Jack Adkins (jadkin05) and Krish Dhayal (kdhaya01)

Restoration Architecture

Data Structures

- Dynamically Allocated Array #1
    - Will store the “restored” pgm with grayscale values and whitespace
    - Assembled row by row as each corrupted row is restored
- Dynamically Allocated Array #2
    - Will store individual rows of the pgm as we restore them
    - Once restored, will be inserted into the full array then cleared for the next row
- Pnmrdr_T struct
    - Used primarily in readaline function
    - Will give us access to a pointer to a struct containing the read in corrupted pgm file
- Custom struct
    - Char pointer (starts as void but eventually to the start of the correct sequence)
    - 2D Array of all the infusion sequences collected from the file

Implementation Plan & Testing Plan

As referenced in spec (edited for personal use):

1. Create the .c file for your restoration program. Write a main function that spits out the ubiquitous “Hello World!” greeting. Compile and run.
    - Time: 10 minutes
    - Testing:
        - See the program compile and print a statement
2. Create the .c file that will hold your readaline implementation. Move your “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
    - Testing:
        - Make sure each file is included in the Makefile instructions and that again we see it compile and print a hello world statement
3. Extend restoration to open and close the intended file, and call readaline with real arguments.
    - Note: How to read in with c
        - No “infile/outfile” available
    - Time: 10 minutes
    - Testing:
        - Use booleans to check if the file is open
        - Insert temporary statements into readline that make sure the given arguments can be accessed
4. Build your readaline function. Extend restoration to print each line in the supplied file using readaline.
    - Simple Version
        - Set a void pointer to the first index in an empty array
        - Read file until a newline symbol, store that line in the array accessed through our void pointer
        - End function at the end of a line or eof
        - Use the Pnmrdr_T struct and its included functionality
    - Testing:
        - Make sure all out pointer within the struct are pointing at their intended data
        - Specific valgrind checks to make sure nothing is lost when passing from readaline into our restoration functions
    - Time: 3 hours
5. Route the output from readaline into the Hanson data structures that you have selected for your restoration implementation.

- Create a function that creates a struct that holds an array of all the sequences from every line and a pointer that will eventually point to the sequence we determine to be the infusion sequence for the originals
- Create a function to go through the array within the struct and find the “correct” infusion sequence for the original lines
    - Double for loop and compare until a duplicate is found
- Using the newly found correct infusion sequence begin the restoration process
- Determine if the line is from the original or fabricated (contains correct infusion sequence)
    - See testing below for other methods besides infusion sequence to find fabricated lines
- If fabricated move on, if original send to a “cleanup” array
    - Within the cleanup array remove corrupted information until it fits the original formatting
    - Replace the “abcdefg” encoding with white space
    - Send to our output array
- Testing:
    - Checking to determine if a line is an original or was inserted
        - Look for a maxval greater than 255
        - Check that the ‘abcdefg’ sequence is maintained
        - Ensure that given line is 70 characters or less
        - If the file is less than 2 width or less than 3 height
- Time: 6 hours

6. Retrieve the image information from your data structures and output your restored “raw” pgm.
    - Calculate the width and height of the pgm
    - Add back the tag and maxval
    - Return the final restored pgm
    - Testing:
        - Ensure that final version stored within dynamic array fits correct criteria for pgm
        - Similar testing as done in #4
        - Run valgrind check to ensure nothing lost or leaking by end of program
    - Time: 1 hour

Testing Plan (continued)

- Print random text in between code to see if it passes certain parts of program
- Check to see that it correctly responds to pgm files and not another file type
    - Attempt to put in another file type to program and see what happens
- Constant make checks to ensure with each small section of code added our Makefile is still compiling
- Valgrind checking throughout