Project One

**Pseudocode & Runtime Analysis**

// This program loads data from a CSV file into the specified data structures (vector, hash table, or tree). The data consists of courses required for the Computer Science program at ABCU, including the course number, name, and any required prerequisites. Aside from loading the data from the CSV file, a user can print a list of all courses in alphanumeric order, print the name and any required prerequisites for a given course, or exit the program.

// Runtime analysis is shown to the left of each corresponding line of code of each of the three loadDataStructure functions and is formatted as follows: Line Cost \* # Times Executes = Total Cost. Total cost of the loadDataStructure functions are shown at the bottom of each function after ‘’, with total runtime analysis (worst-case) shown in Big-O notation after the ‘’ symbol, following the function representing the runtime analysis.

// NOTE: syntax may be similar to C++, as it is the programming language to be used for implementation

// include string, fstream, sstream, iostream, utility, stdexcept, vector, unordered\_map, map

// structure for each course object to be created

STRUCT **Course** {

string number

string name

vector<string> prereqs

// default constructor

Course() : number(""), name ("") {}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// **vector data structure**

vector<Course> **loadDataStructure**(const string& csvPath) { // passing by reference for efficiency

// create vectors to hold courses from CSV

1\*1=1

vector<Course> courses;

// open CSV file for reading

1\*1=1

ifstream file(csvPath);

// checks if CSV file was successfully opened

1\*1=1

if (!file.is\_open()) {

1\*1=1

throw runtime\_error ("Failed to open CSV file successfully: " + csvPath);

}

1\*1=1

string line;

n\*1=n

while (getline(file, line)) { // while not CSV EoF

1\*1=1

istringstream iss(line); // reads string as stream

1\*1=1

Course course; // create a Course instance for every line read

// verify at least 2 parameters for course exist, and store in course instance if so

1\*1=1

if (!getline(iss, course.number, ',') || !getline(iss, course.name, ',')) { // checks if parameter is missing

1\*1=1

throw runtime\_error("Requires at least 2 parameters: number, name");

}

1\*1=1

string prereq;

while (getline(iss, prereq, ',')) { // additional items stored as prereqs in course

1\*1=1

course.prereqs.push\_back(prereq); // add prereq to list of prereqs for this course

n\*1=n

}

1\*1=1

courses.push\_back(course); // add course containing only number and name to list of courses

} // all course data for each course added to courses

vector<string> courseNumbers; // creates vector of course numbers to verify prereqs as courses

1\*1=1

n\*1=n

for (const Course& course : courses) {

1\*1=1

courseNumbers.push\_back(course.number);

} // course numbers added to list

// validate prereqs exist in courseNumbers vector, else throw error

n\*1=n

for (const Course& course : courses) {

n\*1=n

for (const string& prereq : course.prereqs) {

1\*1=1

if (find(courseNumbers.begin(), courseNumbers.end(), prereq) == courseNumbers.end()) {

// reached end without finding prereq in courseNumbers list

1\*1=1

throw runtime\_error("Required prerequisite not listed: " + prereq);

}

}

} // all prereqs validated and courses list fully loaded

RETURN courses // return complete list of courses

}

// sort courses alphanumerically by course number (note that source is near sorted)

// then print the course number, course name, and any listed prerequisites for each course

void **printAllSorted**(vector<Course>& courses) { // referenced to change original vector when sorted

// insertion sort alphanumerically since dataset is small and nearly sorted

integer n = courses.size()

FOR i = 1; i < n; ++i // i = 1st element j is compared to

Course key = courses[i]

integer j = i – 1 // j = 1st element

WHILE j >= 0 && courses[j].name > key.name // compare alphanumerically

courses[j+1] = courses[j] // move courses[0..i-1] elements > key to preceding spot

j = j – 1

ENDWHILE

courses[j+1] = key

ENDFOR

// print all sorted courses

FOR const Course& course : courses

PRINT course.number + “, “ + course.name + “\n”

IF course.prereqs.size() > 0

PRINT “Prerequisites:\n“

FOR const string& prereq : course.prereqs

PRINT prereq + “\n”

