# CS 300 Pseudocode Document

#### Step 1: Opening and Reading the File

**Pseudocode:**

1. **Open File**
   * Function: OpenFile(filePath)
     + Input: filePath (string) - path to the file containing course data
     + Output: File object
   * Begin
     + Try to open the file at filePath
     + If file cannot be opened, print an error message and exit
     + Return the file object
   * End

#### Step 2: Reading Data from the File and Checking Format

**Pseudocode:**

1. **Read and Parse File**
   * Function: ReadAndParseFile(file)
     + Input: File object
     + Output: List of course objects (or binary search tree nodes)
   * Begin
     + Initialize an empty list courseList to hold course objects
     + Initialize a set courseSet to hold all course numbers (for validation)
     + For each line in the file:
       - Split the line by commas into a list tokens
       - If tokens length < 2, print "Invalid line format" and skip line
       - Extract courseNumber from tokens[0]
       - Extract courseTitle from tokens[1]
       - Extract prerequisites from tokens[2:] (if any)
       - Add courseNumber to courseSet
     + For each course in courseList:
       - For each prerequisite in prerequisites:
         * If prerequisite not in courseSet, print "Prerequisite not found" and skip course
       - Create a Course object with courseNumber, courseTitle, and prerequisites
       - Append Course object to courseList
     + Return courseList
   * End

#### Step 3: Creating Course Objects and Storing in Binary Search Tree

**Pseudocode:**

1. **Course Structure**
   * Define Course struct
     + Variables: courseNumber (string), courseTitle (string), prerequisites (list of strings)
   * End
2. **Binary Search Tree Node Structure**
   * Define TreeNode struct
     + Variables: course (Course), left (TreeNode pointer), right (TreeNode pointer)
   * End
3. **Binary Search Tree Structure**
   * Define BinarySearchTree class
     + Variables: root (TreeNode pointer)
     + Functions:
       - Insert(course)
       - InOrderTraversal(node)
   * End
4. **Insert Course into BST**
   * Function: InsertCourse(bst, course)
     + Input: Binary search tree bst, Course course
     + Output: None
   * Begin
     + If bst.root is NULL, set bst.root to new TreeNode(course)
     + Else, call InsertNode(bst.root, course)
   * End
5. **Insert Node into BST (Recursive)**
   * Function: InsertNode(node, course)
     + Input: TreeNode node, Course course
     + Output: None
   * Begin
     + If course.courseNumber < node.course.courseNumber
       - If node.left is NULL, set node.left to new TreeNode(course)
       - Else, call InsertNode(node.left, course)
     + Else
       - If node.right is NULL, set node.right to new TreeNode(course)
       - Else, call InsertNode(node.right, course)
   * End
6. **Load Data into BST**
   * Function: LoadData(filePath)
     + Input: filePath (string)
     + Output: Binary search tree bst
   * Begin
     + Call OpenFile(filePath) and assign to file
     + Call ReadAndParseFile(file) and assign to courseList
     + Initialize bst as a new BinarySearchTree
     + For each course in courseList, call InsertCourse(bst, course)
     + Return bst
   * End

#### Step 4: Printing Course Information and Prerequisites

**Pseudocode:**

1. **Print Course Information (In-Order Traversal)**
   * Function: PrintCourses(bst)
     + Input: Binary search tree bst
     + Output: None
   * Begin
     + Call InOrderTraversal(bst.root)
   * End
2. **In-Order Traversal**
   * Function: InOrderTraversal(node)
     + Input: TreeNode node
     + Output: None
   * Begin
     + If node is not NULL:
       - Call InOrderTraversal(node.left)
       - Print node.course.courseNumber, node.course.courseTitle
       - If node.course.prerequisites is not empty, print "Prerequisites: " and node.course.prerequisites
       - Call InOrderTraversal(node.right)
   * End

### Summary

This pseudocode outlines the steps to open a file, read and parse course data, validate the data, create course objects, insert them into a binary search tree, and finally print the course information in an in-order traversal. This structured approach ensures that each step is clear and logically organized, providing a solid foundation for implementing the code in a subsequent project.

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **for each prerequisite of the course** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |