# CS 300 Pseudocode Document

## Example Function Signatures

Below is an example of a function signature that you can use as a guide to help address the program requirements using each data structure for the milestones. The pseudocode for finding and printing course information is also given below and depicted in bold to help you get started. The provided pseudocode is for a vector data structure, so you may use this pseudocode in your first milestone as is. The hash table and tree structures are also shown below. But these structures are left for you to do in future milestones.

//Vector - Milestone 1

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 - Milestone 2

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

}

//Binary Search Tree – Milestone 3

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

## Example Runtime Analysis

When you are ready to analyze the runtime for the Project One data structures for which you created the pseudocode, use the example chart below to support your work. This particular example is for printing course information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document. The example only covers the search function for the vector structure. You do not have to complete your runtime analysis until Project One. However, working on your analysis now may help you understand the changes as you complete the milestones. Don’t forget to include your charts in Project One. You will submit Project One in Module Six.

Vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **For each course in vector** | 1 | n | n |
| **Compare course number** | 1 | n | n |
| **for each prerequisite of the course** | 1 | 1 | 1 |
| Print prerequisite | 1 | n | n |
| **Total Cost** | | | 4n |
| **Runtime** | | | O(n) |

Hash Table

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Hash key lookup** | 1 | 1 | 1 |
| Compare course number in bucket | 1 | k | k |
| **Print prerequisites** | 1 | p | p |
| **Total Cost** | | | k + p + 1 |
| **Runtime** | | | O(1) average, O(n) worst case |

Binary Search Tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Traverse tree** | 1 | log(n) | log(n) |
| Compare node | 1 | log(n) | log(n) |
| **Print prerequisites** | 1 | p | p |
| **Total Cost** | | | log(n)+p |
| **Runtime** | | | O(log n) average, O(n) worst case |

Advantages and Disadvantages

Vector

* **Advantages:** Simple to implement, sequential memory for fast iteration, built-in sort functions.
* **Disadvantages:** Searching is O(n), insertion/deletion expensive unless at end, not ideal for very large datasets.

Hash Table

* **Advantages:** O(1) average case for search and insert, very fast lookups.
* **Disadvantages:** Requires good hash function, collisions can degrade performance to O(n), more memory overhead.

Binary Search Tree

* **Advantages:** Keeps data sorted automatically, efficient O(log n) searches and traversals when balanced.
* **Disadvantages:** Unbalanced trees degrade to O(n), more complex to implement and maintain.

### Recommendation

For ABCU’s advising program, I recommend using a **Hash Table** for the final implementation. The ability to insert and search in **constant time on average** will make looking up individual courses and prerequisites extremely fast. Even though a Binary Search Tree offers sorted traversal naturally, the sorting requirement for Option 2 can be handled by simply copying hash table values into a temporary list and sorting when needed, which is still efficient for the dataset size. The vector is easy to use, but its linear search time makes it less optimal when fast lookups are a priority.

Project One – Pseudocode and Runtime Analysis

CS 300 – Data Structures and Algorithms   
Nicholas Justus

Vector Implementation – Pseudocode

Load and Validate Courses from File

FUNCTION loadCoursesFromFile(filePath):

    CREATE empty vector called courseList

    CREATE empty set called courseNumbers

    OPEN file at filePath for reading

    FOR each line in file:

        SPLIT line by commas into tokens

        IF number of tokens < 2:

            PRINT "Invalid line: not enough data"

            CONTINUE to next line

        CREATE new Course object

        SET course.courseNumber = tokens[0]

        SET course.name = tokens[1]

        FOR each token from index 2 to end:

            ADD token to course.prerequisites

        APPEND course to courseList

        ADD course.courseNumber to courseNumbers

    // Validate prerequisites

    FOR each course in courseList:

        FOR each prereq in course.prerequisites:

            IF prereq not in courseNumbers:

                PRINT "Missing prerequisite course for " + course.courseNumber + ": " + prereq

    RETURN courseList

Search and Print Course Information

FUNCTION searchCourse(courseList, courseNumber):

    FOR each course in courseList:

        IF course.courseNumber == courseNumber:

            PRINT course.courseNumber + ": " + course.name

            IF prerequisites is empty:

                PRINT "Prerequisites: None"

            ELSE:

                PRINT "Prerequisites: "

                FOR each prereq in prerequisites:

                    PRINT "- " + prereq

            RETURN

    PRINT "Course not found"

Menu

FUNCTION main():

    DECLARE courseList as empty vector

    WHILE True:

        PRINT menu options:

            1. Load Data Structure

            2. Print Course List

            3. Print Course

            9. Exit

        GET userChoice

        IF userChoice == 1:

            courseList = loadCoursesFromFile("courses.txt")

        ELSE IF userChoice == 2:

            printCourseList(courseList)

        ELSE IF userChoice == 3:

            GET courseNumber from user

            searchCourse(courseList, courseNumber)

        ELSE IF userChoice == 9:

            BREAK

        ELSE:

            PRINT "Invalid option"

Hash Table Implementation – Pseudocode

Load and Validate Courses from File

FUNCTION loadCoursesFromFile(filePath):

    CREATE empty HashTable called courseTable

    OPEN file at filePath for reading

    FOR each line in file:

        SPLIT line by commas into tokens

        IF number of tokens < 2:

            PRINT "Invalid line: not enough data"

            CONTINUE

        CREATE new Course object

        SET course.courseNumber = tokens[0]

        SET course.name = tokens[1]

        FOR each token from index 2 to end:

            ADD token to course.prerequisites

        INSERT course into courseTable with key = course.courseNumber

    // Validate prerequisites

    FOR each course in courseTable:

        FOR each prereq in course.prerequisites:

            IF prerequisite not found in courseTable:

                PRINT "Missing prerequisite course for " + course.courseNumber + ": " + prereq

    RETURN courseTable

Search and Print Course Information   
   
FUNCTION searchCourse(courseTable, courseNumber):

    course = courseTable.search(courseNumber)

    IF course not found:

        PRINT "Course not found"

        RETURN

    PRINT course.courseNumber + ": " + course.name

    IF prerequisites is empty:

        PRINT "Prerequisites: None"

    ELSE:

        PRINT "Prerequisites:"

        FOR each prereq in prerequisites:

            PRINT "- " + prereq

Sort and Print Courses Alphanumerically

FUNCTION printCourseList(courseTable):

    CREATE tempList from all courses in courseTable

    SORT tempList by courseNumber in ascending order

    FOR each course in tempList:

        PRINT course.courseNumber + ": " + course.name

Menu

FUNCTION main():

    DECLARE courseTable as empty HashTable

    WHILE True:

        PRINT menu options:

            1. Load Data Structure

            2. Print Course List

            3. Print Course

            9. Exit

        GET userChoice

        IF userChoice == 1:

            courseTable = loadCoursesFromFile("courses.txt")

        ELSE IF userChoice == 2:

            printCourseList(courseTable)

        ELSE IF userChoice == 3:

            GET courseNumber from user

            searchCourse(courseTable, courseNumber)

        ELSE IF userChoice == 9:

            BREAK

        ELSE:

            PRINT "Invalid option"

Binary Search Tree Implementation – Pseudocode

Load and Validate Courses from File

FUNCTION loadCoursesFromFile(filePath):

    CREATE empty BinarySearchTree called courseTree

    OPEN file at filePath for reading

    FOR each line in file:

        SPLIT line by commas into tokens

        IF number of tokens < 2:

            PRINT "Invalid line: not enough data"

            CONTINUE

        CREATE new Course object

        SET course.courseNumber = tokens[0]

        SET course.name = tokens[1]

        FOR each token from index 2 to end:

            ADD token to course.prerequisites

        INSERT course into courseTree using course.courseNumber as key

    // Validate prerequisites

    FOR each course in courseTree (in-order traversal):

        FOR each prereq in course.prerequisites:

            IF prerequisite not found in courseTree:

                PRINT "Missing prerequisite course for " + course.courseNumber + ": " + prereq

    RETURN courseTree

Search and Print Course Information

FUNCTION searchCourse(courseTree, courseNumber):

    course = courseTree.search(courseNumber)

    IF course not found:

        PRINT "Course not found"

        RETURN

    PRINT course.courseNumber + ": " + course.name

    IF prerequisites is empty:

        PRINT "Prerequisites: None"

    ELSE:

        PRINT "Prerequisites:"

        FOR each prereq in prerequisites:

            PRINT "- " + prereq   
 

Sort and Print Courses Alphanumerically

FUNCTION printCourseList(courseTree):

    PERFORM in-order traversal of courseTree

    FOR each course visited:

        PRINT course.courseNumber + ": " + course.name

Menu

FUNCTION main():

    DECLARE courseTree as empty BinarySearchTree

    WHILE True:

        PRINT menu options:

            1. Load Data Structure

            2. Print Course List

            3. Print Course

            9. Exit

        GET userChoice

        IF userChoice == 1:

            courseTree = loadCoursesFromFile("courses.txt")

        ELSE IF userChoice == 2:

            printCourseList(courseTree)

        ELSE IF userChoice == 3:

            GET courseNumber from user

            searchCourse(courseTree, courseNumber)

        ELSE IF userChoice == 9:

            BREAK

        ELSE:

            PRINT "Invalid option"

References

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data structures and algorithms in C++* (2nd ed.). Wiley.

Knuth, D. E. (1997). *The art of computer programming, Volume 1: Fundamental algorithms* (3rd ed.). Addison-Wesley.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.