# CS 300 Pseudocode Document

**Vector — Milestone 1**

TYPE Course:

number: string

title: string

prereqs: list<string> // raw course numbers, uppercased, trimmed

FUNCTION NORMALIZE(text: string) -> string

return UPPER(TRIM(text))

END FUNCTION

FUNCTION PARSE\_LINE\_TO\_COURSE(line: string) -> Course

// Expected CSV: number,title,prereq1|prereq2|...

// No external CSV library, split by comma carefully, then split prereqs by '|'

fields <- SPLIT\_CSV(line) // write your own minimal splitter

c <- new Course

c.number <- NORMALIZE(fields[0])

c.title <- TRIM(fields[1])

if LENGTH(fields) >= 3 AND fields[2] != "" then

raw <- SPLIT(fields[2], "|")

for each p in raw do

c.prereqs.APPEND(NORMALIZE(p))

end for

end if

return c

END FUNCTION

FUNCTION LOAD\_COURSES\_FROM\_FILE\_VECTOR(path: string) -> vector<Course>

vector<Course> courses

file <- OPEN(path)

if file == null then

print "Could not open file."

return courses

end if

// First pass, read all rows, normalize, store

while HAS\_NEXT\_LINE(file) do

line <- READ\_LINE(file)

if line == "" then continue

c <- PARSE\_LINE\_TO\_COURSE(line)

if c.number != "" then

courses.PUSH\_BACK(c)

end if

end while

CLOSE(file)

// Second pass, validate prereqs exist in catalog

// This is a simple O(n^2) approach, acceptable for Milestone 1

for i from 0 to courses.size-1 do

validList <- empty list<string>

for each p in courses[i].prereqs do

if EXISTS\_COURSE\_VECTOR(courses, p) then

validList.APPEND(p)

else

// keep it, but it will print as missing definition

validList.APPEND(p)

end if

end for

courses[i].prereqs <- validList

end for

return courses

END FUNCTION

FUNCTION EXISTS\_COURSE\_VECTOR(courses: vector<Course>, courseNumber: string) -> bool

target <- NORMALIZE(courseNumber)

for each c in courses do

if NORMALIZE(c.number) == target then

return true

end if

end for

return false

END FUNCTION

FUNCTION SEARCH\_COURSE\_VECTOR(courses: vector<Course>, courseNumber: string) -> Course or null

target <- NORMALIZE(courseNumber)

for each c in courses do

if NORMALIZE(c.number) == target then

return c

end if

end for

return null

END FUNCTION

FUNCTION PRINT\_COURSE\_WITH\_PREREQS\_VECTOR(courses: vector<Course>, courseNumber: string)

course <- SEARCH\_COURSE\_VECTOR(courses, courseNumber)

if course == null then

print "Course not found."

return

end if

print course.number + ", " + course.title

if course.prereqs is empty then

print "Prerequisites: none"

else

print "Prerequisites:"

for each p in course.prereqs do

pre <- SEARCH\_COURSE\_VECTOR(courses, p)

if pre != null then

print " " + pre.number + " - " + pre.title

else

print " " + p + " (missing definition)"

end if

end for

end if

END FUNCTION

FUNCTION PRINT\_ALL\_COURSES\_VECTOR(courses: vector<Course>)

list <- COPY\_VECTOR(courses)

SORT\_BY(list, c.number ascending, alphanumeric)

for each c in list do

print c.number + ", " + c.title

end for

END FUNCTION

FUNCTION PROMPT\_FILE\_PATH() -> string

path <- PROMPT("Enter file path, or press Enter for default:")

if path == "" then

path <- "courses.csv"

end if

return path

END FUNCTION

FUNCTION MENU\_VECTOR()

loaded <- false

vector<Course> courses

REPEAT

print ""

print "1. Load file"

print "2. Print all courses [A to Z]"

print "3. Print one course [title + prerequisites]"

print "9. Exit"

choice <- READ\_INT()

SWITCH choice

CASE 1:

path <- PROMPT\_FILE\_PATH()

courses <- LOAD\_COURSES\_FROM\_FILE\_VECTOR(path)

if courses.size > 0 then

loaded <- true

print "Loaded " + TO\_STRING(courses.size) + " courses."

else

loaded <- false

end if

CASE 2:

if not loaded then

print "Please load the data first."

else

PRINT\_ALL\_COURSES\_VECTOR(courses)

end if

CASE 3:

if not loaded then

print "Please load the data first."

else

num <- PROMPT("Enter course number:")

PRINT\_COURSE\_WITH\_PREREQS\_VECTOR(courses, num)

end if

CASE 9:

print "Good bye."

