CS-300 DSA: Analysis and Design

# 6-2 Project One Milestone

Rodrey McCoin

[rodrey.mccoin@snhu.edu](mailto:rodrey.mccoin@snhu.edu)

Southern New Hampshire University

//Vector - Milestone 1

Include libraries and headers

//define structure that will hold course data

Structure “Course”

//define variables

courseNumber

title

prerequisite Class Constructor ( string courseNumber, string title, string prerequisite)

**Pseudocode to open file, read file, parse each line, and check for errors**:

Call Main function

Create a new list for course called courseList

Read the file

If file is open

While getline

//Validate line format

If line is valid

Parse line

Else

Throw error “course format error”

No more lines, end loop

Else

Error message if file cannot be opened

Close file

Function to validate line

If parameters is >= 2

Return true

Else

Return false

Parse line function

Pull the course information from the line and separate with a “,”

Save courseNumber to space 0

Save title to space 1

Save prerequisite to space 2, 3, …n depending on number of prerequisites

Validate the prerequisites

For each prerequisite in prereqTable

If prerequisite is not in table

Throw error “prerequisite is not found”

exit

end if loop

end for loop

//Create course object and store in data structure

courseObject = CreateCourseObject(courseNumber, title, prerequisite)

//create vector to store class objects

Std::vector<string>vectorAppend = {courseVector, courseObject}

Validate the courses

For each course in courseVector

If course.courseNumber = courseNumber

Return true

Else

Return false

End for

**Pseudocode to create course objects and store them in data structures**:

//create course object

Course object (courseNumber, title, prerequisite)

//create new course object

newObject is equal to new courseObject

//create instance variables with data

newObject.courseNumber is equal to courseNumber

newObject.title is equal to title

newObject.prerequisites is equal to prerequisites

return newObject

//create procedure to store them in a vector

courseVector (vector, element)

//add an element to the end of vector

Vector[length of vector] equals element

End of vector procedure

**Pseudocode to search the data structure and print results**:

//procedure to search the data structure and print results

Create procedure printCourseInfo (courseNumber)

For each course in the course vector

If course.courseNumber is equal to courseNumber

Print “Course Number:”, course.courseNumber

Print “Course Title:”, course.title

Print “Prerequisites:”, course.prerequisite

Return

End if

Else throw error “No course found”

End for loop

End procedure

//Hash Table - Milestone 2

**Pseudocode to define how the program opens files, reads data, parses lines, and checks file format**:

Define the hashtable

Hashtable<course> courses

Open and read the file

Select which file is to be opened

Use scanner to scan file

Read the file

If file is open

While getline

//Validate line format

If line is valid

Parse line

Else

Throw error “course format error”

No more lines, end loop

Else

Error message if file cannot be opened

Close file

Function to validate line

If parameters is >= 2

Return true

Else

Return false

Parse line function

Pull the course information from the line and separate with a “,”

Save courseNumber to space 0

Save title to space 1

Save prerequisite to space 2, 3, …n depending on number of prerequisites

**Pseudocode to show how to create course objects and store them in data structure**:

//Create a course object in hash table

Course course equals new Course (courseNumber, title, prerequisite)

Put in Course (courseNumber, course)

**Pseudocode to print out course information and prerequisites**:

//print course information

Void print course information (hashtable<course>courses, string courseNumber) {

//check hash table to see if course exists

If (course containKey course number

System print “error: Course:” courseNumber “does not exist”

Return

//get course from hashtable

Course course equals get courseNumber

//print course information

Print “course number:” get courseNumber, /n “Course title:” get title, /n “Prerequisites:” get prerequisite

//Binary Search Tree - Milestone 3

**Pseudocode to define how the program opens files, reads data, parses lines, and checks file format:**

Open and read the file

Select which file is to be opened

Use scanner to scan file

Read the file

If does not exist

Display error and exit

Else

Create a tree data structure called courses

While getline

//Validate line format

If line is valid

Parse line

Else

Throw error “course format error”

No more lines, end loop

Close file

Function to validate line

If parameters is >= 2

Return true

Else

Return false

Parse line function

Pull the course information from the line and separate with a “,”

