Ian Ryan and Simon Rands
CS40 - HW1 filesofpix - Design Document
9/11/23

RESTORATION ARCHITECTURE:

• Identify what data structures you will need to compute restoration and what each data structure will contain.

- **Atoms** to hold the whitespace infusions and find the repeated whitespace infusion
- **Sequence** to hold the original lines of the pgm once they've been decoded
- **Table** with whitespace atoms as the keys and digit strings as the values
    - Each time we create a new atom and add it to the table, if it returns a digit string, we know that is an original line, and we take it and store it in the sequence. Otherwise, if it returns null, we know it's not an original line.

• Hanson's data structures are polymorphic, so you will have to specify what each void * pointer will point to.

- For the **table**, the void pointer for the key will point to a whitespace atom string and the void pointer for the value will point to a digit string
- For the **sequence**, the void pointers will point to strings for the digits

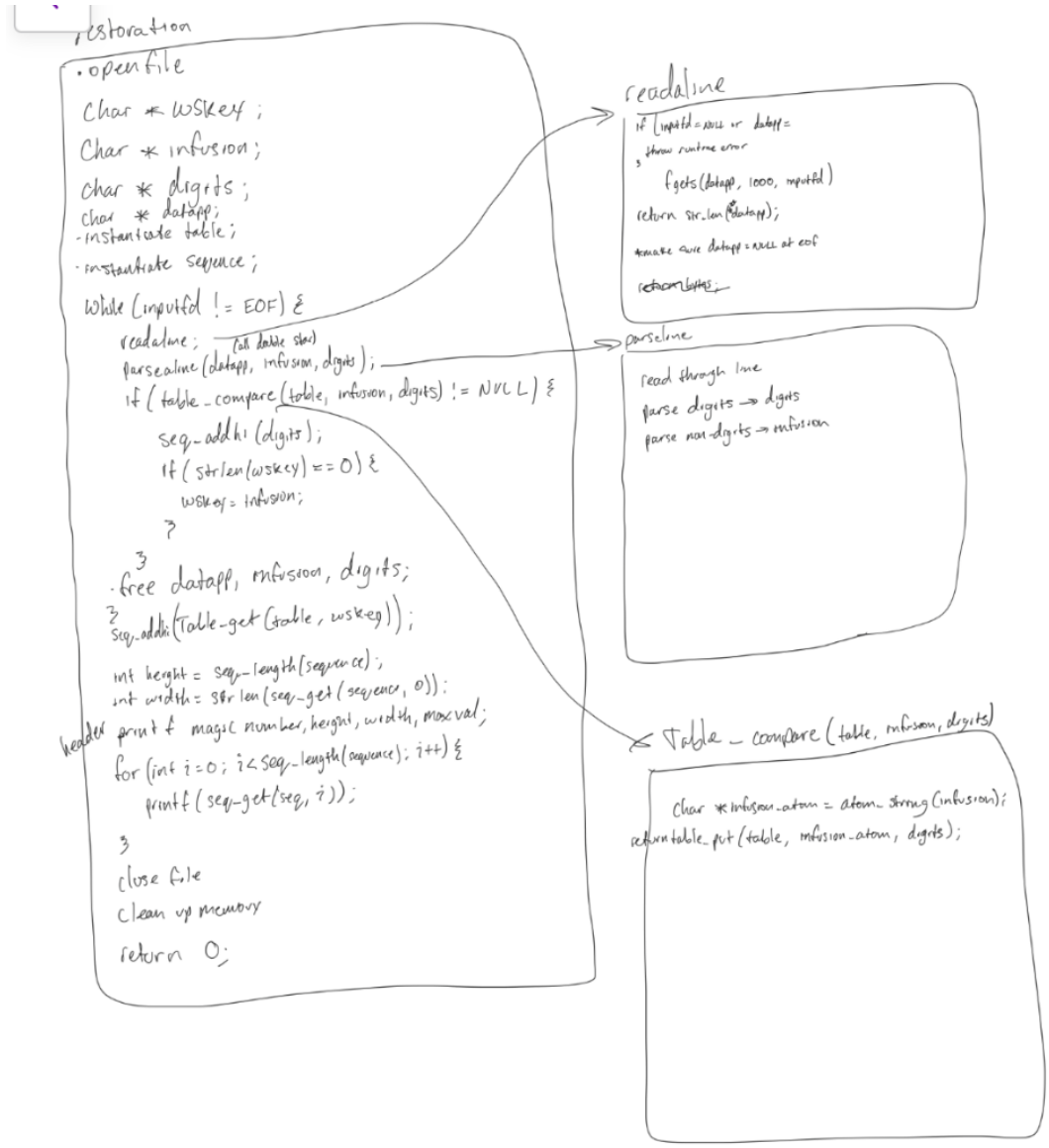
• You are supplied with a set of methods for each of the Hanson structures. Which ones will you use to get information in and out of your structures?

