Favour Okereke-Mba
Timi Dayo-Kayode
01/28/2017
Spring 2018
COMP 40

HW #1 DESIGN DOCUMENT

PART B: READ A LINE

Test Cases

- Cases To Handle and Test:
    - Using a file pointer that points to an unopened file - the program terminates with a Checked Runtime Error
    - Empty/Blank file - *datapp is set to null and the 'readaline' function returns 0.
    - Either or both file pointer and &datapp is null - the program terminates with a Checked Runtime Error.
    - Memory Leaks - use valgrind to find and rectify these.
    - If there is sequence of bytes resembling the null terminator before the actual end of the file - The program terminates, and the file will not be read in its entirety.
    - Non-text file such as image files - expected output; gibberish

- Edge Cases To Look Out For:
    - Multiple new line characters and then an actual line: the *datapp should remain NULL until it gets to the actual line.

- Testing Process:
    - Since the readaline function is a stand alone function that plays a big role in our simlines program, we decided to create a main driver in which to carry out our testing. Within this driver, we made multiple calls to the readaline function with varying input.

PART C: SIMLINES

- Problem Statement: The simlines program identifies and displays similarities within or among one or multiple files. It finds lines that have the same words in the same order in different lines within the same file, or among multiple files.

- Use Cases: The simlines program could be used by both a human client and as a helper to another more sophisticated program. It requires one or more files containing words/sentences. An example usage of the program is as follows:

  simlines testfile otherfile secondtestfile

  In the example above, testfile, otherfile, and secondtestfile are files containing words/sentences and when passed as arguments to the simlines program as shown above, the program checks the files for identical lines both within themselves and in comparison to sentences within the other files passed as arguments. An example output of such an input would be;

  hello world
  testfile----------2
  otherfile----------7
  secondtestfile---------2

  This output shows that the compund word, “hello world” occurs in the files; testfile, otherfile, and secondtestfile in lines 2, 7, and 2 accordingly.

- Assumptions and Constraints:
  - The client passes in one or more arguments to the simlines program.
  - The arguments are valid, readable files gotten from the command line.
  - Once the program successfully executes, it prints out the compound words that occur multiple times as well as the locations -file name and line number- of these occurrences.
  -

- Architectural Design: We plan to use Hanson’s Abstract data structures; Atoms, Tables, and Lists. These data structures will interact with one another as our way of managing and storing the information received from the client. Lines read from files are stored in an atom, and the atoms in turn are hashed into the table; the keys are the atoms, and the

values are lists which contain the file name the string was read from, and the line number it was in.

- Pseudocode:

  While there are files to be read; argv != null{

  - Read in the first file from the provided command line arguments.
  - Open the file and check that the file is opened successfully
  - Loop through all the lines in the file
  - Use the readaline function to get the lines in the file
  - Parse the line to get rid of non-word and separative characters
  - Make an atom out of the parsed line
  - Hash the atom to the table with the atom as the key and a list containing its origin file name and line number as the value.

  ● Close the file

  ● }

  - Now that the parsed lines are stored in the table along with their file origins and line numbers, loop through the table, printing slots that have a list length greater than one in the specified format.
  - Free all allocated memory

- Invariant: The invariant that MUST hold true while the simlines program is part way through reading input lines is that **we have one or more files whose lines haven't been read by our program**. This is true because once the simlines program reads the final line of the final file, the lines are compared, which in our implementation happens close to constant time and the simlines program completes its 'duty'. At this point, the program is no longer *part way through reading files.* Therefore, as long as this invariant holds true, the simlines program is still 'part way' through reading lines.

- Void* Pointers:
  - At the first creation of our table, since Hanson's implementation provides comparison and hashing functions for Atoms, the table will be created as such: Table_T line_groups = Table_new(0, NULL, NULL). The two NULLs show that we are not providing any comparison nor hashing functions for the table because we are hashing our data as atoms and Hanson provides functions to do this.

  - To insert the keys and their respective values into the table, we will be using the function; void *Table_put (T table, const void* key, void *value).

The key will be our atom and the value will be a pointer to a linked list of occurences of the string. We will also be using the function; void *Table_get (T table, const void *key). The key will be the atom containing the string to be searched for.

- In extending the values of a linked list, the Hanson provided function List_push will be used as such; List_push (T list, void *x). The void* pointer x will point to the most recent addition of the linked list which will be a struct containing the file name and line number that the string occurs.

- Implementation Plan:
  - The algorithms and data structures that this program will use as stated above are atoms, tables, and lists. Our program will act as a client to Hanson's library's interface, using his abstract data structures outlined above. Our goal is to have our implementation of all the components of the program ready for testing and debugging by Tuesday, Jan 25th, and have a perfectly working program as specified by the assignments ready by the deadline of February 4th.

  - Program Output:
    - The groups of the output are determined by the length of each linked list's value of an atom in the table. Since the atom represents a line, and any list length with a value greater than 1 represents a line that has occurred multiple times, the Table_map function will be used to get atoms that have lists of length greater than 1. Whatever this function gets and returns will be a lines in one or more files with multiple occurences.

    - Once the Table_map function has identified a linked list with a length > 1, we print out the atom containing the line that represents that group, and each element of the linked list that represents the location -file name and line number- of occurrence.

- Memory Management: The ADS's provide by Hanson free the memory allocated for the data structures themselves, but not the memory allocated for the things these structures contain, and so we plan to write a function; free_all() that will free up all the memory used by the simlines program.

- Testing Plan: We will employ the programmer style testing plan of unit testing by having test files to test each function and section of our program as we write it to

avoid accumulated run time and compile time errors, and to prevent instances of cascading errors.

- Test Cases:
  - There are a number of file type and formatting errors that this program should test for. However, these are handled by Part B; The readaline program. Refer to the Part B section of this document to see what these test cases are.
  - We will also test the program against the following types of formatted input:
    - “Abc * bca????wca” and “Abc bca wca”: Should be considered similar by our program.
    - “Abcd cd abc” and “cd abc Abcd”: Should not be considered similar by our program.
    - An empty file: Should not return anything at all.
    - If the same file is passed as an argument twice: Our program should compare the contents of these files just like it would non-similar files.
    - Providing one or more valid files plus an invalid file: The program won’t give any output because the readaline function treats invalid files as checked runtime errors which will lead to termination of the program.
    -