# CS-300 Module 3 Pseudocode (Vector)

Start Program

Create a vector and name it “courses.”

FUNCTION openFile(fileName: string) RETURNS fileHandle

Attempt to open file with the given file name

IF opening is successful, return the file handle.

IF ELSE, display error message and exit program.

FUNCTION validateLine(line: string) RETURNS Boolean

Separate the line using commas into a list named “tokens.”

IF the length of the is < 2 return FALSE.

FOR every prerequisite in tokens (from index 2 to end):

IF prerequisite does not exist as a courseNumber in the file, return

RETURN TRUE

End FUNCTION

FUNCTION createCourseObject(line: string) RETURNS courseObject

Separate the line using commas into a list named “tokens.”

Create a course object with:

courseNumber = tokens[0]

name = tokens[1]

prerequisites = tokens[2 to end]

RETURN course object

FUNCTION loadCoursesFromFile(fileName: string)

fileHandle = openFile(fileName)

FOR every line in fileHandle:

IF validateLine(line) = TRUE:

Set course = createCourseObject(line)

Append course

IF ELSE, display an error message.

Close fileHandle

FUNCTION searchAndPrintCourse(courseNumber: string)

FOR every course in courses

IF course.courseNumber is equivalent to courseNumber:

Print course details and prerequisites

Return

Display error message “Course not found.”

MAIN

Query the user to input a file name

Retrieve file using loadCoursesFromFile function with the aforementioned file name.

IF TRUE:

Query the user to input a course number or input “exit” to quit.

IF input = “exit”, end

IF ELSE, use function searchAndPrintCourse with the user inputted course number

END

# Module 4 Pseudocode (HashTable)

Class HashTable:

Structure Node:

Define Bid, key, and next pointer.

Initialization:

Initialize vector of nodes and tableSize.

Function hash(key):

Calculate and return hash value.

Function Insert(bid):

Calculate the key for the given bid using its bidId.

Retrieve the node for the key.

If no node found, create a new one.

If node found, insert at the end of the chain.

Function PrintAll():

Loop through all nodes.

For each node, print its contents and then traverse its chain.

Function Remove(bidId):

Calculate key for bidId.

Find the node with the given key and remove it.

Function Search(bidId):

Calculate key for bidId.

Return the bid if found, else return empty bid.

End Class

Function loadBids(file):

Read from CSV.

For each row, create a bid and insert it into the hash table.

Function displayBid(bid):

Display the bid details.

MAIN:

Load CSV file.

While user does not exit:

Display menu.

Take user choice.

Perform the chosen operation.

End while.

End MAIN

# Module 5 Pseudocode (Tree)

STRUCTURE Bid:

bidId, title, fund, amount

STRUCTURE Node:

Bid data

Node pointer to left child

Node pointer to right child

CLASS BinarySearchTree:

Node pointer root

FUNCTION Insert(Bid bid):

IF root is null THEN

create new node with bid and make it root

ELSE

recursively add node in left or right subtree based on bidId comparison

FUNCTION Search(bidId):

START at root

WHILE current node is not null DO

IF current bidId matches THEN

RETURN bid

ELSE IF bidId is smaller THEN

move to left child

ELSE

move to right child

END WHILE

RETURN bid not found

FUNCTION Remove(bidId):

recursively remove node from left or right subtree based on bidId comparison

adjust links and possibly replace node with child or in-order predecessor/successor

FUNCTION InOrder():

recursively print left subtree, current node, then right subtree

FUNCTION loadBids():

OPEN CSV file

FOR each line in file DO

Extract bid data

Insert bid into BinarySearchTree

END FOR

FUNCTION main():

CREATE BinarySearchTree

DO

DISPLAY menu options

GET user choice

SWITCH user choice:

CASE load bids:

loadBids()

CASE display all bids:

CALL InOrder() on BST

CASE find bid:

CALL Search() on BST

CASE remove bid:

CALL Remove() on BST

END SWITCH

WHILE user choice is not exit

# Main Menu Function

FUNCTION mainMenu():

DISPLAY "Menu Options:"

DISPLAY "1. Load Data Structure"

DISPLAY "2. Print Course List"

DISPLAY "3. Print Course"

DISPLAY "4. Exit"

GET userChoice

RETURN userChoice

End FUNCTION

Sorting Functions:

# Vector:

FUNCTION sortVector(Vector<Course> courses) RETURNS Vector<Course>

FOR i = 0 TO length of courses - 1 DO

FOR j = i + 1 TO length of courses DO

IF courses[i].courseNumber > courses[j].courseNumber THEN

SWAP courses[i], courses[j]

END IF

END FOR

END FOR

RETURN courses

End FUNCTION

FUNCTION printCoursesVector(Vector<Course> courses)

sortedCourses = sortVector(courses)

FOR each course in sortedCourses DO

PRINT course.courseNumber, course.name

END FOR

End FUNCTION

# HashTable:

FUNCTION getCoursesFromHashTable(HashTable<Course> courses) RETURNS List<Course>

courseList = empty list

FOR each bucket in courses DO

FOR each course in bucket DO

APPEND course to courseList

END FOR

END FOR

RETURN courseList

End FUNCTION

FUNCTION printCoursesHashTable(HashTable<Course> courses)

courseList = getCoursesFromHashTable(courses)

sortedCourseList = sortList(courseList) // This can use a sorting function like the one in Vector

FOR each course in sortedCourseList DO

PRINT course.courseNumber, course.name

END FOR

End FUNCTION

# Tree:

FUNCTION inOrderTraversal(Node root)

IF root is NOT null THEN

inOrderTraversal(root.left)

PRINT root.course.courseNumber, root.course.name

inOrderTraversal(root.right)

END IF

End FUNCTION

FUNCTION printCoursesTree(Tree<Course> courses)

inOrderTraversal(courses.root)

End FUNCTION