Shawn Pierce

2/13/2023

CS-300

Pseudocode

1.

State the file name

State that were using files

String Filename:

Select eBid\_Monthly\_Sales

Open this file

Store Lines

Read each line

Read each line until the end of file

Get each line

Sort each line

Close file

State the file name

State that were using files

String Filename:

Select eBid\_Monthly\_Sales\_Dec\_2016

Open this file

Store Lines

Read each line

Read each line until the end of file

Get each line

Sort each line

Close file

2.

Declare each file that will need to be opened

Set parameters for the parsing

Open each file individually

Read the data in each file line by line

Parse each line

Check for any errors

Sort the data to match the different courses

Store each course into these categories

Loop with each file until every one is categorized and stored

Declare a search function

Loop the search function to continue searching until the request is found

Once found print the course info and prerequisites used in the data categorized form before

3.

Enter file name: Input ”File name”

If file is found open file

Else print ”Error try again”

Read each line using getline()

Std::string to parse each line

Check for two parameters if >2 “error” if<2 “error”

If line exists then a line needs to be included beginning with courseNumber

Int course objects

Sort course objects into data table

If course object matches file format put in that data table else keep searching

Print course info and prerequisites

Explanation

When writing code one important factor when deciding on how to run the code properly is time. Computers are fast however, they are not instant. What is meant by this is that each line of code does actually take time for a computer to interpret. The more algorithms and lines of code and different functions the longer your could will take to run completely. It is very important to keep this in mind when deciding which angle to take when designing your code and the implementation of the type of code to use. When deciding the worst case run time we keep in mind how many lines have to be read before we get to the line that is needed. In this case the amount of lines to go through was in the thousands making our runtime O(1000+). With this being said whichever code has fewer times of reading over these lines will be the quickest. The example that was the slowest would most definitely have to be the vector sorting and linked list. These two had many times of going over the same lines of data just to come up with one piece of data and then they’re were more algorithms to order the data. If we are going purely for speed I believe that the best technique to use here would be the binary search tree. The benefit of using this is that when code is read it is organized as it comes. This means that we would not be going through the same code over and over again. The code is ran through one time and then slotted into the appropriate node.