

CS 300 Project One

Curtis Felsher



February 8, 2023

# CS 300 Pseudocode Document

// Vector Pseudocode

**Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

Open file with fstream

Define function to parse the file path as a parameter

Open function to open file at file path and assign to variable

Create variable courseList to store all courses from file

Create variable to track current line in file

For loop to iterate each file line

Increment line variable by 1

Check for two parameters on line

If not, throw error

Check for prerequisites

Append current course to courseList variable

Return courseList

Close file

**Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

**Create empty vector called courses**

**Open file**

**While loop if lines left in file**

**Read one line at a time from file**

**Utilize split() function to split line**

**Create new course object**

**Set course object variables to tokens generated from split()**

**Add course object to courses vector**

**Close file**

**Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites**.

Create function searchCourse that takes in str courseName

Create Boolean called found and set it to false

For loop to iterate through vector called courses and increment i until bigger than vector

If statement to check if course in vector is equal to courseName

If true, print course info formatted

If statement to check for course prerequisites

If any are found, print prerequisites

If not found

Print course not found

// Hash Table

**Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

**Create class Course**

**Assign variables courseNumber, courseName, and coursePrerequisites(multiple)**

**Open file with fstream**

**While statement to read data from the file**

**Parse each line and check for courseNumber, courseName, and coursePrerequisites(multiple)**

**If statement to ensure there are at least two parameters on each line**

**Print error if less than two parameters**

**End**

**If statement to ensure that any prerequisite exists as a course**

**Print error is prerequisite course does not exist**

**End**

**If statement to check for duplicate course numbers**

**Print error is duplicate exists**

**End**

**If statement to check for null course name**

**Print error if course title is missing**

**End**

**Store data in hash table**

**Add courseNumber and courseName to hash table as a key pair**

**If statement to check for coursePrerequisites does not equal null**

**Add coursePrerequisites to hashtable as key to courseNumber**

**Print Hash table data**

**For each key within the hashtable**

**Print key and its value**

**End**

**Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

**Utilize class Course**

**Variables: courseNumber, courseName, and coursePrerequisites(multiple)**

**Initialize an empty hash table for storing course objects labeled courses**

**Open file for reading**

**For loop to iterate through each file line**

**Utilize split() function to split line into tokens**

**Create new course object utilizing the token data**

**Store the course object in the hash table by using the course number as the key**

**Close file**

**Design pseudocode that will print out course information and prerequisites.**

**Utilize class Course**

**Variables: courseNumber, courseName, and coursePrerequisites(multiple)**

**Create function printCourseInfo**

**If statement to check if course number already exists in the hash table**

**Retrieve course object from hash table**

**Print course info with formatting**

**Else**

**Print error if no course number found**

// Binary Search Tree

1. **Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

Create Course structure

Create Node structure for binary search tree

Open file with fstream

If statement to check if file is not open

Output opening file error

While loop to read data from file by line

Split line into courseNumber, title, and prerequisites

IF statement to check for at least two parameters per line

Output error

Store course datra in binary search tree

While statement to validate prerequisites

IF statement prerequisites not present

Output format error

Add prerequisites to courseNumber

Close file

1. **Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

Create course class

courseNumber string

course string

vector to store prerequisites

Create constructor to initialize course object

Create function to insert course object into binary search tree

Use courseNumber as key to insert course object into tree

While loop to parse each line for data

Split line into courseNumber, title, and prerequisites

Store all prerequisites into prerequisites vector

Create object course with courseNumber, title, and prerequisites

Insert course object into tree

Close file

1. **Design pseudocode that will print out course information and prerequisites**.

Create function to print course info and its prerequisites

Search through the tree for course object

Output courseNumber and title formatted

If statement to check for empty prerequisites

Output prerequisites: none

Else statement

Output prerequisites formatted

End

// Create Menu

Print “ABCU Course Planner”

Set string equal to 0

Print

“1. Load data structure”

“2. Print Course List”

“3. Print Course”

“4. Exit”

“Please choose an option”

Input user choice

Switch (choice)

Case 1.