DEFAULT:

print "Invalid choice."

END SWITCH

UNTIL choice == 9

END FUNCTION

**Hash Table — Milestone 2**

*TYPE Course:*

*number: string*

*title: string*

*prereqs: list<string> // uppercased, trimmed numbers*

*FUNCTION NORMALIZE(text: string) -> string*

*return UPPER(TRIM(text))*

*END FUNCTION*

*FUNCTION PARSE\_LINE\_TO\_COURSE(line: string) -> Course*

*// CSV format: number,title,prereq1|prereq2|...*

*fields <- SPLIT\_CSV(line) // minimal custom splitter*

*c <- new Course*

*c.number <- NORMALIZE(fields[0])*

*c.title <- TRIM(fields[1])*

*if LENGTH(fields) >= 3 AND fields[2] != "" then*

*raw <- SPLIT(fields[2], "|")*

*for each p in raw do*

*c.prereqs.APPEND(NORMALIZE(p))*

*end for*

*end if*

*return c*

*END FUNCTION*

*// HashTable API assumed:*

*// put(key, value), get(key) -> value or null, contains(key) -> bool, keys() -> iterable<string>*

*FUNCTION LOAD\_COURSES\_FROM\_FILE\_HASH(path: string) -> HashTable<string, Course>*

*table <- new HashTable<string, Course>()*

*file <- OPEN(path)*

*if file == null then*

*print "Could not open file."*

*return table*

*end if*

*// First pass, parse and insert by normalized key*

*while HAS\_NEXT\_LINE(file) do*

*line <- READ\_LINE(file)*

*if line == "" then continue*

*c <- PARSE\_LINE\_TO\_COURSE(line)*

*if c.number != "" then*

*// Collisions are handled internally by the table, for example via chaining*

*table.put(c.number, c)*

*end if*

*end while*

*CLOSE(file)*

*// Second pass, optional validation, do not modify storage here*

*// We simply rely on table.get during printing to resolve prereqs*

*return table*

*END FUNCTION*

*FUNCTION SEARCH\_COURSE\_HASH(table: HashTable<string, Course>, courseNumber: string) -> Course or null*

*key <- NORMALIZE(courseNumber)*

*return table.get(key)*

*END FUNCTION*

*FUNCTION PRINT\_COURSE\_WITH\_PREREQS\_HASH(table: HashTable<string, Course>, courseNumber: string)*

