# CS 300 Pseudocode Document

**Course Object Pseudocode**string course number  
string course name  
vector<string> prereqs

**File Reading Pseudocode**Main {  
 String tempString  
 String fileName  
 Course tempCourse  
 vector<course> courses  
 Get fileName  
 open ofstream(fileName)  
 tempString = getline from file  
 while (end of file isn’t reached) {  
 numCommas = number of commas in tempString  
 be sure number of commas is greater than 0 (means at least 2 parameters in the line)  
 for (i = 0; i < numCommas; i++) {  
 find next comma  
 assign string up until next comma into next variable for tempCourse(first course number, then course name, then fill vector prereqs)   
 if (tempCourse has more than 0 prerequsites) {  
 check to see if the course number exists in the file  
 throw error if it doesn’t  
 }  
 }  
 add tempCourse to the Course vector  
 tempString = getline from file  
 }  
}

## Function Signatures

Below are the function signatures that you can fill in to address each of the three program requirements using each of the data structures. The pseudocode for printing course information, if a vector is the data structure, is also given to you below (depicted in bold).

// Vector pseudocode

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

void printSampleSchedule(Vector<Course> courses) {

}

void printCourseInformation(Vector<Course> courses, String courseNumber) {

**for all courses**

**if the course is the same as courseNumber**

**print out the course information**

**for each prerequisite of the course**

**print the prerequisite course information**

}

**Course Object Pseudocode**

Use course object as outlined in the previous section

**File Reading Pseudocode**

Main {  
 String tempString  
 String fileName  
 Course tempCourse  
 HashTable<Course> courses

Initialize the HashTable with empty Courses  
 Get fileName  
 open ofstream(fileName)  
 tempString = getline from file  
 while (end of file isn’t reached) {  
 numCommas = number of commas in tempString  
 be sure number of commas is greater than 0 (means at least 2 parameters in the line)  
 for (i = 0; i < numCommas; i++) {  
 find next comma  
 assign string up until next comma into next variable for tempCourse(first course number, then course name, then fill vector prereqs)   
 if (tempCourse has more than 0 prerequsites) {  
 check to see if the course number exists in the file by checking for the prerequisite ID at the index that would be generated from a key made with the prereq’s course ID (or any following non-empty indexes)  
 throw error if it doesn’t  
 }  
 }

Generate key based on course ID and size of HashTable  
 add tempCourse to the Course HashTable at the index of the generated key or next empty one

If HashTable gets too full, resize to be bigger  
 tempString = getline from file  
 }  
}

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites if said prerequisites aren’t already in totalPrerequisites

print number of totalPrerequisites

}

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

Assign Key with the Key generated from the course number and size of the HashTable

Search the HashTable for matching courseNumber

if the course is the same as courseNumber

print out the course information

for each prerequisite of the course

print the prerequisite course information

}

**Course Object Pseudocode**

Use course object as outlined in the previous section

**Node Object Pseudocode**

Course course

Node\* Left = nullptr

Node\* Right = nullptr

**Tree Object Pseudocode**

Root = nullptr

**File Reading Pseudocode**

Main {  
 String tempString  
 String fileName  
 Course tempCourse  
 Tree courses

Get fileName  
 open ofstream(fileName)  
 tempString = getline from file  
 while (end of file isn’t reached) {  
 numCommas = number of commas in tempString  
 be sure number of commas is greater than 0 (means at least 2 parameters in the line)  
 for (i = 0; i < numCommas; i++) {  
 find next comma  
 assign string up until next comma into next variable for tempCourse(first course number, then course name, then fill vector prereqs)   
 if (tempCourse has more than 0 prerequsites) {  
 check to see if the course number exists in the file by checking for the prerequisite ID at the index that would be generated from a key made with the prereq’s course ID (or any following non-empty indexes)  
 throw error if it doesn’t  
 }  
 }

Insert tempCourse into courses based on course ID  
 tempString = getline from file  
 }  
}

// Tree pseudocode

int numPrerequisiteCourses(Tree courses) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites if said prerequisites aren’t already in totalPrerequisites

print number of totalPrerequisites

}

void printCourseInformation(Tree courses, String courseNumber) {

make a current pointer and assign it root

while (current is not null)

if (courseNumber == current’s courseID)

print current pointer’s course

return

else if (courseNumber < current’s courseID)

current = current’s left pointer

else //courseNumber > current’s coursed

current = current’s right pointer

}

**PrintMenu() Pseudocode**

Print options 1-4

Option 1 is to load the data from the file

Option 2 is to print the course list

Option 3 is to print the course title and prerequisites for a specified course

Option 4 is to exit the program

**Menu Pseudocode**

Boolean KeepRunning = true

While (KeepRunning)

Call PrintMenu() Function

Get input (1-4) from user

Switch(input)

Case 1:

Load Data Structure (using pseudocode above)

break

Case 2:

Print Course List (outlined below)

Break

Case 3:

Get CourseID from user

PrintCourseInformation(CourseID) (outlined above)

Break

Case 4:

Sets KeepRunning to false

**Print Courses in order Pseudocode**

For Vectors and Hash Table: Sort code

Merge Sort based on Course ID, compare 1 character at a time, first 4 characters in each course ID are Capital letters, and the last 3 characters are digits so no need to split the string to compare.

For example:

While (merging partitions and a comparison needs to be made)

For (int i = 0; i < 7; i++)

If (string1[i] < string2[i])

Add string1 to merged partition

Else if (string1[i] > string2[i])

Add string2 to merged partition

No need to sort a Binary Tree as the insert already sorts it by Course ID

Once the structure is sorted call the respective structure print method which calls the printCourseInformation() method for every item.

**Pros and Cons**

Vector –

Pros:

Dynamically sized

The data loading process is the quickest out of the 3.

Cons:

Searching is slow until the vector is sorted

Hash Table –

Pros:

Quick searches

Cons:

Any time a Course needs to be inserted or searched for, a hash key needs to be calculated, adding an extra step to everything.

Instead of having dynamic size, the hash table needs to be manually resized and copied when resized

Binary Tree -

Pros:

Dynamically sized

Automatically Alphanumerically sorted by CourseID from structure being loaded

Cons:

Longer time to fully load the binary tree structure as it inserts due to comparisons

**Recommendation:**

I recommend using a binary tree as it has the fastest overall use time, and is presorted, which helps with the process for printing the list.