* 1. **Project One**

**Assignment**

**By Justin Ramirez**

**Menu:**

FUNCTION main():

DECLARE data\_structure

WHILE True:

PRINT "Menu:"

PRINT "1. Load Data Structure"

PRINT "2. Print Course List"

PRINT "3. Print Course"

PRINT "4. Exit"

INPUT choice

SWITCH choice:

CASE 1:

data\_structure = load\_data\_structure()

PRINT "Data Structure Loaded"

CASE 2:

IF data\_structure is not initialized:

PRINT "Error: Data Structure not loaded. Please load data first."

ELSE:

PRINT\_course\_list(data\_structure)

CASE 3:

IF data\_structure is not initialized:

PRINT "Error: Data Structure not loaded. Please load data first."

ELSE:

PRINT "Enter the course code:"

INPUT course\_code

PRINT\_course(data\_structure, course\_code)

CASE 4:

EXIT

DEFAULT:

PRINT "Invalid choice. Please enter a valid option."

FUNCTION load\_data\_structure():

// Pseudocode to open the file, read data, parse each line, and create course objects

DECLARE data\_structure

OPEN file

READ each line in file:

PARSE line to extract course information

CREATE Course object from extracted information

ADD Course object to data\_structure

CLOSE file

RETURN data\_structure

FUNCTION PRINT\_course\_list(data\_structure):

// Pseudocode to print an alphanumerically ordered list of all the courses in the Computer Science department

SORT data\_structure by course code

FOR EACH course IN data\_structure:

PRINT course code and title

FUNCTION PRINT\_course(data\_structure, course\_code):

// Pseudocode to print the course title and prerequisites for the given course\_code

FOR EACH course IN data\_structure:

IF course code EQUALS course\_code:

PRINT course title

PRINT prerequisites

BREAK // Exit loop after finding the course

**Runtime analysis in a chart:**

|  |  |
| --- | --- |
| **Data Structure** | **Runtime Complexity** |
| Vector | O(n) |
| Hash Table | O(n) |
| Tree | O(n log n) |

**Evaluation:**

1. Vector:
   * Reading the file and creating course objects: O(n)
   * Advantage: Simple operation, easy to access elements by index.
   * Disadvantage: Adding and deleting may be slow as they require moving elements.
2. Hash Table:
   * Reading the file and creating course objects: O(n)
   * Advantage: Fast access to elements through hashing.
   * Disadvantage: May have collisions,that can lead to slower performance.
3. Tree:
   * Reading the file and creating course objects: O(n log n)
   * Advantage: Efficient searching and insertion.
   * Disadvantage: unsure.

**Recommendation:**

Considering these points and what the program needs, a Hash Table seems like the best choice. It's quick to find things, which is important for this program. It can handle a lot of items without slowing down too much.