Name: SGT Schoenwald-Oberbeck, Jesse
Date: 27JAN2016
Current Module: Data Structures and Algorithms
Project Name: Intersect

Project Goals:
Create a file that takes in n files, finds the words they have in common(the intersect), and prints them out in alphabetical order. The program should not crash when large files are used, and should work as quickly as manageable.

Considerations:
o Files can not be read in all at once, as file size can exceed the memory available.
o Operation must be quick, so efficiency must be a factor in handling the data, and searches thereof.
o Words may only appear once each, regardless of case.

Initial Design:
Main exists within intersect.c, all other functions are in “intersectlib.c,” which is included via the header file “intersectlib.h.” Made file will be “intersect.”

Files will be read in one at a time. Each file will be read a single word at a time, delimited by any manner of whitespace, to include: space, tab, and newline. This will be accomplished by using fscanf. The words will be built into a binary search tree. The tree will be reconstructed for each file, only adding words in common.

Data Flow:
Main will enter a for loop, based on the number of args. The loop will call a function which returns a file handle, and the file handle will passed off to a central file handler function. This function will, upon receiving the first file, build a binary search tree by calling the insert function on each word. The insert function will exclude repeat words based on a case insensitive compare, thereby only retaining the case of the first appearance of the word.
All files beyond the first will be handled by the second portion of the function, which is largely the same as the first, except that it will build a new tree. It will do so by first calling a check/find function, which will return one value if the word is already present, and another if it is not. If the word is present it will be added to the new tree. Whether or not it is present, it will be deleted from the original tree. What is left is a tree of only words that were in both files.
Each subsequent file will either keep the same list or, much more likely, narrow it further.
Finally, the tree will be printed by means of a standard in order print.

Communication Protocol:
No networking was required, and none were implemented or used.

Potential Pitfalls:

o Dealing with large amounts of data at once requires a careful approach. A file cannot be loaded into memory if it exceeds the capacity of memory without significant degradation of the ability of the program. Either crashes will result, or paging will be used. Paging causes significant slowdown.

Test Plan:

User Test:

Small lists of words made by the tester.
Two lengthy novels.
Two lengthy novels and an English dictionary.
An English dictionary and the inverse of that dictionary.

Test Cases:

All test cases completed with correct output.

Small lists:
```
   real  0m0.001s
   user  0m0.000s
   sys   0m0.000s
```
Two novels:
```
   real  0m0.597s
   user  0m0.448s
   sys   0m0.016s
```
Reversed Dictionary:
```
   real  1m7.772s
   user  1m5.488s
   sys   0m0.176s
```

Conclusion:

My approach to this project was planned out before execution in a fair amount of detail. And although some aspects had to change during development, I would say that the time spent on this phase was worth it. I gave some consideration to making a self balancing tree, but given the time constraints for the project, and the fact that the structures are very short lived, I arrived at the conclusion that implementing balancing functions was impractical, and possibly detrimental to performance.