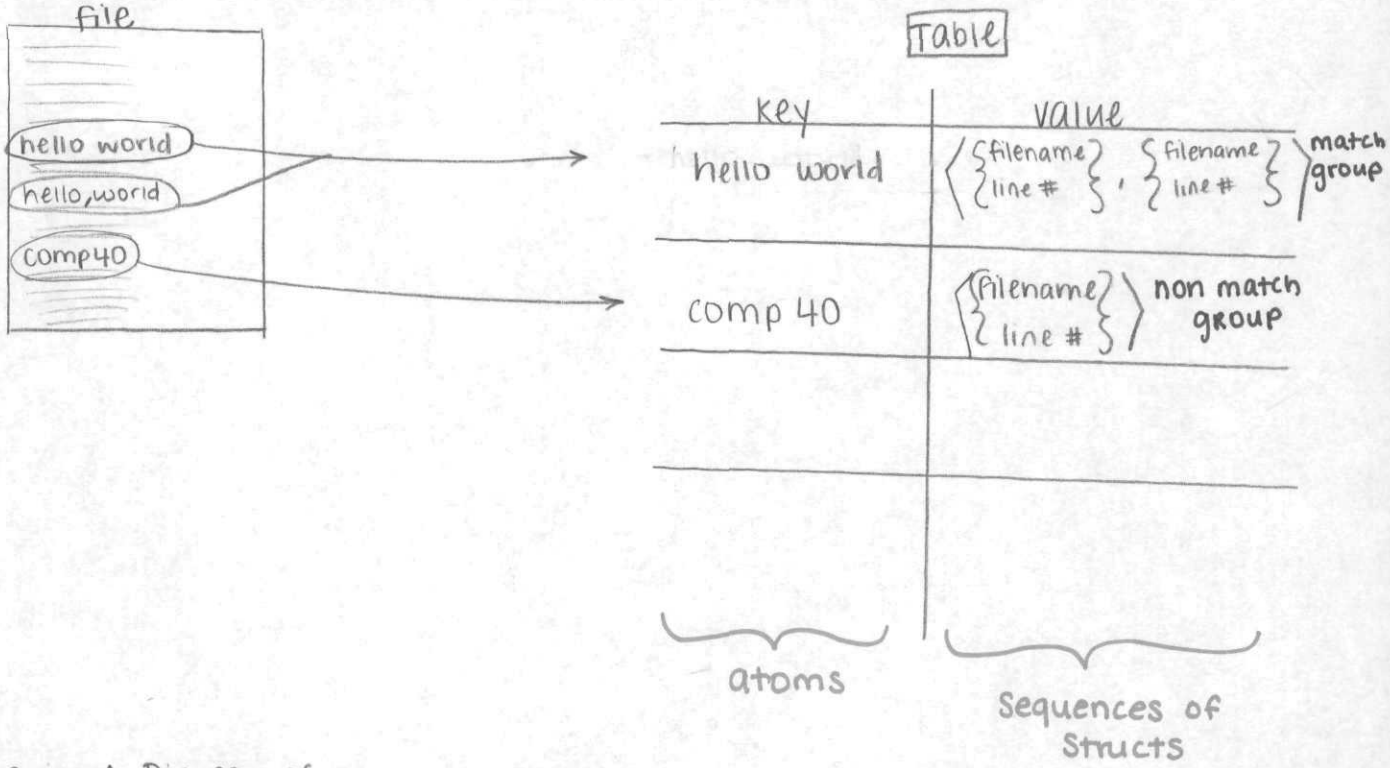
Katie Hoskins and Amoses Holton
Comp 40
Assignment 1

Assignment 1 Design Document

Data Structures Involved:

The primary data structure that we will be using is a Hanson's table of key, value pairs. The keys in the table will be atoms that store line contents (ex. "hello world") and the values will be sequences. The elements of the sequence will be structs containing of a character array containing a filename and an integer to store a line number. As lines are read in, any extra spaces and non-characters will be removed to ensure uniformity of lines. For example, if the lines "hello,world" and "hello world" were both read in, they would both be stored in the table as the key "hello world". Each time a new line is read, there are two possibilities. If the line already exists as a key in the table, another element is added to the value sequence (the filename and line number the newest line was found in). If the key does not exist in the table yet, a new key, value pair is created and added to the table.



