Darrell Percey

Data Structure II

Project 5

Functional Decomposition

**Data Structures**

struct LCS\_Info {

int cArray[MAXSTRING][MAXSTRING];

int mLength;

int nLength;

char x[MAXSTRING];

char y[MAXSTRING];

} LCS\_Info;

struct sequences{

int mArray[2][MAXLENGTH];

int mLength;

int nLength;

int stringAmount;

char listOfStrings[MAXLENGTH][MAXLENGTH];

} sequences;

The LCS structure allows me to hold two strings and a matrix for the c-Array to find the LCS of the two strings. The sequences structure holds a list of strings up to 100 and compares them.

**Files and Functions**

**Test.c**

This is the driver file that calls functions from TableCompare.c and LCS.c to run the program.

**TableCompare.c/h**

/\*

\* Function: compareMultipleSequences()

\*

\* Description:

\* compares the multiple strings by sending

\* them to a 2\*M table for computation

\*

\*/

Is the driver for the multiple comparing sequence file.

/\*

\* Function: createNewSeqNode()

\*

\* Description:

\* Allocates memory for a new sequence structure

\* and sets all values to 0 or NULL

\*

\*/

Makes a new node to hold all the information for the comparing.

/\*

\* Function: readFileMultipleSequence()

\*

\* Description:

\* Reads the multiple strings into a

\* 2-D array

\*

\*/

Reads in all the information from the file that is stdin

/\*

\* Function: findLCSLength()

\*

\* Description:

\* computes the LCS by comparing the letters

\* in the strings.

\*

\*/

Computes the length of the LCS to pass along to test the similarity between strings.

/\*

\* Function: findSimilarity()

\*

\* Description:

\* Computes the percentage of lengths and

\* the percentage of the LCS to the shortest

\* string.

\*

\*/

Compares similarity based on percentage of length and LCS length.

/\*

\* Function: wipeArray()

\*

\* Description:

\* wipes the mArray clean just incase.

\*

\*/

Cleans the array each time it is used after finding the LCS length.

**LCS.c/h**

/\*

\* Function: findLCS()

\*

\* Description:

\* Calls functions to compare two strings and

\* find their LCS to print to the screen.

\*

\*/

This is the driver for the LCS finding of the two sequences text file.

/\*

\* Function: createNewLCSNode()

\*

\* Description:

\* creates and allocates memory for a new

\* LCS structure

\*

\*/

Makes a new node to hold all the information for the LCS of the two strings.

/\*

\* Function: readFileSequences()

\*

\* Description:

\* Reads in the two strings from the file.

\* "twoSequences.txt"

\*

\*/

Reads the two strings from the a file.

/\*

\* Function: LCSTable()

\*

\* Description:

\* Makes a c-array table with a 2-D matrix

\* using the two strings.

\*

\*/

Creates the full c-array of the two strings and passes it to printSequence

/\*

\* Function: printSequence()

\*

\* Description:

\* recursivly finds the LCS of the two strings

\* and prints it to the screen one character

\* at a time.

\*

\*/

Prints the LCS of the two strings to the screen in stdout.

**Conclusion**

I tried some algorithms on the 2nd part of the program before consulting with Dr.Coffey. Afterwards I fixed all the items that we discussed and it seems to run more efficiently and I also cut a lot of coding off the top of the program.