**6-2 Project 1: Menu and Output Pseudocode**

Program Loads

WHILE true {

Four Options are provided:

1. Load Bids
2. Display All Courses
3. Display Single Course

9. Exit

OUTPUT “Enter choice: “

INPUT a numerical option

IF input != option {

OUTPUT “Try 1, 2, 3, or 9.”

return

}

OPTION 1:

***// Loads courses based on Vector, Hash Table, or Binary Tree structure // found below***

Loads bids from CSV file eBid\_Monthly\_Sales.csv

OUTPUT all nodes in the file “ bids read” newline

OUTPUT “time: “ ticks type integer “ clock ticks” newline

OUTPUT “time: “ ticks type integer “ seconds” newline

Goes back to options

***NOTE: CSV FILE MUST BE LOADED, OR NO COURSES DISPLAYED***

OPTION 2:

***// Display all courses depending on Vector, Hash Table, or Binary Tree’s // displayCourses() function and already sorted from load***

***// However, there is a sortCourses() function I made at the bottom***

Call displayCourses() Function

Goes back to options

OPTION 3:

***// Search function differs based on list type***

Call searchCourse() function

OUTPUT tempNode course information

***// course number: course title (prerequisite 1, prerequisite 2)***

Goes back to options

OPTION 9:

EXITS the program

OUTPUT “Good bye.”

Closes Program

**Vector**

**Vector<Course> courses** { (Vector Creation)

Load CSVcourse file

(A comma separates each parameter/parses each line)

The first parameter becomes the header (course.courseNumber)

The second parameter gets added to the header (course.courseTitle)

WHILE the additional parameters are LESS THAN the total parameters {

IF the current parameter has characters GREATER THAN 10 {

Course name EQUALS the current parameter (course.courseTitle)

Course name added to the current course

}

IF the current parameter EQUALS a course number {

Prerequisite EQUALS the current parameter (course.coursePrereq)

Prerequisite added to the current course

}

ELSE {

Current parameter IS the new course in list (course.courseNumber)

}

}

}

**displayCourses(Course course)** {

FOR each course in courses {

IF course < 2 parameters {

OUTPUT “Course has less than two parameters!”

}

ELSE {

Print out the course information

}

}

}

(Example:

CSCI100 Introduction to Computer Science

CSCI101 Introduction to Programming in C++ CSCI100

CSCI200 Data Structures CSCI101

MATH201 Discrete Mathematics

CSCI300 Introduction to Algorithms CSCI200 MATH201

CSCI301 Advanced Programming in C++ CSCI101

CSCI350 Operating Systems CSCI300

CSCI400 Large Software Development CSCI301 CSCI350

)

**void searchCourse(Vector<Course> courses, String courseNumber)** {

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the course

print the prerequisite course information

}

**Hash Table**

**unsigned int HashTable::hashFun(int key) {**

**return key % 10**

**}**

**HashTable<String, String> courses = new HashTable<>(8) { (courses table created with 8 buckets)**

Load CSVcourse file

FOR all keys

hash = 0

Use hashCode() FUNCTION to get key’s hash value for a string

((s[0] \* 31 ^ (n – 1)) + (s[1] \* 31 ^ (n – 2)) + … + s(n – 1))

this.hash = key’s hash code value

Make curCor the current node at 0

For all nodes made so far {

IF this.hash > 2000000000 {

curCor.course.cTit = parameter

}

ELSE IF parameter matches a prior node’s key {

IF curCor.course.cPre1 = NULL {

curCor.course.cPre1 = parameter

}

ELSE {

curCor.course.cPre2 = parameter

}

}

ELSE IF parameter does not match any prior node’s key {

curCor.key = parameter

Make a new node called newCor

curCor->newCor

bucket = hashFun(this.hash)

int bucketsProbed = 0

WHILE bucketsProbed < 8 {

IF hashtable[bucket] = NULL {

Hashtable[bucket] = course

}

bucket = (bucket + 1) % 10

++bucketsProbed

}

}

}

}

}

( Keys and their hashCode() values

CSCI100 1773523291

CSCI101 1773523292

CSCI200 1773524252

MATH201 1558971819

CSCI300 1773525213

CSCI301 1773525214

CSCI350 1773525368

CSCI400 1773526174

)

( Organized buckets

0 MATH201

1 CSCI100

2 CSCI101

3 CSCI200

4 CSCI300

5 CSCI301

6 CSCI350

7 CSCI400

)

**displayCourses(Course course)** {

Make curCor the current course at 0

bucketsProbed = 0

WHILE bucketsProbed < 8 {

IF course < 2 parameters {

OUTPUT “Course has less than two parameters!”

