**Hash Table Pseudocode**

**Main Flow:**

BEGIN

DISPLAY menu

WHILE user does not choose to exit

READ user choice

SWITCH on choice

CASE 1:

CALL loadCoursesIntoHashTable

CASE 2:

CALL displayAllCoursesInOrder

CASE 3:

PROMPT user for course ID

CALL findAndDisplayCourseInfo

CASE 9:

EXIT

END SWITCH

END WHILE

END

**Loading Courses into Hash Table:**

FUNCTION loadCoursesIntoHashTable(filePath, hashTable)

OPEN file at filePath

FOR each line in file

PARSE line into courseID, title, and prerequisites

CREATE new Course object with parsed data

CALCULATE hash key using courseID

INSERT Course object into hashTable using hash key

END FOR

CLOSE file

END FUNCTION

**Displaying All Courses:**

FUNCTION displayAllCoursesInOrder(hashTable)

CREATE a temporary vector to hold courses

FOR each bucket in hashTable

FOR each node in bucket

ADD course to temporary vector

CALL sortCoursesByAlphanumeric(temporary vector)

CALL displayAllCourses(temporary vector)

END FUNCTION

**Finding and Displaying Course Information:**

FUNCTION findAndDisplayCourseInfo(hashTable, courseID)

CALCULATE hash key using courseID

SEARCH for course in hashTable using hash key

IF course is found

PRINT course.title and course.prerequisites

ELSE

PRINT "Course not found."

END FUNCTION

**Runtime Analysis Chart**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Structure | Operation | Line Cost | # Times Executes | Total Cost | Runtime |
| Hash Table | Load Courses into Hash Table | 1 | n | n | O(1)(avg) / O(n) (worst) |
|  | Sort Courses | 1 | n\*log(n) | n\*log(n) | O(n\*log(n)) |
|  | Search for Course | 1 | 1 | 1 | O(1)(avg) / O(n) (worst) |

**Advantages and Disadvantages Analysis**

**Advantages:**

* Fast Access: Hash tables provide O(1) average-case access time for search, insertion, and deletion, which makes them very efficient for lookups.
* Flexible Size: Hash tables can dynamically grow as more elements are added, reducing the chance of collisions.
* No Sorting Required: Since hash tables don’t require sorting to access data quickly, they are efficient in scenarios with frequent lookups.

**Disadvantages:**

* Collisions: Hash collisions can occur, which might degrade performance to O(n) in the worst case.
* Complex Implementation: Hash tables are more complex to implement and require careful management of hash functions and collision handling strategies.
* Memory Usage: Hash tables often use more memory due to the requirement of storing additional information such as keys and pointers.