Harsh Sinha
Nana Adjekum

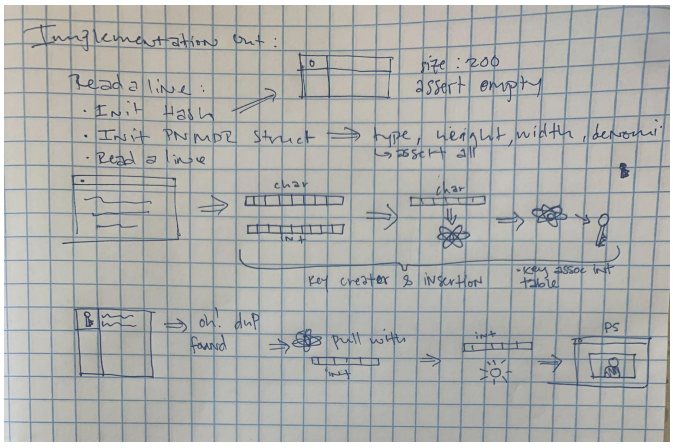
Design Plan: FilesofPix

Restoration Architecture

Readaline
Implement a readaline function that reads a line from the input file, taking into account the provided notes about a single '\n' character at the end of each row. The data is read into a char array line by line.

Restoration plan:
Separate each character in the line into infusion string or numeric sequence which, after processing will be added to the hash

1. Create a struct of Infusion with string *seq which will be an atom, and int numseq. The char *string seq array will hold all non-digit ASCII characters in a line, and the int numseq array will contain the numseq. The atom will work as the key for the hash and the values are inserted into a Hash Table. This way, all lines with the same seq (original to the image) will get recorded with their associated numseq. This will improve time complexity.
2. With a list of Infusion, we will identify the sequence that has multiple matches and use the associated numseq to identify original lines.
3. Construct the final image raster which will be a List* to the List of all matching nums list in the RawNum objects. Create a struct to represent the final PGM image (with int magic number (P2), int row, int col, int max val, List * image raster). Use the final image raster to compute the row, col, maxVal and construct the final pgm object in file for output.



Flow of code Picture:
Atom→ atom data structure
Key→ key from hash function
Sun→ brightness
Picture→ final file

Harsh Sinha
Nana Adjekum

Implementation Plan

Main – Read file with arguments and call readaline function (1 hours)

Given a file name, open the file and confirm it is valid, call readline and separate line

Initialisation: (4 hours)

readaline Function – reads raw corrupted data line by line into char array

size_t readaline(FILE fd, **datapp)

Use readaline to get contents of a line + size of that line in the file

- Create a char *line_being_read and save the pointer as char* start = line_being_read;
- append the contents of a line into line_being_read and expand the size with realloc
- loop: use getc to read char by char (make sure not eof) - read up to '\n', increment size_of_line for each char
- set *datapp to start pointer of line_being_read or to NULL if eof or size_of_line == 0

a. char *line = *datapp --> line

b. size_t size_of_line = readaline(FILE fd, **datapp) --> size_of_line

In read line also need to keep track of the og dimensions and the brightness #

- Struct to keep track of og dimensions and other info

  Struct PNDMR:
  - Create a struct to represent the final PGM image (with int magic number (P2), int row, int col, int max val, List * image raster). Use the final image raster to compute the row, col, maxVal and construct the final pgm object in file for output.

    Int
  - Test case: assert og dimensions and restored dimensions

Build out a Table with hash capabilities prior to everything

- Init the table should have a limit of rehash: 200 chars
- Run initialisation test cases here (e.g. assert is empty)

Restoration: (10 hours)

- After reading in a line, we call a function to separate line and isolate infusions and raw numbers
  - isolation(string)
    - This function gets sent a string from the file as is
    - We are going to use a while loop that will go through the line and when it reaches a number, it sends it to a type cast that converts the char to an int
      - Store casted char→int to an array

Harsh Sinha
Nana Adjekum

        - The other chars are to be kept in the char array and eventually sent to the hash function
- Storing things in the table
    - Hash function(string)
        - Need to have the key corresponding to a list of int which correspond with brightness values or int in the string
        - Creates said key and stores the ints associated with it
    - When a key is create more than once → duplicate
        - Will cease to store things in the table and return the first number stored in the map
        - From this moment on the table will only call isolation and qsort to create hash-keys from the created atoms and to find the integers
- Building out the P5 file
    - Need to calculate the brightness for each integer→ brightness funct
        - From the isolation function we get each char (num)
        - We need to then convert it to an int
        - Then calculate the brightness using the saved int in the pnmdr struct
    - Emplace the map read in the the PNMDR struct
        - Use PNMDR write to create final output file
            - Feed data from uncorrupted file
    - Change the type

Output – outputFinalPgm(ImageRaster *imageRaster) to read into another file (2 hours)
Test cases – brainstorm and test various edge cases (4 hours)
Compile - make sure all files compile and run as we progress(2 hours)

Harsh Sinha
Nana Adjekum

Test Cases Design Plan

As we implement each function we test edge cases as well as valgrind before implementing new ones.

*Use Assert
Test file for everything

1. Readaline Test Case:
    - Read more than one file
    - Read incorrect file name
    - Empty File
    - Regular File
    - File of Newline Character
    - File with 2 or less lines
    - File with more than 200 lines
    - Line with more than 500 characters
    - Line with more than 1000 characters
2. String Manipulation Test Case(s):
    - Regular Line
    - Non-alpha-numeric Character Infusions
    - Case-sensitive Character Infusions
3. HashTable Test Case(s):
    - Hash Function Correctness
        - Check if function is developing correct keys for each atom
    - No/Empty numeric sequence
    - No infusion sequence (all unique)
    - Memory Allocation (make sure memory being allocated)
        - List every time we malloc anything so we can keep track and free memory e.g. struct, table, pointers
        - Free up memory when table hashed
        - Free up memory when done with readaline and there's extra space
4. Other Test Case(s):
    - Successfully closing program after doing the restoration
    - The named input file cannot be opened
    - An error is encountered reading from an input file
    - Input is expected but not supplied
    - Memory cannot be allocated using malloc