Restoration Architecture

1. Create a counter to find the number of rows in the corrupted file

---

2. Get a line from file using “readaline” function
    a. Increment the widthCounter by 1
3. Create a function “getSequence” to get sequence of ASCII characters in line
    a. Loop over line and add all non-digit chars to an Atom structure so we can compare “strings” of non-digit sequences.
        i. We will use the “Atom_new” function to create these structures and pass the line from “readaline” (*datapp* variable) into the first argument and the line length returned from “readaline” into the second argument
4. Store the Atom sequence in a Hanson Table as the key and the value will be the corrupted line (*datapp*) the sequence came from
        i. The void * pointer argument for cmp and hash in the “Table_new” function will be set to NULL since we are using Atom strings.
    a. The function we will use to add data to the table is “Table_put” which will take the void *’s Atom string (as the key) and *datapp* (as the value).
        i. If there is a repetition of sequences, “Table_put will replace the current value with the new value given to it. It will then return a pointer to the line where we will store that line in a Hanson List of char *’s (“strings”).
        ii. We are using a Hanson List for constant time addition to the back of the List since the corrupted lines are stored sequentially.
5. We will repeat steps 2-4 until we reach the end of the file

---

6. We will then use the “getSequence” function to get the sequence of non digit chars from the list holding the original rows of the pgm file.
7. Using the sequence, we will call “Table_get” and send the sequence for the void * argument. This will ensure that we copy the last original line from the pgm file in the Table into the List containing all the other original lines.

---

8. Create a function, “removeNonDigit”, which will remove all non digit chars from the corrupted lines.
9. Loop through the list containing the original rows and run each index of the list through the “removeNonDigit” function.

---

10. Now we send the list containing the uncorrupted lines to a function, “imageBuilder,” which will print the header of the raw pgm, whose information we have collected throughout the “restoration” function, and the uncorrupted lines from the list.

---

11. Deallocate any memory used on Heap

Implementation and Testing Plan

1. Create the .c file for the restoration program. Write a main function that spits out the ubiquitous “Hello World!” greeting. Compile and run. **Time: 10 minutes**
    i. ***Test*** - Compiling and running the code to test if the make file has been correctly made for the file. The expected output should be the “Hello World!” greeting in the terminal.
2. Create the .c file that will hold the readaline implementation. Move the “Hello World!” greeting from the main function in restoration to your readaline function and call readaline from main. Compile and run this code. **Time: 10 minutes**
    i. ***Test*** - Compiling and running the code to test if the make file has been correctly made for the file and to also test if both files have been linked correctly. The expected output should be the “Hello World!” greeting in the terminal.
3. Use the CS40 Idiomes Page on the Course Website to build a file opener function. **Time: 15 minutes**
    a. This function will open files from the command line and output any errors to stderr.
    b. If something that prevents a file from opening is encountered (i.e filename doesn’t exist, memory couldn’t be allocated, etc) then the appropriate error will print and exit the program.
        i. ***Test*** - Compile and run the code with various filenames. Some file names will be valid and the program should output a success message in a print statement located at the end of the function.
        ii. ***Test*** - Compile and run the program with invalid file names to see if the appropriate error message prints to stderr and the program exits.
4. Use the arguments from the main function in the restoration program to build an error checker for input passed from the command line in the terminal. **Time: 10 minutes**
    i. ***Test*** - Compile and run the program with no extra command line arguments to see if the program outputs the correct error message
    ii. ***Test*** - Compile and run the program with too many command line arguments to see if the program outputs the correct “too many argument” error message to stderr
    iii. ***Test*** - Compile and run the program with the correct number of arguments and check if a success print message, located after the error checker, prints.
5. Build the limited version of the readaline function. This version will be able to read lines of length 1000 or less. **Time: 40-60 minutes**
    a. Create an int widthCounter variable and a char array to hold the chars from the file.
    b. Function will use a while loop with the condition that the current character read is not “\n” since we know that all lines end with the new line character.
        i. ***Test*** - Print the current character to the terminal to ensure that the while loop is stopping at “\n” and not reading one character less than it should.

