OPEN/READ FILE

Open File

IF file not found

“File not found”

Else

WHILE not at end of file

Read line

Check for 2 parameters

IF not found

Error

Else

Read Line

If number of parameters => 3

Check that each parameter after 2 = 1st parameter of a line in file

If Parameter does not exist

Error

Else

Read next line

Create Course Objects

Initialize Course variables

Open File

While not at end of file

pushback course to file

Search File

Get Input

Open File

While not at end of file

If input = courseNumber

Print course Information

If course includes prerequisites

Get number of prerequisites

For (i=0; i < numPrereqs; i++)

Print Prerequisite[i] information

Else

Error

MENU

Display Menu options

If input = 1

Load Data Structure

If input = 2

Print Course List

If input = 3

Print Course Info

If input = 4

EXIT

Load Data Structure

Read File by line

If parameters < 2

Not Valid

Else add to hash list in alphanumeric order\*\*

Print Course List

For (i=0; i< Course List Size; i++)

Print Course Info

Print Course

SearchSize = Course List Size / 2

While course != search course

If course > course at SearchSize

For (i= SearchSize; i < Course List Size; i++)

SearchSize = SearchSize / 2;

If course < course at SearchSize

For (i= SearchSize; i < Course List Size; i--)

SearchSize = SearchSize / 2;

If course = search course

Print Course Info

The program will parse each line in the file by separating the variables by the comma in the line “,”. If there are more than 2 items, then it will have a prerequisite added if it matches another course. It will stop read each line separately then end when there are no more lines to read. If there are less the 2 variables, then it is skipped as it doesn’t meet the required parameters.

To create an object for each course on each line I will of course create an object in the code and set the object course name, number, and prerequisites. If there are no prerequisites, then it will remain null.

My understanding of Big O notation is lacking at best, but from my understanding in the pseudocode there are 7 lines that would be read repeatedly so that would make it O(n).

There are many advantages and disadvantages of each data structure type. Out of them all, I find trees, or binary trees, the most interesting. It seems one of the best ways to store a large amount of data that would have to be searched through or organized. The big drawback to it is that it is more complicated to delete or remove a node in the tree thin in other data structures. Linked lists have the advantage of that is much simpler to add or delete nodes in the list, but it can be more time consuming to move items in the list as it is sequential, so you need to go through the list starting at the front to find an item in the list. Lastly hash tables are one that I had a bit of trouble wrapping my head around. From my understanding hash tables can be extremely fast to transverse compared to other data structures especially when you start having large amounts of data to be included. The drawback of them, though, is that they can be a bit complicated in understanding.

I think, especially with my current understanding of these data structures, that a binary tree would be the best to use. Having MATH201 as the root node and having the rest sorted by the number in the course number. Making it have 4 courses on the right and 3 on the left, making transversal much faster when searching for a node. The big O notation for this would also be O(n) from my understanding but would really be determined by the height of the tree itself.