DRAWING A. Diagram of data structures involved.

This table will simultaneously store all the lines that were read in from input files as well as match groups (any key, value pair in which the value sequence has more than one member, indicating more than one location in the input files).

Void Pointers

Void * pointers will be used for three aspects of our table data structure. The first will be to point to the key, which in our case is an atom containing the characters from a line in the input document. The second will be to point to the values, which for us are sequences containing location information for the lines. Void * pointers will also be used to point to each element of the value sequences, which for us are structs of a character array and an integer.

Invariant for Simlines

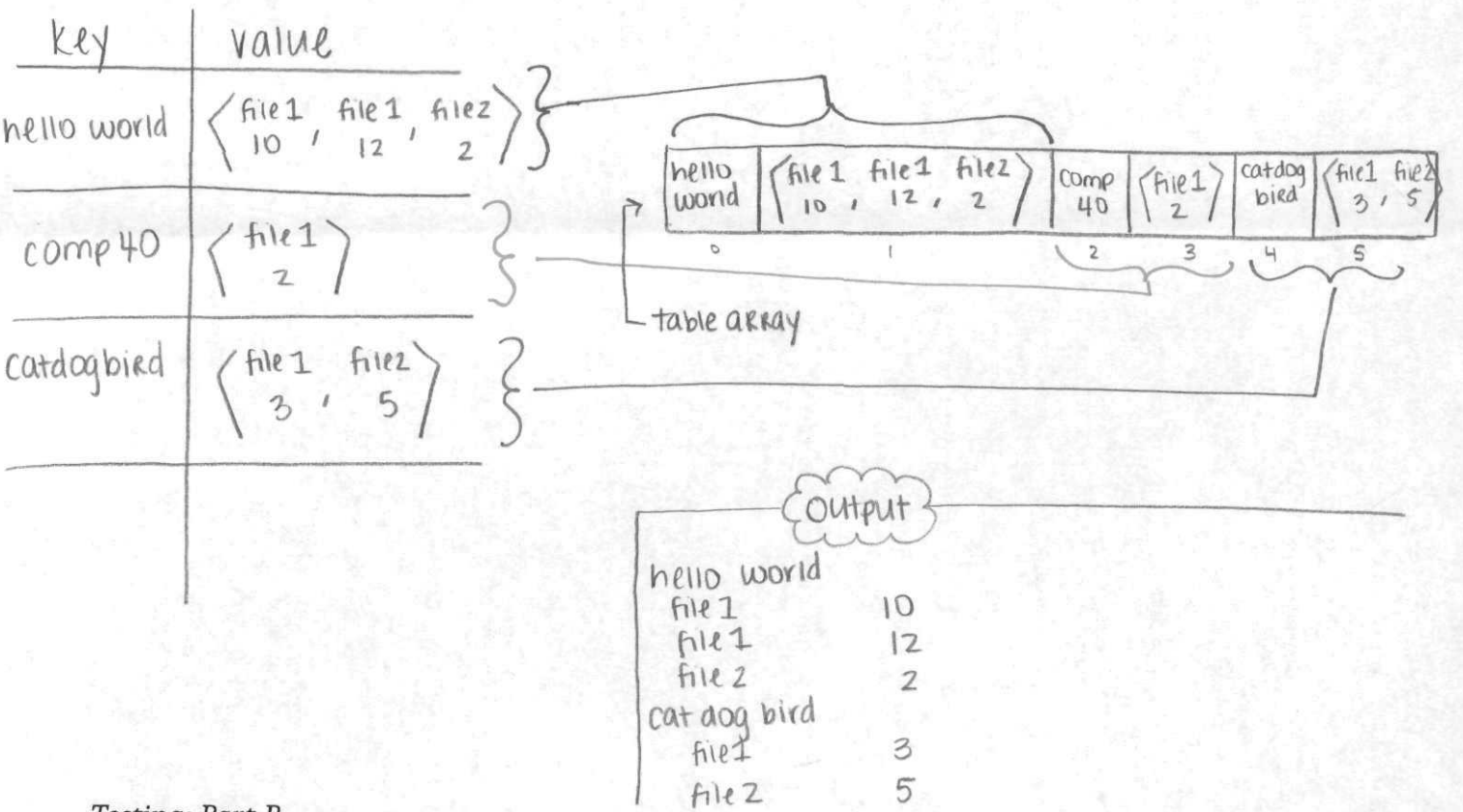
When the program is currently reading the nth line from the input files, there will be n – 1 – x total elements in all of the sequences stored in the table, where x is the number of empty lines that were in the file (these are not stored in the table).