*course <- SEARCH\_COURSE\_HASH(table, courseNumber)*

*if course == null then*

*print "Course not found."*

*return*

*end if*

*print course.number + ", " + course.title*

*if course.prereqs is empty then*

*print "Prerequisites: none"*

*else*

*print "Prerequisites:"*

*for each p in course.prereqs do*

*pre <- table.get(p) // p already normalized*

*if pre != null then*

*print " " + pre.number + " - " + pre.title"*

*else*

*print " " + p + " (missing definition)"*

*end if*

*end for*

*end if*

*END FUNCTION*

*FUNCTION PRINT\_ALL\_COURSES\_HASH(table: HashTable<string, Course>)*

*list <- empty vector<Course>*

*for each k in table.keys() do*

*c <- table.get(k)*

*if c != null then*

*list.PUSH\_BACK(c)*

*end if*

*end for*

*SORT\_BY(list, c.number ascending, alphanumeric)*

*for each c in list do*

*print c.number + ", " + c.title*

*end for*

*END FUNCTION*

*FUNCTION PROMPT\_FILE\_PATH() -> string*

*path <- PROMPT("Enter file path, or press Enter for default:")*

*if path == "" then*

*path <- "courses.csv"*

*end if*

*return path*

*END FUNCTION*

*FUNCTION MENU\_HASH()*

*loaded <- false*

*table <- null*

*REPEAT*

*print ""*

*print "1. Load file"*

*print "2. Print all courses [A to Z]"*

*print "3. Print one course [title + prerequisites]"*

*print "9. Exit"*

*choice <- READ\_INT()*

*SWITCH choice*

*CASE 1:*

*path <- PROMPT\_FILE\_PATH()*

*table <- LOAD\_COURSES\_FROM\_FILE\_HASH(path)*

*// crude count*

*count <- 0*

*for each \_ in table.keys() do count <- count + 1 end for*

*loaded <- (count > 0)*

*if loaded then*

*print "Loaded " + TO\_STRING(count) + " courses."*

*else*

*print "No courses loaded."*

*end if*

*CASE 2:*

*if not loaded then*

*print "Please load the data first."*

*else*

*PRINT\_ALL\_COURSES\_HASH(table)*

*end if*

*CASE 3:*

*if not loaded then*

*print "Please load the data first."*

*else*

*num <- PROMPT("Enter course number:")*

*PRINT\_COURSE\_WITH\_PREREQS\_HASH(table, num)*

*end if*

*CASE 9:*

*print "Good bye."*

*DEFAULT:*

*print "Invalid choice."*

*END SWITCH*

*UNTIL choice == 9*

*END FUNCTION*

**Binary Search Tree — Milestone 3**

*TYPE Course:*

*number: string*

*title: string*

*prereqs: list<string> // uppercased, trimmed*

*// Minimal BST keyed by course number, string compare*

*TYPE BSTNode:*

*key: string*

*data: Course*

*left: BSTNode*

*right: BSTNode*

*TYPE BST:*

*root: BSTNode*

*FUNCTION NORMALIZE(text: string) -> string*

*return UPPER(TRIM(text))*

*END FUNCTION*

*FUNCTION PARSE\_LINE\_TO\_COURSE(line: string) -> Course*

*// CSV format: number,title,prereq1|prereq2|...*

*fields <- SPLIT\_CSV(line) // write a simple splitter*

*c <- new Course*

*c.number <- NORMALIZE(fields[0])*

*c.title <- TRIM(fields[1])*

*if LENGTH(fields) >= 3 AND fields[2] != "" then*

*raw <- SPLIT(fields[2], "|")*

*for each p in raw do*

*c.prereqs.APPEND(NORMALIZE(p))*

*end for*

*end if*

*return c*

*END FUNCTION*

*FUNCTION BST\_FIND(t: BST, key: string) -> BSTNode or null*

*k <- NORMALIZE(key)*

*cur <- t.root*

*while cur != null do*

*if k == cur.key then return cur*

*if k < cur.key then cur <- cur.left else cur <- cur.right*

*end while*

*return null*

*END FUNCTION*

*FUNCTION BST\_INSERT(t: BST, course: Course)*

*node <- new BSTNode*

*node.key <- course.number*

*node.data <- course*

*node.left <- null*

*node.right <- null*

*if t.root == null then*

*t.root <- node*

*return*

*end if*

*cur <- t.root*

*prev <- null*

*while cur != null do*

*prev <- cur*

*if node.key < cur.key then cur <- cur.left else cur <- cur.right*

*end while*

*if node.key < prev.key then prev.left <- node else prev.right <- node*

*END FUNCTION*

*FUNCTION LOAD\_COURSES\_FROM\_FILE\_BST(path: string) -> BST*

*tree <- new BST*

*tree.root <- null*

*file <- OPEN(path)*

*if file == null then*

*print "Could not open file."*

*return tree*

*end if*

*while HAS\_NEXT\_LINE(file) do*

*line <- READ\_LINE(file)*

*if line == "" then continue*

*c <- PARSE\_LINE\_TO\_COURSE(line)*

*if c.number != "" then*

*BST\_INSERT(tree, c)*

*end if*

*end while*

*CLOSE(file)*

*return tree*

*END FUNCTION*

*FUNCTION SEARCH\_COURSE\_BST(tree: BST, courseNumber: string) -> Course or null*

