Class Course {

String courseNumber;

String courseName;

Vector<String> prerequisites;

}

Class HashTable {

struct Node {

Course\* course;

Node\* next;

int key;

Node() {

key = 0;

next = nullptr;

}

Node(Course\* c) {

course = c;

}

Node(Course\* c, int k) {

course = c;

key = k;

}

int size = 0;

}

}

struct Node {

Course\* course;

Node\* left;

Node\* right;

Node() {

left = nullptr;

right = nullptr;

}

Node(Course\* c) : Node() {

course = c;

}

}

class BinarySearchTree {

Node\* root;

}

While (lines to read > 0) {

Split the line up into a vector named ’args’ using commas as the split character

If the size of the vector is greater than or equal to 2, it has at least a course number and a course name {

Create a new course using the courseNumber and courseName

}

Else, return error

If there are course prerequisites {

For each course prerequisite {

Add prerequisite to the new course

}

}

}

// Sort (Vector)

void sortVector(Vector<Course> courses) {

For (i = 1; i < size of course vector; i++) {

Create a variable j which is equal to i initially

While j is less than courses.at(j – 1) {

Swap the values into the correct order

}

}

For each course in the courses vector {

Print the course information in sorted order

}

}

// Print number of course prerequisites (Vector)

int numPrerequisiteCourses(Vector<Course> courses) {

Create an integer variable to store the total number of prerequisites

For each course in the vector courses {

Increment total prerequisites by the number of prerequisites in the current course

}

}

// Print course information (Vector)

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

For each course in the courses vector {

If the given course number matches {

Print the course information

}

}

}

// Print number of course prerequisites (Hash table)

int numPrerequisiteCourses(Hashtable<Course> courses) {

int totalPrerequisites = 0;

For each course in courses {

totalPrerequisites += size of course.prerequisites vector

}

return totalPrerequisites;

}

// Print course information (Hash table)

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

Hash the courseNumber and store it in a variable representing the bucket

Set the number of buckets probed to zero

While (bucket is not empty since start && the number of buckets probed is less than N) {

if (the bucket is not empty && the bucket's courseNumber matches search query) {

Print the course number, the course title, and course prerequisites

}

Increment the index of the bucket

Increment the number of buckets probed

}

Return null when nothing is found

}

// Print the number of course prerequisites (Binary Tree)

void numPrerequisiteCourses(Node\* node) {

If there is not a node anymore, return and exit the function

Recursively execute this function on the node's left subnode

Add the left node's course prerequisites to the prerequisites vector

Add the right node's course prerequisites to the prerequisites vector

Recursively execute this function on the node's right subnode

}

// Print course information (Binary Tree)

void printCourseInformation(Node\* node, String courseNumber) {

Start with the root node of the tree

While there is still a node that exists in the tree {

If the current node's course's course number matches the given courseNumber {

Print the matching node's course's title, course number, and prerequisites

Return and exit the function

}

If the given courseNumber is less than the current node's course's course number {

Move on to the left subnode and loop again

} else {

Move on to the right subnode and loop again

}

}

Display that the course was not found if no node has the specified course

}

void DisplayMenu() {

Print option number one to load data into the Course data structure

Print option number two to print an alphanumerically sorted list of the courses

Print option number three to print a specific course and its prerequisites

Print option number four which exits the program

}