ENDFOR

ENDIF

ENDFOR

}

// if searched for course number is found, then print course name and any prerequisites

void **printCourseInfo**(const vector<Course>& courses, string number) { // referenced for efficiency

FOR const Course& course : courses

IF course.number == number

PRINT course.number + “, “ + course.name + “\n”

PRINT “Prerequisites: “

IF course.prereqs.size() == 0

PRINT “None\n”

ELSE

bool isFirst = true // determines if comma is needed

FOR const string& prereq : course.prereqs

IF !isFirst

PRINT “, “

ENDIF

PRINT prereq

isFirst = false

ENDFOR

PRINT “\n”

ENDIF

RETURN // if course found and info printed, then exit function

ENDIF

PRINT “Course “ + number + “ not found\n”

ENDFOR

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// **hash table (unordered map) data structure**

// hashing each key to a unique index [no h(x)] may result in high mem overhead/resize/cache concerns

unordered\_map<string, Course> **loadDataStructure**(const string& csvPath) { // passing by reference

1\*1=1

unordered\_map<string, Course> courses // hashed via key (string); each value contains a Course

1\*1=1

ifstream file(csvPath)

1\*1=1

string line

n\*1=n

WHILE getline(file, line)

1\*1=1

istringstream iss(line)

1\*1=1

Course course

1\*1=1

IF !getline(iss, course.number, ‘,’) || !getline(iss, course.name, ‘,’)

1\*1=1

THROW runtime\_error “Requires at least 2 parameters: number, name”

ENDIF

1\*1=1

string prereq

n\*1=n

WHILE getline(iss, prereq, ‘,’)

1\*1=1

course.prereqs.push\_back(prereq)

ENDWHILE

1\*1=1

courses.insert({course.number, course})

ENDWHILE

// verify prerequisite is listed as a course

n\*1=n

FOR auto& pair : courses

n\*1=n

FOR const string& prereq : pair.second.prereqs

1\*1=1

IF courses.find(prereq) == courses.end() // prereq not found in courses

1\*1=1

THROW runtime\_error(“Required prerequisite not listed: “ + prereq

ENDIF

ENDFOR

ENDFOR

1\*1=1

1\*1=1

1\*1=1

CLOSE file

RETURN courses // return complete list of courses

}

// sort courses alphanumerically by course number (note that file source is nearly sorted already)

// then print the course number, course name, and any listed prerequisites for each course

void **printAllSorted**(const unordered\_map<string, Course>& courses) { // pair.first holds key (number)

// insertion sort alphanumerically since dataset is small and nearly sorted

integer n = courses.size()

FOR i = 1; i < n; ++i // i = 1st element j is compared to

Course key = courses[i]

integer j = i – 1 // j = 1st element

WHILE j >= 0 && courses[j].name > key.name // compare alphanumerically

courses[j+1] = courses[j] // move courses[0..i-1] elements > key to preceding spot

j = j – 1

ENDWHILE

courses[j+1] = key

ENDFOR

// print all sorted courses

FOR const Course& course : courses

PRINT course.number + “, “ + course.name + “\n”

IF course.prereqs.size() > 0

PRINT “Prerequisites:“ + “\n”

FOR const string& prereq : course.prereqs

PRINT prereq + “\n”

ENDFOR

ENDIF

ENDFOR

}

void **printCourseInfo**(const unordered\_map<string, Course>& courses, string number) {

FOR const Course& course : courses

IF courses.number == number

PRINT course.number + “, “ + course.name + “\n”

PRINT “Prerequisites: “

IF course.prereqs.size() == 0

PRINT “None\n”

ELSE

bool isFirst = true // determines if comma is needed

FOR const string& prereq : course.prereqs

IF !isFirst

PRINT “, “

ENDIF

PRINT prereq

isFirst = false

ENDFOR

PRINT “\n”

ENDIF

RETURN // if course found and info printed, then exit function

ENDIF

PRINT “Course “ + number + “ not found\n”

ENDFOR

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// **binary search tree (map) data structure**

map<string, Course> **loadDataStructure**(const string& csvPath) { // passing by reference

1\*1=1

map<string, Course> courses

1\*1=1

ifstream file(csvPath)

