.
   6. Next, the program updates the letter count iterator, and if we have placed the same letter three times (three months in a rotation), reset it to zero and update the letter pointer to the next letter. If the numerical pointer is equal to four, reset the number to zero. (‘D’ -> ‘A’). If a reset has occurred, increase the reset pointer by one. The reset pointer is now the new month pointer.
   7. Finally, the program inserts the letters to the months in the excel sheet. Two strings are initialized to hold rotation information and a string array to hold the information already inserted. The rotation information is grabbed and split up in a for-loop that spans all the rows to isolate the month information. If-statements are used to find which month is present and concatenates the rotation letter. Afterward, the rotation string is re-pieced together and inserted back into the sheet.

**Part 3: Exporting the Hours Audit**

In part 3, the created excel sheet is exported into the master folder.

1. The file name is created by concatenating all the provided scholar information, and the excel file is outputted with the name.
2. If there were zero issues throughout the program, a “successfully completed” statement is outputted. However, if there were issues, an error warning is outputted.

**Appendix C: Batch Auto Hours Auditor**

I created a new program that branches off the original Auto Hours Auditor that speeds everything up again. Where the Auto Hours Auditor does one audit at a time, you can just do Step 1: Gathering Scholar Data for each scholar you need to do and click “Run” to do every audit at once. Make sure to check the logs if an error occurred however!

If you need to update this code as well, I essentially kept the original code but turned it into a for-loop to iterate through each file.