Read data from file and load into appropriate data structure

Break

Case 2.

Sort data within structure by alphabetical order

Output formatted data

Break

Case 3.

Output “Enter course ID”

If courseID not equal to null and exists

Output course title and prerequisites

Else

Output “Please try again”

Break

Case 4.

Exit program

Break

Default:

Invalid choice. Please choose from either 1, 2, 3, or 4”

END

**Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

**//Vector**

**Sort vector based on courseNumber**

**Output Courses in the computer science program:**

**For each course in the sorted vector**

**Output “CourseNumber, title, and all prerequisites”**

**//Hash Table**

**Create array of keys from the hash table using courseNumber to generate key**

**Sort key array in order**

**Output Courses in the computer science program:**

**For each key within the sorted key array**

**Retrieve course info using generated key**

**Output “Key, title, and all prerequisites”**

**//Binary Search Tree**

**Create empty array to store course info**

**Traverse the tree in-order to add courses to array**

**Output Courses in the computer science program:**

**For each course stored in the array**

**Output “CourseNumber, title, and all prerequisites”**

## Example Runtime Analysis

When you are ready to begin analyzing the runtime for the data structures that you have created pseudocode for, use the chart below to support your work. This 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.

// Vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open file with fstream | 1 | 1 | 1 |
| Define function to parse the file path as a parameter | 1 | 1 | 1 |
| Open function to open file at file path and assign to variable | 1 | 1 | 1 |
| Create variable courseList to store all courses from file | 1 | 1 | 1 |
| Create variable to track current line in file | 1 | 1 | 1 |
| For loop to iterate each file line | 1 | n | n |
| Increment line variable by 1 | 1 | n | n |
| Check for two parameters on line | 1 | n | n |
| If not, throw error | 1 | 1 | 1 |
| Check for prerequisites | 1 | n | n |
| Append current course to courseList variable | 1 | N | N |
| Return courseList | 1 | 1 | 1 |
| Close file | 1 | 1 | 1 |
| **Show how to create course objects** | NA | NA | NA |
| **Create empty vector called courses** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **While loop if lines left in file** | 1 | N | N |
| **Read one line at a time from file** | 1 | N | N |
| **Utilize split() function to split line** | 1 | N | N |
| **Create new course object** | 1 | 1 | 1 |
| **Set course object variables to tokens generated from split()** | 1 | N | N |
| **Add course object to courses vector** | 1 | N | N |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 10N + 12 |
| **Runtime** | | | O(n) |

// Hash Table

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create class Course** | 1 | 1 | 1 |
| **Assign variables courseNumber, courseName, and coursePrerequisites(multiple)** | 1 | 1 | 1 |
| **Open file with fstream** | 1 | 1 | 1 |
| **While statement to read data from the file** | 1 | N | N |
| **Parse each line and check for courseNumber, courseName, and coursePrerequisites(multiple)** | 1 | N | N |
| **If statement to ensure there are at least two parameters on each line** | 1 | N | N |
| **Print error if less than two parameters** | 1 | 1 | 1 |
| **If statement to ensure that any prerequisite exists as a course** | 1 | N | N |
| **Print error is prerequisite course does not exist** | 1 | 1 | 1 |
| **If statement to check for duplicate course numbers** | 1 | N | N |
| **Print error is duplicate exists** | 1 | 1 | 1 |
| **If statement to check for null course name** | 1 | N | N |
| **Print error if course title is missing** | 1 | 1 | 1 |
| **Add courseNumber and courseName to hash table as a key pair** | N | N | N^2 |
| **If statement to check for coursePrerequisites does not equal null** | 1 | N | N |
| **Add coursePrerequisites to hashtable as key to courseNumber** | N | N | N^2 |
| **Print Hash table data** | 1 | 1 | 1 |
| **For each key within the hashtable** | 1 | N | N |
| **Print key and its value** | 1 | 1 | 1 |
| **Show how to create course objects** | NA | NA | NA |
| **Utilize class Course** | 1 | 1 | 1 |
| **Variables: courseNumber, courseName, and coursePrerequisites(multiple)** | 1 | 1 | 1 |
| **Initialize an empty hash table for storing course objects labeled courses** | 1 | 1 | 1 |
| **Open file for reading** | 1 | 1 | 1 |
| **For loop to iterate through each file line** | 1 | N | N |
| **Utilize split() function to split line into tokens** | 1 | N | N |
| **Create new course object utilizing the token data** | 1 | N | N |
| **Store the course object in the hash table by using the course number as the key** | N | N | N^2 |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 3n^2 + 11n + 14 |
| **Runtime** | | | O(3n^2) |