How Match Groups will be Determined – how output sections will be written

As lines are read and stored as keys in the table, match groups will automatically be created since inserting a key that already exists in a Hanson's table will just update the results for that value, instead of just adding another key, value pair. Therefore, if a new line is read in (and transformed to the general *word [space] word* format) and already exists in the table, the match group is automatically formed when the filename and line number struct for that new line is added to the existing sequence for the value.

refer to drawing A for match group examples.

To print the match groups to the screen, first we will put the data from the table into an array using the Table_toArray( ) function. This will create an array of our key, value pairs that can easily be cycled through. For each key, value pair in the array, we will check if the length of the value sequence is greater than one (meaning there were multiple matches found). If this is the case, the key, value pair represents a match group, and the atom containing the actual words in the line is printed to the screen followed by the data in the sequence (the filenames and line numbers where the matches were found). The order of the files will be in the same order in which they were read since they were added to the table as reading occurred. If the sequence only contains one element, this means that no matches to the original line were found, and thus the key, value pair does not make a match group. These pairs are ignored.



Testing: Part B

We plan to do our testing by constructing test text files that will be fed as arguments to the readaline function. The following files will be created:

- An empty text file – function should return 0 and datap* is set to NULL
- A text file with one line of 15 characters – returns 15, sets datap* to point to the first byte in a byte array with 15 elements and sets the file seek pointer at EOF

- A text file with two lines of 15 characters – returns 15, sets datap* to point to the first byte in a 15 element byte array and sets the file seek pointer to point to the first unread character in the second line
- A text file with one line of 2000 characters – returns 2000, sets datap* to point to the first element in a 2000 element byte array and sets the file seek pointer at EOF

We will also perform the following tests:

- Passing the function a NULL argument – should terminate with a checked runtime error

Testing: Part C

To test our simlines program, we will create the following test files and use them as arguments:

- 1 file with a single, empty line – check that the table is empty and nothing was stored
- 1 file with one line of chars - check that there is only one key, value pair in the array and the sequence value has one element with the proper filename and line number
- 2 empty files – table should be empty indicating that nothing was stored
- 1 file with 2 valid but non-similar lines and one empty line – table should have 2 key, value pairs but no match groups
- 1 file with all the same lines – table should have one key, value pair that creates a match group, all lines in the file should be included in the match group's sequence
- 2 files with multiple identical lines – table should have one key, value pair that represents one match group with all lines included
- 2 files with 2 sets of identical lines – table should have 2 key value pairs that each represent a match group
- 2 files with the same order of words, but different word splits (anything that is not a-z or 0-9) – table should only have one key value pair representing a match group
- One file that is not a text file that can be opened – should raise an error
- 3 files with some matches and some not – make sure that match group data is printed in the order of input files
- 1 file with a filename greater than 20 – make sure the format upon printing the name is still in the correct format

The following tests will also be performed:

- Valgrind to ensure there are no memory leaks
- No input to the function – should raise an error