    c. While the loop is running the characters from the file will be stored in an array of size 1000 which simulates a line from the pgm file
        i. ***Test*** - After the while loop is done running, a print for loop will be used to ensure that all the appropriate characters are inside the array. The output of this should match the respective line when using the “cat” command on the original file send to readaline
    d. The char *datapp will point to the front of this array
        i. ***Test*** - The same test as the one above will be run but instead of using the variable for the array the datapp variable will be used to see if the pointer is working correctly.
    e. widthCounter will increment by 1 and the next character will become the current character
        i. ***Test*** - The widthCounter and current character will be printed to the terminal to see if they are incrementing as they should be. The expected output should match the respective lines in the original file when using the “cat” command.
    f. readaline will return the widthCounter and indirectly the line read through the datapp pointer once the loop has ended
6. Extend restoration to print each line in the supplied file using readaline. **Time: 30 minutes**
    a. Create a while loop that will continue calling readaline until the end of the file has been reached
    b. The returned int value including the elements of the line passed back from readaline will be printed to terminal
        i. ***Test*** - The printed values on the terminal will be compared to the original files using diff. There shouldn’t be any output from diff testing.
7. Build the getSequence function **Time: 25 minutes**
    a. Using the Atom_length method to get the length of each line, the line will be parsed for non digit characters using a for loop.
    b. The non digit characters will be added to a char Hanson Sequence.
    c. Once the line has been parsed the Hanson Sequence will be made into an Atom and returned to the restoration program
        i. ***Test*** - Once the line’s non digit sequence has been returned, the line and sequence will be printed to the terminal. The sequence should display all non digit characters found in the corrupted line.
8. Route the output from readaline and the getSequence functions into a Hanson Table (findOriginalRow function) **Time: 60 - 90 minutes**
    a. A list that will contain all the original rows of the pgm file will be created.
    b. A rowCounter variable will be created and a while loop will run until the end of the file has been reached.
    c. During each loop, the readaline function and getSequence functions will be called once.

    d. The keys of the buckets will be set to the returned Atom from getSequences and the values of the buckets will be the corresponding corrupted lines from readaline.
        i. ***Test*** - Using the Table_get method we can print to output of the table to the terminal and it should match the sequence and line that were put inside it.
    e. If a non digit sequence is repeated in the Hanson Table, the table will return the current value in the table and replace that value with the new corrupted line inputted.
        i. ***Test*** - We will run the getSequence on the returned line from the Table and print it as well as the line from the list. Then we'll compare the sequence from the line put inside the List to the sequence of the line that replaced the old value in the Table. They should be the same.
    f. The returned corrupted line will then be stored in the List created which houses all the original rows of the pgm file.
    g. The rowCounter variable will be incremented by 1 and the loop will start again.

9. At the end of the findOriginalRow function we need to ensure that the corrupted lines in the hash table aren't original. So, we call the getSequence function and send the output into the Table_get method. The returned line will then be pushed onto the List of original lines. **Time: 20 minutes**
        i. ***Test*** - Using a for loop, we'll run through the entire list printing the returned Atom strings from getSequence. The output from all the lines should be the same.
10. Create a function removeSequence which will parse through the entire List of original rows using a double for loop. **Time: 30 minutes**
    a. The first for loop will parse through all indices of the List and the second will parse through every character in the line stored in that specific index.
    b. A Hanson Sequence will be created outside the lists to house the characters
    c. The inner for loop will then check if the character is a non digit one. If it isn't then the character will be pushed onto the Hanson Sequence
        i. ***Test*** - The character pushed onto the Hanson Sequence will be printed to the terminal. The output should display a numeric character.
    d. The current character will then be incremented.
    e. After the inner loop has finished iterating, in the first for loop, the Hanson Sequence will be converted to an Atom String and replace the current line in the List of original rows.
        i. ***Test*** - The replaced line in the List will be printed to the terminal. The output should contain all the numeric characters of the original corrupted row.
    f. The Hanson Sequence will be emptied and the first loop will iterate.
        i. ***Test*** - The length of the sequence will be printed and it should output 0.
11. Build the printRaw function which will retrieve the information from the List **Time: 10 minutes**

a. The function will output the raw pgm file type to the terminal.
b. The widthCounter returned from the readaline function will be outputted to the terminal
c. The rowCounter will be printed as the height to the terminal
d. The List of uncorrupted rows will then be looped and printed to the terminal
    i. ***Test*** - Redirect all terminal output into a file and run the Pnmrdr module to see if the interface is able to read the raw file. The badformat error should not be outputted.
    ii. ***Test*** - Use the pgm reader extension on vscode to see if the file is able to be read. The extension can only read valid pgm files so it should output an image only when the file is valid.