1\*1=1

string line

n\*1=n

WHILE getline(file, line)

1\*1=1

istringstream iss(line)

1\*1=1

Course course

1\*1=1

IF !getline(iss, course.number, ‘,’) || !getline(iss, course.name, ‘,’)

1\*1=1

THROW runtime\_error “Requires at least 2 parameters: number, name”

ENDIF

1\*1=1

string prereq

n\*1=n

WHILE getline(iss, prereq, ‘,’)

1\*1=1

course.prereqs.push\_back(prereq)

ENDWHILE

1\*1=1

courses[course.number] = course // add course to courses

ENDWHILE

// verify prerequisite is listed as a course

n\*1=n

FOR auto& pair : courses

n\*1=n

FOR const string& prereq : pair.second.prereqs

1\*1=1

IF courses.find(prereq) == courses.end() // prereq not found in courses

1\*1=1

THROW runtime\_error(“Required prerequisite not listed: “ + prereq

ENDIF

ENDFOR

ENDFOR

1\*1=1

1\*1=1

1\*1=1

CLOSE file

RETURN courses // return complete list of courses

}

// sort courses alphanumerically by course number (note that file source is nearly sorted already)

// then print the course number, course name, and any listed prerequisites for each course

void **printAllSorted**(const map<string, Course>& courses) { // pair.first holds key (number)

// insertion sort alphanumerically since dataset is small and nearly sorted

integer n = courses.size()

FOR i = 1; i < n; ++i // i = 1st element j is compared to

Course key = courses[i]

integer j = i – 1 // j = 1st element

WHILE j >= 0 && courses[j].name > key.name // compare alphanumerically

courses[j+1] = courses[j] // move courses[0..i-1] elements > key to preceding spot

j = j – 1

ENDWHILE

courses[j+1] = key

ENDFOR

// print all sorted courses

FOR const Course& course : courses

PRINT course.number + “, “ + course.name + “\n”

IF course.prereqs.size() > 0

PRINT “Prerequisites:“ + “\n”

FOR const string& prereq : course.prereqs

PRINT prereq + “\n”

ENDFOR

ENDIF

ENDFOR

}

void **printCourseInfo**(const unordered\_map<string, Course>& courses, string number) {

FOR const Course& course : courses

IF courses.number == number

PRINT course.number + “, “ + course.name + “\n”

PRINT “Prerequisites: “

IF course.prereqs.size() == 0

PRINT “None\n”

ELSE

bool isFirst = true // determines if comma is needed

FOR const string& prereq : course.prereqs

IF !isFirst

PRINT “, “

ENDIF

PRINT prereq

isFirst = false

ENDFOR

PRINT “\n”

ENDIF

RETURN // if course found and info printed, then exit function

ENDIF

PRINT “Course “ + number + “ not found\n”

ENDFOR

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// **main**

int **main**() {

const string csvPath = <path\_to\_CSV\_file.csv>

character userInput = ‘’

string number

PRINT “Welcome to the course planner.\n\n”

WHILE userInput != ‘9’

PRINT “1. Load Data Structure.\n”

PRINT “2. Print Course List.\n”

PRINT “3 Print Course.\n”

PRINT “9. Exit\n\n”

PRINT “What would you like to do? “

getline(cin, userInput)

PRINT “\n”

SWITCH userInput

CASE ‘1’:

TRY

auto courses = loadDataStructure(csvPath)

CATCH

PRINT “auto courses = loadDataStructure(csvPath)” + e.what()

ENDTRY

CASE ‘2’:

TRY

printAllSorted(courses)

CATCH

PRINT “printAllSorted(courses): “ + e.what()

ENDTRY

CASE ‘3’:

PRINT “What course do you want to know about? ”

getline(cin, number)

TRY

printCourseInfo(courses, number)

CATCH

PRINT “printCourseInfo(courses, number) + e.what()

ENDTRY

CASE ‘9’:

BREAK;

DEFAULT:

PRINT userInput + “ is not a valid option.\n\n”

ENDSWITCH

ENDWHILE

PRINT “Goodbye”

RETURN 0 // execution terminated without error

}