*node <- BST\_FIND(tree, courseNumber)*

*if node == null then return null else return node.data*

*END FUNCTION*

*FUNCTION PRINT\_COURSE\_WITH\_PREREQS\_BST(tree: BST, courseNumber: string)*

*course <- SEARCH\_COURSE\_BST(tree, courseNumber)*

*if course == null then*

*print "Course not found."*

*return*

*end if*

*print course.number + ", " + course.title*

*if course.prereqs is empty then*

*print "Prerequisites: none"*

*else*

*print "Prerequisites:"*

*for each p in course.prereqs do*

*node <- BST\_FIND(tree, p) // p already normalized*

*if node != null then*

*print " " + node.data.number + " - " + node.data.title*

*else*

*print " " + p + " (missing definition)"*

*end if*

*end for*

*end if*

*END FUNCTION*

*FUNCTION PRINT\_ALL\_COURSES\_BST(tree: BST)*

*PROCEDURE IN\_ORDER(n: BSTNode)*

*if n == null then return*

*IN\_ORDER(n.left)*

*print n.data.number + ", " + n.data.title*

*IN\_ORDER(n.right)*

*END PROCEDURE*

*IN\_ORDER(tree.root)*

*END FUNCTION*

*FUNCTION PROMPT\_FILE\_PATH() -> string*

*path <- PROMPT("Enter file path, or press Enter for default:")*

*if path == "" then path <- "courses.csv"*

*return path*

*END FUNCTION*

*FUNCTION MENU\_BST()*

*loaded <- false*

*tree <- null*

*REPEAT*

*print ""*

*print "1. Load file"*

*print "2. Print all courses [A to Z]"*

*print "3. Print one course [title + prerequisites]"*

*print "9. Exit"*

*choice <- READ\_INT()*

*SWITCH choice*

*CASE 1:*

*path <- PROMPT\_FILE\_PATH()*

*tree <- LOAD\_COURSES\_FROM\_FILE\_BST(path)*

*// count nodes with an in-order pass*

*count <- 0*

*PROCEDURE COUNT(n)*

*if n == null then return*

*COUNT(n.left)*

*count <- count + 1*

*COUNT(n.right)*

*END PROCEDURE*

*COUNT(tree.root)*

*loaded <- (count > 0)*

*if loaded then*

*print "Loaded " + TO\_STRING(count) + " courses."*

*else*

*print "No courses loaded."*

*end if*

*CASE 2:*

*if not loaded then*

*print "Please load the data first."*

*else*

*PRINT\_ALL\_COURSES\_BST(tree)*

*end if*

*CASE 3:*

*if not loaded then*

*print "Please load the data first."*

*else*

*num <- PROMPT("Enter course number:")*

*PRINT\_COURSE\_WITH\_PREREQS\_BST(tree, num)*

*end if*

*CASE 9:*

*print "Good bye."*

*DEFAULT:*

*print "Invalid choice."*

*END SWITCH*

*UNTIL choice == 9*

*END FUNCTION*

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses (linear search)** | 1 | N | N |
| **if course == target** | 1 | 1 | 1 |
| **print "for each prerequisite of the course"** | 1 | 1 | 1 |
| **for each of k prerequisites** | 1 | K | K |
| **print prerequisite info** | 1 | K | k |
| **Total Cost** | | | n + 2k + 2 |
| **Runtime (Big-O)** | | | O(n + k) |

I recommend implementing a **Binary Search Tree (BST)** because it offers fast average-case search, insertion, and deletion in **O(log n)** time and allows the entire catalog to be printed in sorted order using a single in-order traversal (**O(n)**). These properties directly meet the advisors’ two main requirements: quickly locating specific courses throughout the day and efficiently generating an ordered course list. A BST also maintains sorted order without requiring additional sorting steps, unlike a vector or hash table, making it the most efficient and scalable option for this application.

