**Project 1 Milestone**

By: Darrell Walker

**Define how the program opens the file, reads the data, parses each line, and checks for file format errors:**

FUNCTION readFileAndParseData(filename):

OPEN file in read mode

INITIALIZE empty list courseList

FOR each line in file:

SPLIT line by comma to get data

// Check if the line has at least two parameters

IF length of data < 2:

PRINT "Error: Line does not contain enough parameters"

CONTINUE

// Create a course object from the data

course = createCourseObject(data)

// Add the course object to the courseList

ADD course to courseList

CLOSE file

// Validate prerequisites for each course

validatePrerequisites(courseList)

// Insert courses into the binary search tree

courseTree = insertCoursesIntoBST(courseList)

RETURN courseTree

**Validate the sample file to make sure it is formatted correctly:**

FUNCTION validatePrerequisites(courseList):

INITIALIZE empty list courseNumbers

// Collect all course numbers

FOR each course in courseList:

ADD course.courseNumber to courseNumbers

// Check prerequisites

FOR each course in courseList:

FOR each prereq in course.prerequisites:

IF prereq NOT IN courseNumbers:

PRINT "Error: Prerequisite", prereq, "for course", course.courseNumber, "does not exist"

**Create course objects and store them in the appropriate data structure:**

// Function to create a course object from data

FUNCTION createCourseObject(data):

courseNumber = data[0]

courseName = data[1]

prerequisites = data[2 to end]

RETURN Course(courseNumber, courseName, prerequisites)

// Insert courses into a binary search tree

FUNCTION insertCoursesIntoBST(courseList):

INITIALIZE BinarySearchTree courseTree

FOR each course in courseList:

CALL courseTree.insert(course)

RETURN courseTree

**Define Course and BinarySearchTree classes:**

// Course class definition

CLASS Course:

FUNCTION \_\_init\_\_(courseNumber, courseName, prerequisites):

SET self.courseNumber = courseNumber

SET self.courseName = courseName

SET self.prerequisites = prerequisites

// BinarySearchTree class definition

CLASS BinarySearchTree:

FUNCTION \_\_init\_\_():

SET self.root = None

// Insert function to add a course to the tree

FUNCTION insert(course):

IF self.root is None:

SET self.root = TreeNode(course)

ELSE:

CALL self.\_insert(self.root, course)

FUNCTION \_insert(node, course):

IF course.courseNumber < node.course.courseNumber:

IF node.left is None:

SET node.left = TreeNode(course)

ELSE:

CALL self.\_insert(node.left, course)

ELSE:

IF node.right is None:

SET node.right = TreeNode(course)

ELSE:

CALL self.\_insert(node.right, course)

// TreeNode class definition

CLASS TreeNode:

FUNCTION \_\_init\_\_(course):

SET self.course = course

SET self.left = None

SET self.right = None

**Print course information and prerequisites:**

// Function to print course information and prerequisites

FUNCTION printCourseInformation(courseTree, courseNumber):

course = searchCourse(courseTree.root, courseNumber)

IF course is not None:

PRINT "Course Number:", course.courseNumber

PRINT "Course Name:", course.courseName

PRINT "Prerequisites:", JOIN course.prerequisites with ", "

ELSE:

PRINT "Course not found"

// Function to search for a course in the binary search tree

FUNCTION searchCourse(node, courseNumber):

IF node is None:

RETURN None

IF node.course.courseNumber == courseNumber:

RETURN node.course

ELSE IF courseNumber < node.course.courseNumber:

RETURN searchCourse(node.left, courseNumber)

ELSE:

RETURN searchCourse(node.right, courseNumber)