Name: Daniel Ems
Date: 01/27/2017
Current Module: Data Structures and Algorithms
Project Name: Intersect

Project Goals:

The projects objective is to take in an unknown number of files, and find the words that intersect between all files passed. The objective is to familiarize students with different data structures, architectures, algorithms. Finally, to demonstrate the importance of addressing the time complexities associated with the each.

Considerations:

- Reading the first file into the program as well as subsequent files.
- Deciding which data structure to store, organize and compare strings.
- Determining what properties to incorporate into the data structure to identify strings that intersect.
- Identifying the most efficient way to search and compare strings.
  Labeling strings that intersect
- Printing the results alphabetically.

Initial Design:

The initial approach was to find analogies within the problem, and then incorporate code form other assignments with similar requirements. Three previous assignments that lent similarity were the BSTnodes, Codec and Word Sorter projects. The snippets of code that were initially adopted were reading in an entire file into a buffer from Codec, and sorting words alphabetically from Word Sorter. This approach offered a quick start to the assignment.

After implementation of the initial approach, certain problems became impassable. The first issue was reading in the entire file into a buffer. It was immediately apparent the time complexity and memory requirement was unnecessary. Further more, the utilization of strtok made the code confusing and difficult to read. This called for re-approaching the problem, keeping these issues in mind.

The initial approach also offered solutions to two requirements; storing the words, and printing alphabetically. Utilizing a binary search tree allowed for building in a count property into the node which is used to identify words that intersect. Additionally, utilizing a binary search tree afforded the opportunity to order the tree. As words were passed to the tree, the were strcasemp'd with words already in the tree to determine whether the word to be inserted should be created to the right or left of the current node. This handled the issue of printing alphabetically.

To address the issue of reading in files, a buffer was created, and calloced to the maximum word size the program would read, 256. fgetc was placed in a while loop and each character was checked whether or not it was white space. If the letter was white space, a NULL byte was inserted into the buffer at that location to terminate the string. The string passed to the Insert function for the first file. The string was passed to word_check for subsequent files read.

To identify intersected words, a counter was built into the nodes of the tree. The counter was incremented only if two parameters were met. First, the word being searched, must exist in the tree. Second, the count of the matched node must not exceed the numerical order of the file being read (file 1 = 1, file 2 =2 , file 3 =3, etc.). The count was then used to print which words were intersects after all files were read.

Data Flow:

1. Command line arguments are checked and processed.
2. A buffer is calloced and the first file is read in one word at a time.
3. each word is passed to the Insert function, if the word does not exist, a new node is created.
4. After the first file is read, the tree has been built.
5. The remaining files are read in to the buffer, one word at at time, one file at a time.
6. Each word in each file is passed to the word_check function, which checks for a match in the tree.
7. If a match is found, the counter is incremented as long as both parameters are met.
8. This process is repeated for every file following the first file, passed to the program.
9. Once all files have been read, all nodes that have a count = to argc -1, are printed.
10. The appropriate FILE's, malloc's, trees, are respectively closed, freed or destroyed

Potential Pitfalls:

The most notable pitfall is that of time complexity. This involves numerous elements of one's program; architecture, algorithms, data flow, and organization. Each data structure implemented has both pro's and cons. Furthermore, in order for any data structure to be efficient, it must be implemented appropriately. This becomes particularly important when addressing the requirements of the exercise; accepting, organizing, and search multiple large files. These elements must be considered in determining which tool to employ. Despite the data structure selected, reducing the time complexity of the program is synonymous with maximizing the potential of the approach taken.

Test Plan:

User Test:
1. Test reading in files.
2. Ensure the words are being created and stored appropriately.
3. Identify that all files are being read in appr0priately.
4. Constantly check to valgrind.
5. Test your trees accuracy in storing your strings.
6. Test the accuracy of matching strings.
7. Test the accuracy of identifying intersecting words.

Test Cases:
1. Place lyrics to Hotel California file 1.
2. Read, store, search and print until done so accurately.
3. Divide the lyrics in half, and store them into file 2.
4. Repeat step 2.
5. Divide into quarters and eighths, store in files 3 & 4.
6. Repeat step 2.
7. Place A Tale of Two Cities and Moby Dick into two files.

8. Repeat step 2.

Conclusion:

The most effective aspect of the project was the planning. Independent reading provided the following approach to addressing programming problems; 1. Plan, 2. Restate the problem, 3. Divide the problem, 4. What can I Do, 5. Reduce the problem, 6. Find analogies, 7. Experiment, 8. Do not get frustrated. Following these guidelines before I started typing afforded many advantages to prior assignments. Each component to the strategy lent a hand in paving a path ensuring efficiency and effectiveness in solving this problem.

The most detrimental aspects encountered were utilizing tools not well practiced, primarily binary search trees. This often led to issues that required a great deal of time to resolve. Furthermore, the inexperience may have led to an under utilization of the structure. This unfortunately could result in undermining the architecture, and efficiency of the program.

Future projects will be more effective if the tools implemented are better understood prior to the assignment. While the training tool of learning under pressure is well understood. Being properly prepared is half the battle. Further reading, practice and study would serve to make implementation much more effective, arguably maximizing on learning objectives.