**VECTOR — Load & Build**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **open file** | 1 | 1 | 1 |
| **for each input line (n):** |  |  |  |
| **parse/split** | 1 | n | n |
| **create Course object** | 1 | N | n |
| **push\_back into vector (amortized O(1))** | 1 | n | n |
| **Total Cost** | | | 3n + 1 |
| **Runtime (Big-O)** | | | O(n) |
| **Memory:** | | | O(n) |

**BST — Load & Build**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Code** | **Line Cost** | |  | | --- | |  |  |  | | --- | | **# Times Executes** | | Total Cost |
| open file | **1** | **1** | **1** |
| for each input line (n): |  |  |  |
| parse/split | **1** | **N** | **N** |
| create Course object | **1** | **N** | **N** |
| insert into BST (avg O(log n)) | **log n** | **N** | |  | | --- | |  |  |  | | --- | | **n log n** | |
| (worst case: O(n)) | | | **n²** |
| Total Cost (avg) | | | n log n + 2n + 1 |
| Runtime (Big-O) | | | **O(n log n)** average, **O(n²)** worst |
| Memory | | | O(n) |

**BST — Search & Print One Course**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | Line Cost | # Times Executes | Total Cost |
| find course in BST | log n | |  | | --- | |  |  |  | | --- | | 1 | | log n |
| if course == target | |  | | --- | |  |  |  | | --- | | 1 | | |  | | --- | |  |  |  | | --- | | 1 | | |  | | --- | |  |  |  | | --- | | 1 | |
| print "for each prerequisite" | |  | | --- | |  |  |  | | --- | | 1 | | |  | | --- | |  |  |  | | --- | | 1 | | |  | | --- | |  |  |  | | --- | | 1 | |
| for each of k prerequisites | |  | | --- | |  |  |  | | --- | | 1 | | k | k |
| find prerequisite in BST | log n | k | log n |
| print prerequisite info | 1 | k | k |
| Total Cost (avg) | | | log n + k log n + 2k + 2 |
| Runtime (Big-O) | | | |  | | --- | |  |  |  | | --- | | **O(log n + k log n)** → simplifies to **O(k log n)** | |
| Memory | | | **O(1)** (no new structures created) |

**Hash Table — Load & Build**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | | | Total Cost |
| for each input line, n | | | | | |
| parse or split | |  | | --- | |  |  |  | | --- | | 1 | | | |  | | --- | |  |  |  | | --- | | 1 | | |  | | --- | |  |  |  | | --- | | 1 | | |
| create Course object | |  | | --- | |  |  |  | | --- | | 1 | | | n | n | |
| put into hash table, avg O(1) | |  | | --- | |  |  |  | | --- | | 1 | | | n | n | |
| worst case, heavy collisions | | | | n² | |
| Total Cost, avg | | | | 3n + 1 | |
| Runtime, Big O | | | | O(n) average, O(n²) worst | |
| Memory | | | | O(n) | |
|  | | | | | |

Hash Table — Search and Print One Course

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| get course by key | 1 | 1 | 1 |
| if not found, print message | 1 | 1 | 1 |
| print header line | 1 | 1 | 1 |
| print “for each prerequisite” | 1 | 1 | 1 |
| for each of k prerequisites | 1 | k | k |
| get prerequisite by key, avg O(1) | 1 | k | k |
| print prerequisite info | 1 | k | k |
| Total Cost, avg | | | 2k + 4 |
| Runtime, Big O | | | O(k) average, O(k + n) worst if lookups degrade |
| Memory | | | O(1) |

**Hash Table — Print All Courses in Order**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | Line Cost | # Times Executes | Total Cost |
| gather all values via keys() | 1 | n | n |
| copy to vector | 1 | n | n |
| sort by course number, A to Z | log n | n | n log n |
| print each course | 1 | n | n |
| Total Cost | | | |  | | --- | |  |  |  | | --- | | n log n + 2n | |
| Runtime, Big O | | | O(n log n) |
| Memory | | | O(n) extra for the list |