# CS 300 Pseudocode Document

****

The course data is as follows:

CSCI100,Introduction to Computer Science

CSCI101,Introduction to Programming in C++,CSCI100

CSCI200,Data Structures,CSCI101

MATH201,Discrete Mathematics

CSCI300,Introduction to Algorithms,CSCI200,MATH201

CSCI301,Advanced Programming in C++,CSCI101

CSCI350,Operating Systems,CSCI300

CSCI400,Large Software Development,CSCI301,CSCI350

**1.) Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors**

//readFile - Milestone 2

Vector<LinkedList<Course>> readFile(String filename, int tableSize) {

**Create courseVec as Vector of size tableSize, initialized with empty LinkedLists heads**

**Declare readVal as String**

**try {**

**Open file using filename**

**While file has more lines:**

**Set readVal = readline from file**

**If readVal contains fewer than 1 commas:**

**Print warning: "Malformed line: " + readVal**

**Continue**

**course = CourseFactory(readVal)**

**key = hashID(course.ID, tableSize) // page below**

**If courseVec[key] is empty:**

**Add course to courseVec[key]**

**Else:**

**Prepend course to courseVec[key] // for O(1) insert**

**}**

**catch FileNotFoundException:**

**Print "Error: Could not open file."**

**finally:**

**Close file**

**return courseVec**

}

Define function hashID(string id, int tableSize){

**hash = 0**

**For each character c in id:**

**hash = (hash \* 31 + ASCII(c)) mod tableSize**

**Return hash**

**}**

**2.) Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

//CourseFactory - Milestone 2

Course CourseFactory(String readVal) {

**Create new Course:**

**ID = ""**

**Name = ""**

**Prerequisites = empty vector**

**Next = null**

**Split readVal by ',' into tokens**

**If tokens.length < 2:**

**Throw error or return null**

**Set Course.ID = tokens[0]**

**Set Course.Name = tokens[1]**

**For i = 2 to tokens.length - 1:**

**Append tokens[i] to Course.Prerequisites**

**return Course**

}

**3.) Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites.**

//Vector - Milestone 2

void searchCourse(Vector<LinkedList<Course>> table, String targetID) {

**key = hash(targetID) mod table.length**

**linkedList = table[key]**

**For each course in linkedList:**

**If course.ID equals targetID:**

**Print course.ID**

**Print course.Name**

**Print "Prerequisites:"**

**For each prereq in course.Prerequisites:**

**Print prereq**

**return**

**Print "Course not found."**

}

**4.) Create pseudocode for a menu. The menu will need to perform the following actions:**

void main() {

**Declare filename as string**

**Declare tableSize as int**

**Declare myDataStructure as Vector<LinkedList<Course>>**

**Declare sortedCoursesArray as Array<Course>**

**While true:**

**Print Menu:**

**1 - Load Data from File**

**2 - Sort and Display Courses**

**3 - Search for Course by ID**

**9 - Exit**

**Set choice equal to user input**

**SWITCH choice:**

**Case 1:**

**Set myDataStructure equal to readFile(filename, tableSize)**

**Case 2:**

**Set sortedCoursesArray equal to sortDataStructure(myDataStructure)**

**Call printDataStructure(sortedCoursesArray)**

**Case 3:**

**Prompt for TargetID**

**Call searchCourse(myDataStructure, TargetID)**

**Case 9:**

**Exit**

**Default:**

**Print "Invalid option"**

}

**5) Sort the course information by alphanumeric course number from lowest to highest.**

// The reason Im using an array is because a hash map wasn’t designed for this.

// they are designed for fast look up and retrieval, not sorting data.

Array<course> sortDataStructure(Vector<LinkedList<Course>> courseVec){

**Declare courses as empty Array<Course>**

**For each bucket in courseVec:**

**For each course in bucket:**

**Append course to courses array**

**Sort courses array lexicographically based on Course.ID (merge sort)**

**return courses**

}

**6) Print the sorted list to a display.**

void printDataStructure(sortedCoursesArray){  
 **for all courses**

**print out the course information**

**for each prerequisite of the course**

**print the prerequisite course information**

}

## Runtime Analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **readFile:** *initialize* | O(tableSize) | 1 | O(tableSize) |
| **readFile:** *read and validate* | 2 | n | 2n |
| **courseFactory:**  *create, parse prereq* | 1+k | n | n+nk |
| **insert:** | O(m) | n | ~n (m is small) |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | n(3+k) |
| **Runtime** | | | O(n) |

Hash tables offer extremely fast insertions and lookups on average due to the hash function efficiently mapping keys to indices. Each insertion is quick, involving a simple calculation and insertion at the bucket head. However, a significant limitation arises because hash tables do not inherently maintain sorted order. To display sorted data, an external sorting operation (like merge sort) must occur, increasing complexity and memory use.