# CS 300 Pseudocode Document

## Example Function Signatures

Below is an example of a function signature that you can use as a guide to help address the program requirements using each data structure for the milestones. The pseudocode for finding and printing course information is also given below and depicted in bold to help you get started. The provided pseudocode is for a vector data structure, so you may use this pseudocode in your first milestone as is. The hash table and tree structures are also shown below. But these structures are left for you to do in future milestones.

//Vector - Milestone 1

Parser(Vector<Course> courses, String courseNumber){

Check for file type

Start stream

If File open

While file working

Getline(CourseNumber, CourseName, Prerequisite)

If line finished

Pushback line

Close stream

Else

Throw error failed to open

Parseheader (for class name)

Parsecontent(for prereqs and other info)

void searchCourse(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**

**}**

**Set default node**

**Key = UINT\_MAX**

**Next = null**

Void DisplayCourses (Vector<Course> courses, String courseNumber)

Print first value(course number, second value(course name), third value(prerequisites)

Loop through values printing each of them separating them with a “,”

//Hash Table - Milestone 2

Parser(Vector<Course> courses, String courseNumber){

Check for file type

Start stream

If File open

While file working

Getline(CourseNumber, CourseName, Prerequisite)

If line finished

Pushback line

Close stream

Else

Throw error failed to open

Parseheader (for class name)

Parsecontent(for prereqs and other info)

}

void searchCourse(HashTable<Course> courses, String courseNumber) {

**create key for courseNumber**

**Create node at key**

**If node not null and node key not equal to UINT\_max and node not 0**

**Return courseNumber**

}

{

While node not equal to null{

If node key not equal to UINT\_MAX and node not equal to 0

Return node courseNumber

}

Node = next node

}

Return courseNumber

}

{

//Binary Search Tree – Milestone 3

Parser(Vector<Course> courses, String courseNumber){

Check for file type

Start stream

If File open

While file working

Getline(CourseNumber, CourseName, Prerequisite)

If line finished

Pushback line

Close stream

Else

Throw error failed to open

Parseheader (for class name)

Parsecontent(for prereqs and other info)

}

void searchCourse(Tree<Course> courses, String courseNumber) {

Set current node equal to root

While current node not null

If current node equal to 0

Return current node

}

Else if current node less than 0{

Current node becomes left node

}

Else{

Current node becomes right node

}

}

Node node

Return node

}

MENU

Set choice to 0

While choice not equal to 0

Print menu choices 1. Load file data, 2. Print ordered list of courses, 3 print course title and prereqs, 9. Exit

Switch choice

Case 1 load course data(courses,csvPath)

Case 2 print ordered list

Case 3 print course info (courseNumber)

Case 9:exit program

Default: incorrect answer please put in 1,2,3, or 9

## Example Runtime Analysis

When you are ready to analyze the runtime for the Project One data structures for which you created the pseudocode, use the example chart below to support your work. This particular example is for printing course information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document. The example only covers the search function for the vector structure. You do not have to complete your runtime analysis until Project One. However, working on your analysis now may help you understand the changes as you complete the milestones. Don’t forget to include your charts in Project One. You will submit Project One in Module Six.

| **Code** | **Line Cost** | **# Times Executes** | **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 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | Time executed | cost |
| Parser(Vector<Course> courses, String courseNumber) | 1 | n | n |
| Check for file type | 1 | n | n |
| Start stream | 1 | 1 | 1 |
| If File open | 1 | n | n |
| While file working | 1 | n | n |
| Getline(CourseNumber, CourseName, Prerequisite) | 1 | n | n |
| If line finished | 1 | n | n |
| Pushback line | 1 | n | n |
| Close stream | 1 | n | n |
| Else Throw error failed to open | 1 | 1 | 1 |
| Parseheader (for class name) | 1 | n | n |
| Parsecontent(for prereqs and other info) | 1 | n | n |
| Total cost 13n+2 | | | |
| Runtime 0(n) | | | |
| Code Line Cost Time executed cost | | | |
| **create key for courseNumber** | 1 | n | 1 |
| **Create node at key** | 1 | n | 1 |
| **If node not null and node key not equal to UINT\_max and node not 0** | 1 | n | n |
| **Return courseNumber** | 1 | n | n |
| While node not equal to null{ | 1 | n | n |
| If node key not equal to UINT\_MAX and node not equal to 0 | 1 | n | n |
| Return node courseNumber | 1 | n | n |
| Node = next node | 1 | 1 | 1 |
| Return courseNumber | 1 | n | n |
| Total Cost 9n(3) | | | |
| Runtime 0(n) | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | Time Executed | cost |
| Set current node equal to root | 1 | 1 | 1 |
| While current node not null | 1 | n | n |
| If current node equal to 0 | 1 | n | n |
| Return current node | 1 | n | n |
| Else if current node less than 0{ | 1 | n | n |
| Current node becomes left node | 1 | 1 | 1 |
| Else{  Current node becomes right node | 1 | 1 | 1 |
| Node node | 1 | n | n |
| Return node | 1 | n | n |
| Total Cost 9n(3) | | | |
| Runtime 0(n) | | | |

Advantages and disadvantages

Binary search trees allow you to traverse the list of items in your file and sort them. This is not the case for something like a hash table. which are great as they will only compare if the key they have matches the data. This would make them the fastest to use typically. Though vectors are better if you would like to add new data as it is simply added to the end of the list that is parsed. When using vectors, you need to make a function to call so that the data can be printed in what order you want it to be printed. Overall, hash tables are the fastest and would be the type I would like to use for project 2. Them being faster makes them more efficient and thus better to use for this assignment.