# CS 300 Project One: Pseudocode and Explanations

## 1. Pseudocode Overview

### Module 3 - System for Summarizing Student Data

### Objective: By reading student\_data.txt and producing a summary\_report.txt, assist ABCU advisers in tracking student success.

START  
OPEN "student\_data.txt" FOR READING  
CREATE empty list studentRecords  
  
WHILE NOT end of file  
 READ line from file  
 SPLIT line by commas into: studentID, firstName, lastName, courseCode, courseName, grade  
 CREATE studentRecord with those fields  
 ADD studentRecord to studentRecords list  
END WHILE  
  
CLOSE file  
  
OPEN "summary\_report.txt" FOR WRITING  
  
FOR EACH student IN studentRecords  
 DISPLAY student name and ID  
 COUNT number of courses  
 CHECK grades below C  
 IF grade is below C  
 ADD recommendation for advisor to follow up  
 END IF  
 WRITE summary line to "summary\_report.txt"  
END FOR  
  
CLOSE "summary\_report.txt"  
END

### Module 5 - Using Classes and Vectors in a Course Planner

### Structure: The application allows the user to print course details after reading a CSV file including courses and prerequisites.

File Pseudocode Input:

METHODS FileName: ReadCoursesFromFile

OPEN file

IF error THEN DISPLAY message and RETURN  
  
 WHILE file has lines  
 SPLIT line by comma  
 Continue if there aren't enough tokens.

SET courseId = token[0], courseTitle = token[1]  
 ADD to the prerequisites for the remaining tokens.

CREATE Course object and ADD to list  
 END WHILE  
  
 CLOSE file  
END PROCEDURE

Course Objectives and Functions:

CLASS Course:  
 courseId, courseTitle, prerequisites  
  
AddCourseToList (course) is the procedure.

ADD to list  
  
PROCEDURE SortCoursesById(courseList)  
 SORT list by courseId  
  
PROCEDURE FindCourseById(courseId)  
 FOR each course  
 IF match THEN RETURN course  
 RETURN null

Printing:

PROCEDURE PrintAllCourses()  
 SORT course list  
 FOR each course: DISPLAY courseId and title  
  
PROCEDURE PrintSingleCourse(courseId)  
 FIND course  
 IF not found THEN DISPLAY message  
 ELSE DISPLAY details and prerequisites

### Hash Table Project Pseudocode (Final Project)

Goal: Effectively manage bid data by using a hash table.

Start program  
  
DEFINE Bid struct:  
 bidId, title, fund, amount  
  
DEFINE class HashTable:

- vector<Node\*> table  
 - Hash function  
 - Node contains Bid + next pointer  
 - Add, Remove, Search, Print functions  
  
In main():  
 When the user hasn't chosen to exit  
 DISPLAY menu  
 IF option is Load:  
 PARSE CSV  
 Add bids to the hash table.

IF option is Show:  
 CALL PrintAll  
 IF option is Find:  
 INPUT bidId  
 CALL Search  
 IF option is Remove:  
 INPUT bidId  
 CALL Remove  
 END WHILE  
  
End program

## 2.An explanation and assessment of data structures

## Course Planner uses vectors:

## Pros: Simple, dynamically resizable, easy to sort. - Cons: Not optimized for fast search unless sorted. Hash tables, which are employed in bid management: - Pros: Fast O(1) average-case lookup, insert, and delete. - Cons: Potential for collisions (resolved via chaining), not sorted. Linked Lists (used for chaining in hash tables):

## - Pros: Good for dynamic memory usage, easy insertion/deletion. - Cons: Linear-time search.

## 3. Runtime Analysis

-O(n), where n is the number of lines in the file, is the parsing algorithm for CSV files.

-Adding or searching a hash table: O(1) average, O(n) worst-case (assuming every item ends up in the same bucket).

-Sorting Vectors: O(n log n).

-In the worst scenario, linked list traversal takes O(n) for every chain.

## 4. Reflection Summary

I showed that I understood file parsing, object-oriented design, and how to use data structures effectively in my project. Input validation and exception handling during string-to-double conversions are two ways I improved robustness. Effective management of dynamic bid data was facilitated by the creation of a hash table using linked list chaining, and the code's modular design allowed for maintenance and reuse. These methods will be useful in practical settings where structured data processing is essential.

## Runtime Analysis Table

The table below provides a sample runtime analysis for the `searchCourse()` function using the vector data structure.

|  |  |  |  |
| --- | --- | --- | --- |
| Code Line | Cost per Line | # of Executions | 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 |