Project 1

Jasmine Villarreal

Professor Oluwayomi Adamo

August 17, 2025

**Vectors Milestone**

**Pseudocode:**

**Process CVS and Validate Data**

FUNCTION Process csv (Path)

OPEN the CSV file at the given path

IF file fails to open \

THEN

DISPLAY “ERROR: unable to execute file.”

RETURN

INITIALIZE newRows as a vector of vector<string>

FOR EACH line in file:

SPLIT line by commas INTO row

ADD row TO newRows

FOR EACH row IN newRows:

IF length of row < 2

THEN

DISPLAY "ERROR: Invalid format! You must include Course Number AND Course Title,”

RETURN

INITIALIZE courseLibrary as a vector<Course>

INITIALIZE courseNumbers as a vector<string>

FOR EACH row IN newRows:

ADD row 0 TO courseNumbers // first element is course number

FOR EACH row IN newRows:

FOR index FROM 2 TO length of row - 1:

SET prereq TO row(index)

IF prereq NOT IN courseNumbers:

DISPLAY "ERROR: No prerequisites found " + prereq

RETURN

FOR EACH row IN newRows:

CALL addCourse(row, courseLibrary)

RETURN courseLibrary

**Create and ADD courses from a row**

FUNCTION addCourse(row, courseLibary)

INITILIZE newCourse AS course

SET newCourse.courseNumber to ROW 0

SET newCourse.courseName to ROW 1\

INITILIZE prerequisites as a vector<string>

For index from 2 TO length of row - 1:

IF row(index) is NOT EMPTY

ADD row(index) TO prerequisites

SET newCourse.prerequisites to prerequisites

ADD newCourse TO courseLibrary

**Search for a course by Number:**

FUNCTION SearchCourse(courseLibaray, courseNumber):  
 FOR EACH course IN course Library:

IF course.courseNumber == courseNumber:

RETURN course

RETURN an empty course // this will return if not found

**DISPLAY Course Information**

FUNCTION DisplayCourse(course):

PRINT “ Course Number, Course Name, Prerequisites”

PRINT course.courseNumber + “,” + course.courseName + “,”

IF course.prerequisites IS EMPTY:

PRINT “NOT AVAIABLE”

ELSE

FOR EACH prereq IN course.prerequisites:

PRINT prereq + “,”

**4-3 milestone Pseudocode**

**Open + Read FILE**

CREATE COURSE with couseNumber, title, prereq

CREATE Hashtable courseObject

FUNCTION LoadDataFromFile(filename)

OPEN file with NAME filename

IF file CANNOT open

THEN PRINT “ERROR”

ELSE

EXIT FUNCTION

FOR lines in FILES

SPLIT into TOKENS

IF tokens. length LESS THAN 2

THEN PRINT ‘ERROR” // this is because the line will exclude 2 parameters

CONTINUE TO NEXT LINE

courseNumber = token[0]

courseTitle = token[1]

prequesities = empty list

FOR IDEX I from 2 to tokens.length - 1

ADD tokens token [i] to prereq

CREATE Course object with courseNumber, courseTitle, prereq  
INSERT course object into hash table with key EQUALS courseNumber

END FOR

CLOSE FILE

END FUNCTION

**File Format + Prereq**

FUNCTION ValidateCourseData

FOR EACH course in hash table

FOR EACH prereq in course.prerequisites

IF prerequisite does not exist in hash table

THEN PRINT "Error: Prerequisite" + prerequisite + " not found as a course."

End FOR

END FUNCTION

**Course Object Creation + Hash Table Insertion**

STRUCT COURSE

courseNumber EQUALS String

courseTitle EQUALS String

prerequisites EQUALS List of Strings

Function InsertCourse courseNumber, courseTitle, prerequisites

Create new Course object

Set courseNumber

Set courseTitle

Set prerequisites

INSERT KEY AS courseNumber ////this will add the course into the hash table

END FUNCTION

**PRINT HASH TABLE**

FUNCTION PrintAllCourses

FOR EACH COURSE

PRINT course.courseNumber + ": " + course.courseTitle

END FUNCTION

FUNCTION PrintCourseDetails(courseNumber)

course EQUALS Search hash table for courseNumber

IF course not found

PRINT "Course not found."

EXIT FUNCTION

PRINT "Course Number: " + course.courseNumber

PRINT "Course Title: " + course.courseTitle

IF course.prerequisites IS NOT empty

THEN Print "Prerequisites: "

FOR each prereq in course.prerequisites

THEN Print prereq

ELSE

PRINT “None”

END FUNCTION

**5-3 milestone Pseudocode**

**Load the Course data for user**

FUNCTION LoadCourseData

OPEN file WITH filename

IF file CANNOT be opened

THEN PRINT “ERROR”

RETURN

FOR each line IN file

IF line IS EMPTY

THEN CONTINUE TO next line

courseData[] TO Split by comma INTO tokens

IF courseData IS LESS THAN 2

THEN PRINT “ RETRY: missing item”

CONTINUE TO NEXT LINE

VALIDATE EACH course TO prereq

CREATE NEW course OBJECT with courseNumber, courseTitle, AND prereq

INSERT Course INTO Binary Search Tree

CLOSE FILE

**Create course + insertion into bst**

CLASS :

courseNumber

courseTitle

prereq[] // this will include the # of courses planned

FUNCTION InsertINTOBST(root, newCourse):

IF root IS NULL

THEN root = new Node(newCourse)

ELSE IF newCourse.courseNumber LESS THAN root.course.courseNumber:

Root.left = Insert INTO bst

ELSE

root.right = InsertIntoBST(root.right, newCourse)

RETURN to root

**During file reading**

CREATE Course object

root = InsertIntoBST(root, Course)

**Print All Courses (In-Order Traversal)**

FUNCTION PrintAllCourses(root):

If root is null:

Return

PrintAllCourses(root.left)

Print "Course: " + root.course.courseNumber + ", " + root.course.courseTitle

If root.course.prerequisites is not empty:

Print "Prerequisites: " + Join(root.course.prerequisites, ", ")

Else:

Print "Prerequisites: None"

PrintAllCourses(root.right)

**Print Specific Search**

Function FindCourse(root, courseNumber):

If root is null:

Return null

If courseNumber == root.course.courseNumber:

Return root.course

Else If courseNumber < root.course.courseNumber:

Return FindCourse(root.left, courseNumber)

Else:

Return FindCourse(root.right, courseNumber)

Function PrintCourseInfo(courseNumber):

course = FindCourse(root, courseNumber)

If course is null:

Print "Course not found."

Else:

Print "Course: " + course.courseNumber + ", " + course.courseTitle

If course.prerequisites is not empty:

Print "Prerequisites: " + Join(course.prerequisites, ", ")

Else:

Print "Prerequisites: None"

**Menu:**

FUNCTION MainMenu

dataStructureChoice TO NONE

vectorCourses TO EMPTY

hashTable TO NULL

bstRoot TO NULL

LOOP FOREVER

PRINT "1. Load data from file into chosen data structure"

PRINT "2. Print all CS courses in alphanumeric order"

PRINT "3. Print a specific course"

PRINT "9. Exit"

opt TO READ INPUT

IF opt == 1 THEN

PRINT "Choose data structure: (V)ector / (H)ash Table / (T)ree"

ds TO READ INPUT

PRINT "Enter file path:"

path TO READ INPUT

IF ds == "V" THEN

vectorCourses TO Vector\_LoadData(path)

dataStructureChoice TO "V"

ELSE IF ds == "H" THEN

hashTable TO HT\_LoadData(path, 179)

dataStructureChoice TO "H"

ELSE IF ds == "T" THEN

bstRoot TO BST\_LoadData(path)

dataStructureChoice TO "T"

ELSE

PRINT "Invalid data structure choice."

END IF

ELSE IF opt == 2 THEN

IF dataStructureChoice == "V" THEN

Vector\_PrintAllAlphanumeric(vectorCourses)

ELSE IF dataStructureChoice == "H" THEN

HT\_PrintAllAlphanumeric(hashTable)

ELSE IF dataStructureChoice == "T" THEN

BST\_PrintInOrder(bstRoot)

ELSE

PRINT "No data loaded."

END IF

ELSE IF opt == 3 THEN

PRINT "Enter course number:"

key TO READ INPUT

IF dataStructureChoice == "V" THEN

PrintCourseInfo\_Vector(vectorCourses, key)

ELSE IF dataStructureChoice == "H" THEN

PrintCourseInfo\_HT(hashTable, key)

ELSE IF dataStructureChoice == "T" THEN

PrintCourseInfo\_BST(bstRoot, key)

ELSE

PRINT "No data loaded."

END IF

ELSE IF opt == 9 THEN

PRINT "Goodbye."

BREAK

ELSE

PRINT "Invalid option."

END IF

END LOOP

END FUNCTION

**Evaluation:**

When looking at the vector, it is very simple to use and understand. Adding courses is quick since you can just place them into the list one after another, which means loading the data only takes linear time. The downside is when you need to find a specific course because the program has to look through the whole list to find it. Printing the courses in order is also slower since the list has to be sorted first. Vectors are straightforward and do not take up much memory, but they are not the most efficient choice when the data set gets larger.

A hash table works differently and is much faster for both adding and finding courses. In most cases, it only takes constant time to do either of those operations, which makes it strong for quick searches. The trade off is that hash tables do not keep things in order naturally. If you want to print all the courses in order, you have to collect them and sort them afterwards, which takes extra time. Hash tables also use more memory and sometimes run into collisions that can slow things down.

The binary search tree falls somewhere in between and offers a good balance. It lets you add and search for courses efficiently most of the time and it naturally keeps everything in order. This means you can print the courses in alphanumeric order just by using an in order traversal, which is very efficient. The main drawback is that if the tree becomes unbalanced the performance can slow down, and it does use a little more memory because each node stores pointers to its children.

After looking at all three, the binary search tree seems like the best choice for this project. It makes it easy to search for individual courses and it produces an ordered list without needing to sort it afterwards. While a hash table is quicker for single lookups, it adds more steps when you want everything in order. A vector is the simplest structure, but it is not the most efficient once the data set grows. The tree provides the right mix of efficiency and ordered output, which makes it the most practical structure for ABCU’s advising program.