Mary Sanders

Oluwayomi Adamo

CS 300

August 11, 2023

Project One

1. **Vector Pseudocode**

INCLUDE <fstream>

openParse()

OPEN file

IF (file != OPEN || file == NULL)

OUTPUT error message

BREAK

ELSE

WHILE (endOfList != TRUE)

READ listItem

IF (itemParam < 2 || prereqName not found in courseFile)

OUTPUT error message

ELSE

PARSE each line into courseId, courseName, prereqName

CLOSE file

createCourse(parameters)

INITIALIZE createCourse() variables, including vector structure

OPEN file

WHILE (endOfList != TRUE)

APPEND course to vector

CLOSE file

searchCourse(parameters)

INITIALIZE search variables

INPUT searchCourse parameters

OPEN file

READ file

WHILE (endOfList != TRUE || courseList != NULL)

IF (vectorItem == searchItem)

OUTPUT vectorItem parameters

ELSE

OUTPUT course not found message

CLOSE file

END

1. **Hash Table Pseudocode**

INCLUDE <fstream>

openParse()

OPEN file

IF (file != OPEN || file == NULL)

OUTPUT error message

BREAK

ELSE

WHILE (endOfList != TRUE)

READ listItem

IF (itemParam < 2 || prereqName not found in courseFile)

OUTPUT error message

ELSE

PARSE each line into courseId, courseName, prereqName

CLOSE file

createCourse(parameters)

INITIALIZE createCourse() variables, including hash table structure

OPEN file

WHILE (endOfList != TRUE)

APPEND course to hash table

CLOSE file

searchCourse(parameters)

INITIALIZE search variables

INPUT searchCourse parameters

OPEN file

READ file

WHILE (endOfList != TRUE || courseList != NULL)

IF (hashItem == searchItem)

OUTPUT hashItem parameters

ELSE

OUTPUT course not found message

CLOSE file

END

1. **Binary Tree Pseudocode**

INCLUDE <fstream>

openParse()

OPEN file

IF (file != OPEN || file == NULL)

OUTPUT error message

BREAK

ELSE

WHILE (endOfList != TRUE)

READ listItem

IF (itemParam < 2 || prereqName not found in courseFile)

OUTPUT error message

ELSE

PARSE each line into courseId, courseName, prereqName

CLOSE file

CreateCourse(parameters)

INITIALIZE node variable

OPEN file

WHILE (endOfList != TRUE)

IF (node > course number)

IF (node->left == NULL)

Node->left = new node

ELSE

ADD node(node->left, course)

ELSE

IF (node->right == NULL)

Node->right = new node

ELSE

ADD node(node->right, course)

CLOSE file

CourseSearch()

INITIALIZE course variables

OPEN file

IF (course id < current courseNum)

TRAVERSE left subtree

ELSE

TRAVERSE right subtree

RETURN bid

CLOSE file

printCourse()

OPEN file

IF (node != NULL)

inOrder node->left

PRINT course info

inOrder node->right

ELSE

RETURN

CLOSE file

1. **Menu Pseudocode**

Menu()

INITIALIZE switch variable, course variables

PRINT menu options

GET userInput

WHILE (userInput != 9)

CASE 1:

CALL loadCourseData

BREAK

CASE 2:

PRINT sorted courseList

BREAK

CASE 3:

PRINT courseTitle, prereqs

BREAK

CASE 4:

EXIT

DEFAULT:

PRINT error message

PRINT menu options

BREAK

1. **Alphanumeric Sort Pseudocode**

printCourse()

OPEN file

IF (node != NULL)

inOrder node->left

PRINT course info

inOrder node->right

ELSE

RETURN

CLOSE file

1. **Evaluation**

Each of the data types given has advantages and disadvantages depending on the context of its use. Using a vector offers several benefits, primarily its rapid file reading speed and quick insertion, having the quickest runtime. However, a vector is the slowest when it comes to searching, with a worst-case runtime of O(N). Hash tables are quicker to search but are slow to sort. The programmer also needs to know the size of a hash table ahead of time. A binary search tree automatically sorts new entries as it adds them, which is why I chose to use this structure in my project.