6-2 Project One

# Justin Brown

# CS-300

Open file

IF value = -1

File not found

ELSE

File found

WHILE NOT EOF

Parse lines

IF Value < 2

RETURN ERROR

ELSE

Read parameters

IF third or more parameter is first elsewhere

Continue

ELSE

RETURN “Error”

File closed

LOOP through file

WHILE NOT EOF

FOR each line in file

FOR first and second value

Use pushback to add value to vector

IF there is a third value

Use pushback to add value until new line

LOOP through vector

IF user input = course number

PRINT course information

FOR each prerequisite of the course

PRINT the prerequisite course information

------------------------------------------------------------------------------------------------------------------------------------------

OPEN file

IF value = -1

File not found

ELSE

File found

PARSE line for course title and number

IF prerequisite is found

ADD to array

IF course parameters < 2

PRINT error

END program

ELSE

ADD course number, name, and prerequisite to hash table

IF prerequisite exists

ADD to hash table

IF prerequisite is not found

SKIP

PRINT error

CREATE constructor

WHILE file is open

STORE in hash table

CREATE constructor with parameters

INITIALIZE variables for opening file

OPEN file

WHILE file is open

PRINT course information

STORE in hash table

------------------------------------------------------------------------------------------------------------------------------------------

OPEN file

IF value = -1

File not found

ELSE

File found

WHILE NOT EOF

PARSE lines

IF value < 2

RETURN error

ELSE

PARSE parameters

IF parameter GREATER than or EQUAL to 3

CONTINUE

CLOSE file

WHILE NOT EOF

FOR each LINE in FILE

FOR first and second value

ADD course name and course ID

IF third value exists

ADD prerequisite until newline found

CREATE insert method

IF root = NULL

current course = root

ELSE

IF course number < root

ADD left

IF left = NULL

ADD course number

ELSE

IF course number < leaf

ADD to left

IF course number > leaf

ADD to right

IF course number > root

ADD right

IF right = NULL

ADD course number

ELSE

IF course number < leaf

ADD left

IF course number < leaf

ADD right

PROMPT user input

IF root != NULL

TRAVERSE left, PRINT if value found

TRAVERSE right, PRINT if value found

-----------------------------------------------------------------------------------------------------------

DISPLAY

1: Load Data Structure

2: Print Course List

3: Print Course

4: Exit

--------------------------------------------------------------------------------------------------------------------

USE Switch Case

CASE 1:

LOAD Data Structure

CASE 2:

PRINT Course List

CASE 3:

PRINT Course

CASE 4:

EXIT Program

---------------------------------------------------------------------------------------------------------------------

Mid = Low + (High-Low)/2

Pivot = courseName(Low)

WHILE courseName(Mid) < pivot

Low = Low + 1

WHILE pivot < courseName(high)

Temp = Low

Low = High

High = Temp

Mid = 0

Low = start

High = end

IF start >= end

RETURN

DISPLAY results

-------------------------------------------------------------------------------------------------------------------------------

**Vectors**

**Vectors are very efficient at reading lines and inserting new data values, especially for smaller data sets such as the one provided for this exercise. However, while Vectors are good for smaller data sets, as the size of data increases, so does the run time. The reason for this is a Vector must search each value until the desired target is found. If this project required to sort and search through one hundred or more courses, in the worst case, a Vector would have to search through each course until the end is reached.**

**--------------------------------------------------------------------------------------------------------------------Hash Tables**

**Hash Tables assign keys to values, making them great for searching through sets of data. Although the drawback to this is that these lists are often not sorted properly and often run into collisions when being created, which increases the amount of time needed for the insertion process.**

**--------------------------------------------------------------------------------------------------------------------**

**Binary Trees**

**Binary Trees are faster than Vectors when it comes to searching, as they are ordered in numerical order. Trees begin at the root and then insert values based on their numerical value. Smaller values are inserted to the left and larger values to the right. If data is already organized, then the height of the tree increases, and as the height increases, so does run time as it takes more time to descend the tree to find the desired value.**

**--------------------------------------------------------------------------------------------------------------------**

**Conclusion**

**For this specific task, a Vector is most suited in terms of time efficiency. Based on my understanding of each data structure that I have worked with; a Vector would work best here. Vectors excel at reading files and adding objects, especially with smaller data sets such as the list of courses we were given to work with for this exercise.**