//Binary Search Tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create Course structure | 1 | 1 | 1 |
| Create Node structure for binary search tree | 1 | 1 | 1 |
| Open file with fstream | 1 | 1 | 1 |
| If statement to check if file is not open | 1 | 1 | 1 |
| Output opening file error | 1 | 1 | 1 |
| While loop to read data from file by line | 1 | N | n |
| Split line into courseNumber, title, and prerequisites | 1 | N | n |
| IF statement to check for at least two parameters per line | 1 | N | n |
| Output error | 1 | 1 | 1 |
| Store course data in binary search tree | 1 | N | n |
| While statement to validate prerequisites | 1 | N | n |
| IF statement prerequisites not present | 1 | 1 | 1 |
| Output format error | 1 | 1 | 1 |
| Add prerequisites to courseNumber | 1 | n | n |
| Close file | 1 | 1 | 1 |
| **Show how to create course objects** | NA | NA | NA |
| Create course class | 1 | 1 | 1 |
| courseNumber string | 1 | 1 | 1 |
| course string | 1 | 1 | 1 |
| vector to store prerequisites | 1 | 1 | 1 |
| Create constructor to initialize course object | 1 | 1 | 1 |
| Create function to insert course object into binary search tree | 1 | N | n |
| Use courseNumber as key to insert course object into tree | 2 | 1 | 1 |
| While loop to parse each line for data | 1 | N | n |
| Split line into courseNumber, title, and prerequisites | 1 | N | n |
| Store all prerequisites into prerequisites vector | 1 | N | n |
| Create object course with courseNumber, title, and prerequisites | 1 | 1 | 1 |
| Insert course object into tree | 1 | N | n |
| Close file | 1 | 1 | 1 |
| **Total Cost** | | | 11n + 17 |
| **Runtime** | | | O(n) |

**Based on the advisor’s requirements, analyze each data structure (vector, hash table, and tree).** **Explain the advantages and disadvantages of each structure in your evaluation.**

**Vectors have many pros and cons. A pro of using a vector is the ability to be resizable. They can grow and shrink depending on the need. Vectors are also easy to implement and use for common operations. A con of a vector is the runtime. It can be slower due to way the algorithm loops and performance suffers due to the memory overhead required.**

**Hash tables also have many pros and cons. A few pros of a hash table include better synchronization vs other data structures, higher efficiency versus something like a BST, and high-speed data manipulation. Hash tables do also suffer from cons. A few of these include inefficiency when collisions occur, null values are not allowed, and can be complex to implement.**

**A pro of a BST include extremely fast insertion and deletion times if nodes are properly balanced. Searching within a tree is also extremely fast due to low complexity. A BST is also simple compared to other structures such as a hash table and elements can be automatically sorted as they are inserted into the BST. Cons of a BST include poor performance for data sets that need to be accessed randomly and must be balanced for performance to not degrade.**

**Now that you have analyzed all three data structures, make a recommendation for which data structure you will plan to use in your code**. **Provide justification for your recommendation, based on the Big O analysis results and your analysis of the three data structures**.

For this project, I would recommend the implementation of a binary search tree for data storage. Binary search tree traversal happens in order without the need for sorting since it is automatic. This is a great option since the courses need to be sorted in alphanumeric order. The other options would need this functionality to be implemented. Traversal with a binary search tree is quicker (unlike a vector, which is slower) so performance will not suffer with the use of a BST.