Systems Programming
Project #3 - Indexer

David Lambropoulos
Demetrios Lambropoulos

Professor Brian Russell
March 12, 2015

Rutgers University
Department of Computer Science
School of Arts and Sciences

1 How To Use

Enter the following:

./index < FILE TO SAVE TO > < FILE/DIRECTORY TO READ FROM >

Given that you are to run this program, there are a few events that can occur:

No Arguments Entered

```
dal220@Chronos:~/Desktop/pa3$ ./index
WARNING: No arguments! Correct execution count not be ensured!
Try again:
Sample: index <inverted-index file name> <directory or file name>
Execution Time: 0.000058s
```

Too Few Arguments Entered

```
dal220@Chronos:~/Desktop/pa3$ ./index ery
WARNING: Too few arguments! Correct execution could not be ensured!
Try again:
Sample: index <inverted-index file name> <directory or file name>
Execution Time: 0.000056s
```

Too Many Arguments Entered

```
dal220@Chronos:~/Desktop/pa3$ ./index "sample1.txt" adfs adsf
WARNING: Too many arguments! Correct execution could not be ensured!
Try again:
Sample: index <inverted-index file name> <directory or file name>
Execution Time: 0.000064s
```

Correct Number of Inputs, Reading File is Binary

```
dal220@Chronos:~/Desktop/pa3$ ./index "sample1.txt" "index"

index is a binary file!
Execution Time: 0.000188s
```

Correct Number of Inputs, Reading File/Directory Doesn't Exist

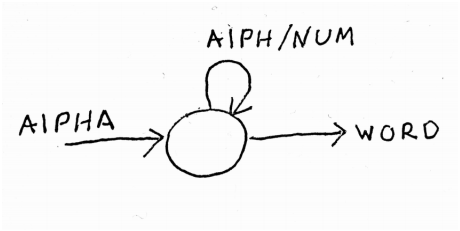
```
dal220@Chronos:~/Desktop/dsad$ ./index "sample4.txt" "hello.txt"

Could not open file/directory hello.txt to store into sample4.txtExecution Time: 0.000030s
```

Correct Number of Inputs, Saving File Already Exists

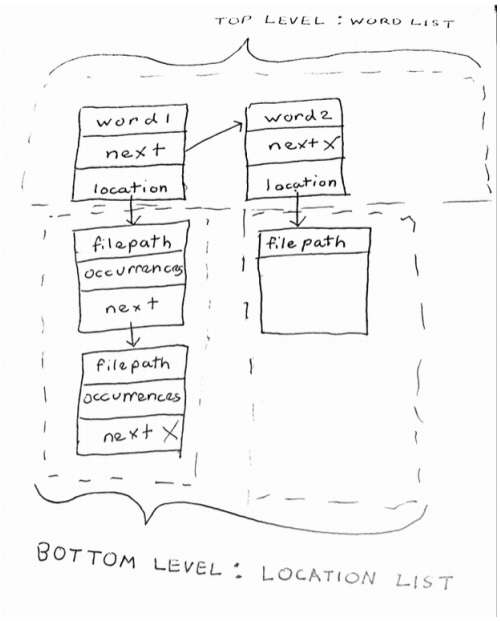
```
dal220@Chronos:~/Desktop/dsad$ ./index "sample2.txt" "sample.txt"
The file you have specified to save to already exits!
Execution Time: 0.006245s
```

2 Modularity/Functionality

Anything that has to do with files is handled by the fileManager.c file and returned by to the main where this information is then tokenized appropriately and stored in arrays. The following illustration demonstrates the finite state machine used for reading words. Such that words are defined as no more than 200 characters and begin with an alphabetic character and followed by alpha/numeric characters.



The sorted linked list of words in which each word contains a sorted linked list of locations and occurrences ressembles the following:



3 Scalability

Our program has some bugs that exist within, it will accept and store a single large file but when files are in a directory, the program can read them as noted in the testcases.txt. However, for very large files occurring in the directory there are some errors such that it still sorts the words but begins to stop storing file locations. However, this is only for large files within a directory.

4 Algorithms/Implementation/Run Time Analysis

fileManager

int isRegularFile(const char* path); $\boxed{O(1)}$

int isDirectory(const char* path); $\boxed{O(1)}$

int isCharacterDevice(const char* path); $\boxed{O(1)}$

int isBlockDevice(const char* path); $\boxed{O(1)}$

int isPipe(const char* path); $\boxed{O(1)}$

int isSymbolicLink(const char* path); $\boxed{O(1)}$

char* fileToString(const char* path); $\boxed{O(n)}$

int numBinaryFileInDirectory(const char* path); $\boxed{O(n^2)}$

int numFilesInDirectory(const char* path); $\boxed{O(n)}$

int isBinary(const char* path); $\boxed{O(n)}$

char** directoryToString(const char* path); $\boxed{O(n^2)}$

void printFilesInDirectory(const char* path); $\boxed{O(n)}$

char** filesInDirectory(const char* path); $\boxed{O(n^2)}$

long int bufferSize(FILE* fp); $\boxed{O(n)}$

int Exists(const char* path); $\boxed{O(1)}$

word-list

WordListPtr WLCreate(); $O(1)$

void destroyWordList(WNode* head); $\boxed{O(n)}$

void WLDestroy(WordListPtr wl); $\boxed{O(n)}$

int WLInsert(WordListPtr wl, char* word, char* location); $\boxed{O(n)}$

WordListIteratorPtr WLCreateIterator(WordListPtr wl); $\boxed{O(1)}$

void WLDestroyIterator(WordListIteratorPtr wi); $\boxed{O(1)}$

char* WLNextItem(WordListIteratorPtr wi); $\boxed{O(1)}$

void PrintJSON(WordListPtr list, char *file); $\boxed{O(n)}$

location-list

LocationListPtr LLCreate(); $O(1)$

void destroyLocationList(LNode* head); $\boxed{O(n)}$

void LLDestroy(LocationListPtr ll); $\boxed{O(n)}$

int LLInsert(LocationListPtr ll, char* location); $\boxed{O(n)}$

LocationListIteratorPtr LLCreateIterator(LocationListPtr ll); $\boxed{O(1)}$

void LLDestroyIterator(LocationListIteratorPtr li); $\boxed{O(1)}$

char* LLNextItem(LocationListIteratorPtr li); $\boxed{O(1)}$

tokenizer

TokenizerT *TKCreate(char* ts); $\boxed{O(n)}$

void TKDestroy(TokenizerT* tk); $\boxed{O(n)}$

char *TKGetNextToken(TokenizerT* tk); $\boxed{O(n)}$

alerts

void warning(char * msg); $\boxed{O(1)}$

void error(char * msg); $\boxed{O(1)}$

void success(char * msg); $\boxed{O(1)}$

void genwarn(); $\boxed{O(1)}$

void generr(); $\boxed{O(1)}$

void gensucc(); $\boxed{O(1)}$

5 Extra Credit

5.1 Binary handling

Will note if a binary file is being read and will not allow and prompt the user with an error message as seen above.