}

ELSE {

OUTPUT curCor’s key, title, prerequisite 1 & 2

}

curCor->next

++bucketsProbed

}

}

(Example:

CSCI100 Introduction to Computer Science

CSCI101 Introduction to Programming in C++ CSCI100

CSCI200 Data Structures CSCI101

MATH201 Discrete Mathematics

CSCI300 Introduction to Algorithms CSCI200 MATH201

CSCI301 Advanced Programming in C++ CSCI101

CSCI350 Operating Systems CSCI300

CSCI400 Large Software Development CSCI301 CSCI350

)

**void searchCourse(HashTable<Course> courses, String) {**

int key = hash(course)

int bucketsProbed = 0

Make curCor the current node

IF curCor != nullptr and curCor->course.key = key {

Return curCor->course

}

WHILE curCor != nullptr {

IF curCor->course.key = key {

Return curCor.course.key

}

curCor = curCor->next

}

}

**Binary Tree**

Struct Courses {

String cNum

String cTit

String cPre1

String cPre2

}

Struct Node {

Courses courses

Node \*left

Node \*right

}

void addNode(Node\* node, Courses course) {

IF root == nullptr {

Root = new Node(course)

}

Node\* curNode = root

IF curNode->right == nullptr {

curNode->right = new Node(course)

}

ELSE {

curNode = curNode->right

}

}

void insertCourses(Courses course) {

IF root == nullptr {

root = new Node(course)

}

ELSE {

addNode(root, course)

}

}

void loadCourses(Node node, Courses course) {

For all files in the CSV file {

hash = 0

Use hashCode() FUNCTION to get course’s cNum hash value for a string

((s[0] \* 31 ^ (n – 1)) + (s[1] \* 31 ^ (n – 2)) + … + s(n – 1))

hash = course.cNum

IF root = NULL {

root = node(course)

node->left = NULL

node->right = NULL

}

ELSE {

WHILE root != NULL {

IF this.hash > 2000000000 {

curCor.course.cTit = parameter

}

ELSE IF parameter matches a prior node’s key {

IF curCor.course.cPre1 = NULL {

curCor.course.cPre1 = parameter

}

ELSE {

curCor.course.cPre2 = parameter

}

}

ELSE {

FOR new nodes {

curCor = root

IF node’s hash < curCor’s hash {

IF curCor->left = NULL {

curCor->left = node

curCor = NULL

}

ELSE {

curCor = curCor->left

}

ELSE {

IF curCor->right = NULL

curCor->right = node

curCor = NULL

}

ELSE {

curCor = curCor->right

}

}

}

Node->left = NULL

Node->right = NULL

}

}

}

}

Try {

FOR each course in courses {

Courses courses

Courses.cNum = file [i][1]

Courses.cTit = file [i][0]

Courses.cPre1 = file [i][2]

Courses.cPre2 = file [i][3]

Bst->insertCourses(courses)

}

}

Catch ERROR {

OUTPUT “Files not loaded correctly.”

}

}

displayCourses(Courses courses) {

nNum = 1

FOR all courses {

IF course < 2 keys {

OUTPUT “Course has less than two parameters!”

}

ELSE {

OUTPUT nNum << “. “ << courses.cNum << “ “ << courses.cTit << “ “ courses.cPre1 << “ “ << courses.cPre2 << end newline

}

nNum++

}

}

(Example:

1. CSCI100 Introduction to Computer Science
2. CSCI101 Introduction to Programming in C++ CSCI100
3. CSCI200 Data Structures CSCI101
4. MATH201 Discrete Mathematics
5. CSCI300 Introduction to Algorithms CSCI200 MATH201
6. CSCI301 Advanced Programming in C++ CSCI101
7. CSCI350 Operating Systems CSCI300
8. CSCI400 Large Software Development CSCI301 CSCI350

)

void searchCourse(HashTable<Course> courses, String cNum) {

curCor = root

FOR all courses {

IF search = curCor->course.cNum {

return curCor

}

ELSE IF search < curCor->course.cNum {

curCor->left

}

ELSE IF search > curCor->course.cNum {

curCor->right

}

}

}

sortCourses(Node node) {

hash = 0

FOR indices < list size - 1 {

FOR each index < list size – indices – 1 {

IF hashCode(course.cNum) > hashCode(next->course.cNum) {

Create a tempNode

tempNode = current Node

current Node = next Node

next Node = tempNode

}

}

}

}