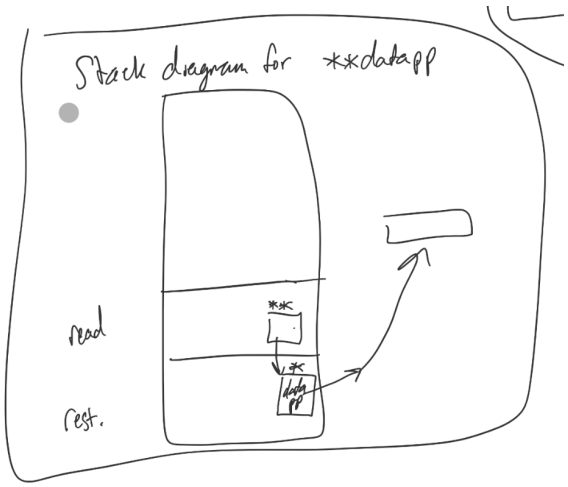
- **Atom**: We will use Hanson's atom data structure to compare all of the whitespace infusions and find the key whitespaces sequence. We plan to use $extern\ const\ char\ *\ Atom\_string(const\ char\ *\ str)$ to create an atom from the infused whitespace characters in each line, and then compare those using table_put. This will allow us to find the key whitespace sequence and also store all lines with that sequence as the values for that key in the table.
- **Table**: **table_put** will be used to place digit strings as values with infusions as keys into the table. If we use **table_put** on a key that already exists in the table, it returns the value (digit string) at that key. As the whitespace infusion is the same for each original line, this will allow us to access the previous original line each time we add a new original line to the table. We will then return this digits string to restoration to be added to the sequence. We will also use table_get to acquire the last line of the original image raster/file from the table.
- **Sequence**: We will use a sequence to hold the lines that we have verified are from the original image. Each time that **table_put** finds a repeated key, it will output the previous value and we will store that line in the back of the sequence using **seq_addhi**. Once we've added all the lines to the sequence, we will print through each line to output the original image raster.

Helper Functions:

- In addition to readaline, we plan to create the helper functions parseline and table_compare
    - Parseline will separate the digits from the whitespace infusions in each line
    - Table_compare will atomize the whitespace infusions as keys and insert the digits as corresponding values in a table

**Below is a rough drawing of the program flow with some pseudocode for different functions we plan to have.





Here is our rough stack diagram aiding our understanding of **datapp

IMPLEMENTATION PLAN

- 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**
- 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: 15 minutes**
- Extend restoration to open and close the intended file, and call readaline with real arguments (FILE *inputfd, char **datapp). **Time: 30 minutes**
- Initialize variables and data structures to be used throughout the program. **Time: 30 min**
    - Strings for whitespace key, infusion, digits, datapp
    - Table - holds atomized infusions as keys and corresponding digits as values
    - Sequence - holds original, cleaned up digits
- Build and test readaline function. **Time: 120 min**
    - Check args for null, throw runtime error if null
    - Use fgets with a stack-allocated string in readaline, size 1000, inputted
    - Malloc str_len bytes for datapp using stack-allocated string length
    - Return string length of either datapp or stack allocated string
    - **OR** use realloc to contract the size of datapp after the line’s length is known
    - Return bytes
- Extend restoration to print each line in the supplied file using readaline. **Time: 30 minutes**
- Build and test parse line function **Time: 2.5 hrs**
    - Read through datapp, writing digits to digits string and non-digits to infusion string
    - Parse using a loop until finding the null character
    - Write digits to digits string, making sure to add whitespace
    - Write non-digits to infusion string
    - Need to malloc memory for both of these strings
- Write and test table_compare function. **Time: 90 min**
    - Implement atom creation for whitespace infusions
- Add original digit lines (determined by table_compare) to sequence **Time: 20 min**
- Assemble metadata/header from the sequence length and the length of an entry in the sequence **Time: 10 minutes**
- Write the printing function that takes original image data from the sequence and prints it it to the user **Time: 30 minutes**
- Close file, clean up memory, etc. **Time: 15 min**

TESTING PLAN

- Test readaline
    - Test that the file is open when we pass it
        - Use fopen, open the file, and then set it to a variable and print it. If that variable is NULL, the file has not been opened properly; if that variable is not NULL, then it passes the test.
    - Test that *datapp is pointing to the right thing

        - Define **datapp as pointing to a string and then pass it to readaline and then print it within readaline to ensure that it still has access and is being passed to the function correctly
    -
- Test parseline
    - Test different implementations of for and while loops to find the most efficient one
    - Give parseline a known string and print out the resulting digits and infusion strings to ensure they are being assembled properly
    - Make sure there's no memory issues with digits and infusion strings
    - Edge cases:
        - Need to insert proper whitespaceage into digits string while parsing datapp
- Test table_compare
    - Test atom creation process by creating two atoms from a known string and comparing them to ensure they are equal.
    - Test table_put to ensure that we are not missing any inputs that are placed into the table and then returned upon next table_put call
    - Test table_get that it can find that last entry using the whitespace key
    - Edge cases
        -
- Restoration as a whole
    - Test outputs using pnmrdr to verify that the images are not corrupted. If they are able to read by pnmrdr, then they must be valid.
- Test edge cases
    - When readaline is open, test that there are lines to read, if not set *datapp = null and return bytes = 0;
    - For table_compare, the last original line will be left in the table, so we have to extract it after the while loop