Architecture

Data representations

- Atom
    - Inserted sequence key: const char * returned by Atom_string(pattern_cstr); canonical pointer for the non-digit sequence. Do not free.
- Table
    - Hanson Table mapping Atom as key and a Sequence of char * (which represents restored lines) as values.
    - Table_new, Table_get, Table_put, Table_map
- Sequence
    - The Sequence will contain Atoms which contains char *
    - Seq_new, Seq_addhi, Seq_length, Seq_get

Containers and what each void* points to

- Table key type: const void * → actually a canonical Atom string const char *.
- Table value type: void * → actually a Sequence
- Sequence element type: void * → actually a char*.

Memory ownership and cleanup

- Allocate each Sequence; free by iterating and freeing elements first, then the sequence.
- Table is freed after sequences are freed.
- Atoms are never freed as permitted by the spec.

Implementation and Testing Plan

Testing plan is in green!!!

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

    a. Test by running the program

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.

    a. Test by running the program

3. Extend restoration to open and close the intended file, and call readaline with real arguments.

    a. Test by including filename in an argument.

    b. Test by reading a file in standard input.

4. Build your readaline function. Extend restoration to print each line in the supplied file using readaline.

    a. Test when readaline is called where there are no more lines to be read. *datapp should be set to null and returns 0.

    b. Initialize *datapp from the parameter by using malloc to allocate 1000 bytes of memory (for now).

    c. Use fgets() to read line data and store it in *datapp. (for full credit, we would use fgetc() instead, along with malloc and realloc, to read larger lines)

        i. Test with a testing main function with our own input lines of digits with inserted non-digit sequence like "abcdefg"

            1. Convert the lines into bytes.

        ii. Print the data in datapp in int and see if it matches our original input.

5. Extend restoration. For each line:

    a. Obtain the non-digit sequence by walking through the line in datapp. Store that sequence in an Atom.

        i. Test by inserting our own sequence in lines like "abcdefg" and printing the sequence we get from this step to see if it is correct.

b. Store the line with the non-digit sequence removed back in datapp.

    i. Test by printing the restored line and see if it matches our intended input without the “abcdefg”.

c. Insert the Atom in Step a. as a key into a Table. If it’s empty, create a Sequence with pointer to datapp as its first element and store the Sequence pointer as the value to the table. If it’s not empty, append to the existing Sequence there and save the Sequence pointer in a temporary pointer. This is the sequence that would store the original rows before it was hacked.

    i. Test with input files with only lines with the same non-digit sequence

        1. There should only be one key-value pair in the Table. Print to see if correct.

    ii. Then test with files with each line having a unique sequence.

        1. There should be as many key-value pairs as lines. Walk through the Table and print each one

    iii. Test with a mixture

6. After reading through each line, write the header and each row in the Sequence (in bytes) from Step 5c to standard output.

    a. Test by writing each row in ints instead of bytes and checking with our original file.

7. Free the memory in the Table.

    a. Run with valgrind