# CS 300 Pseudocode Document

****

The course data is as follows:

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

**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**

// readFile – Milestone 3 (BST Implementation)

BST<Course> readFile(String filename) {

**// Create an empty Binary Search Tree for Course objects, ordered by Course.ID**

**Create myDataStructure as an empty BST<Course>**

**Declare readVal as String**

**try {**

**Open file using filename**

**while file has more lines:**

**set readVal = read next line from file**

**// Validate format: ensure there is at least one comma separating courseID and courseName**

**If countOccurrences(readVal, ',') < 1:**

**Print "Warning: Malformed line: " + readVal**

**continue // Skip to the next line**

**// Create a Course object from the current line**

**course = CourseFactory(readVal)**

**// Insert the course into the BST using Course.ID as the key for ordering**

**InsertNode(myDataStructure, course)// function below**

**}**

**catch FileNotFoundException:**

**Print "Error: Could not open file."**

**finally:**

**Close file**

**return myDataStructure**

}

InsertNode(myDataStructure, course){

**If tree.root is null:**

**tree.root = course**

**course.left = null**

**course.right = null**

**Return**

**current = tree.root**

**While current is not null:**

**If course.ID < current.ID:**

**If current.left is null:**

**current.left = course**

**course.left = null**

**course.right = null**

**Return**

**Else:**

**current = current.left**

**Else:**

**If current.right is null:**

**current.right = course**

**course.left = null**

**course.right = null**

**Return**

**Else:**

**current = current.right**

}

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

// CourseFactory – Milestone 3

Course CourseFactory(String readVal) {

**Create new Course:**

**ID = ""**

**Name = ""**

**Prerequisites = empty vector**

**Split readVal by ',' into tokens**

**If tokens.length < 2:**

**Throw error or return null**

**Set Course.ID = tokens[0]**

**Set Course.Name = tokens[1]**

**For i = 2 to tokens.length - 1:**

**Append tokens[i] to Course.Prerequisites**

**return Course**

}

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

// searchCourse – Milestone 3 (BST Implementation)

Node\* seekCourse( myDataStructure, desiredKey){

**If myDataStructure is null:**

**Return null**

**If myDataStructure’s key == desiredKey:**

**Return node**

**Else if desiredKey < myDataStructure’s key:**

**Return seekCourse(myDataStructure’s left, desiredKey)**

**Else:**

**Return seekCourse(myDataStructure’s right, desiredKey)**

}

void searchCourse(BST<Course> myDataStructure, String targetID) {

**course = seekCourse** **(myDataStructure, targetID) //**targetID = desiredKey

**if course is not null:**

**Print "Course ID: " + course.ID**

**Print "Course Name: " + course.Name**

**Print "Prerequisites:"**

**For each prereq in course.Prerequisites:**

**Print prereq**

**else:**

**Print "Course not found."**

}

**4.) Create pseudocode for a menu. The menu will need to perform the following actions:**

void main() {

**Declare filename as string**

**Declare myDataStructure as Vector<LinkedList<Course>>**

**Declare sortedCoursesArray as Array<Course>**

**While true:**

**Print Menu:**

**1 - Load Data from File**

**2 - Sort and Display Courses**

**3 - Search for Course by ID**

**9 - Exit**

**Set choice equal to user input**

**SWITCH choice:**

**Case 1:**

**Set myDataStructure equal to readFile( filename)**

**Case 2:**

**Call printDataStructure( myDataStructures root)**

**Case 3:**

**Prompt for TargetID**

**Call searchCourse( myDataStructure, TargetID)**

**Case 9:**

**Exit**

**Default:**

**Print "Invalid option"**

}

**5) Sort the course information by alphanumeric course number from lowest to highest.**

The sorted binary Tree doesn’t need to be directly sorted as this structure inertly orders them on insert. The sorted pattern only needs to be exhumed via the correct insert conditions and traversal pattern.

**6) Print the sorted list to a display.**

Void printDataStructure( course){  
 **IF course is null**

**Return**

**printDataStructure(courses left)**

**print course**

**printDataStructure(courses right)**  
}

## Runtime Analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **readFile:** *read and validate* | 2 | n | 2n |
| **courseFactory:**  *create, parse prereq* | 1+k | n | n+nk |
| **insertNode:** | logn | n | nlogn |
| **Total Cost** | | | n(3+k)+nlogn |
| **Runtime** | | | nlogn |

Unlike the linear structure, inserting into a BST has additional complexity due to the nature of its ordered placement. Each insertion operation requires traversing down the tree, causing a logarithmic cost per insertion. While initially more costly than simply appending elements to the vector, this upfront complexity provides substantial long-term benefits. Specifically, this complexity ensures that the data is inherently organized, enabling efficient lookups and sorted traversals. This built-in order reduces subsequent overhead when sorting or searching.