Save courseNumber to space 0

Save title to space 1

Save prerequisite to space 2, 3, …n depending on number of prerequisites

**Pseudocode to show how to create course objects and store them in data structure**:

//Create course object and store in tree data structure

Create class Course

Define courseNumber

Define title

Define prerequisite

Create constructor to hold (courseNumber, title, prerequisite)

This courseNumber equals courseNumber

This title equals title

This prerequisite equals prerequisite

End constructor

End class

**Pseudocode to print out course information and prerequisites:**

//Define function to traverse and display course info and prerequisites

Function display Courses node

Display courseNumber, title, prerequisite for the current node

If the node has children

For each child

Display courses with child

End for

End if

End function

Call display courses with root node of courses for tree data structure as argument

//Menu and sorting courses

**Pseudocode for menu:**

//create loop for menu

Set int choice equal to 0

While choice does not equal 4

Display “Menu:”

Display “1.) “Load data structure”

Display “2.) “Print course list”

Display “3.) “Print Course with prerequisites”

Display “4.) “exit”

Display “Enter choice:”

Get user input for menu option

If option 1

Get file

Call the data structure name

Set data structure to be returned

End if

If option 2

Call print\_course\_list from data structure

End if

If option 3

Call print\_course from data structure then course name

End if

If option 4

Display “You are exiting the program!”

Exit loop

End if

End while loop

**Pseudocode for sorting courses in alphanumeric order and print out list:**

**Vector:**

Create sortVector (Vector courses)

//sort courses in alphanumeric order

For I from 0 to couses size minus 2 do

Minimum index equal to i

For j from I plus 1 to course size minus 1 do

If courses [j].courseNumber is less than courses{minimum index].courseNumber then minimum index equals j

Swap course i with courses minimum index

Print course list from vector courses

sortCourses(courses)

for each course in courses do

print course.courseNumber, course.title

**Hash Table:**

//express hash table as an array

Hashtable string, course from course table equals

Get computer science course table

Vector string keys equals course table.get keys

Sort keys begin, keys end

For each key in keys

Print key, course table. Get key.course name

**Tree:**

//create in order traversal to sort tree

Create inOrderTraversal (Node node)

If node is not null then

Recusively traverse the left subtree

Print node for courseNumber, node for title

Recursively traverse the right subtree

**Vector:**

|  |  |  |  |
| --- | --- | --- | --- |
| code | Cost per line | Number of times executed | Total cost |
| Opens file | 1 | O(n) | O(n) |
| Reads data from file | 1 | O(n) | O(n) |
| Parses each line | O(log n) | O(n) | O(n log n) |
| Checks for formatting errors | O(1) | O(n) | O(n) |
| Create course objects | O(log n) | O(n) | O(n log n) |

**Hash Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| code | Cost per line | Number of times executed | Total cost |
| Opens file | 1 | O(n) | O(n) |
| Reads data from file | 1 | O(n) | O(n) |
| Parses each line | O(log n) | O(n) | O(n log n) |
| Checks for formatting errors | O(1) | O(n) | O(n) |
| Create course objects | O(log n) | O(n) | O(n log n) |

**Tree:**

|  |  |  |  |
| --- | --- | --- | --- |
| code | Cost per line | Number of times executed | Total cost |
| Opens file | 1 | O(n) | O(n) |
| Reads data from file | 1 | O(n) | O(n) |
| Parses each line | O(log n) | O(n) | O(n log n) |
| Checks for formatting errors | O(1) | O(n) | O(n) |
| Create course objects | 1 | O(n) | O(n) |

**Advantages and Disadvantages:**

**Vector:**

* **Advantages:**
  + Automatically adjusts size when an item is added or removed from the structure.
  + Easy to add an remove items as the last element in the structure.
* **Disadvantages:**
  + Harder to access data in the middle.

**Hash Table:**

* **Advantages:**
  + Quick data retrieval time.
  + Compresses data to a smaller size.
* **Disadvantages:**
  + Data (hash) collision: two different inputs may produce the same output, which can cause poor performance, security, integrity for the system and the data.

**Tree:**

* **Advantages:**
  + Flexible size, changes as nodes are added and removed.
  + Easy to maintain.
  + Easy to add and remove nodes from the structure.
* **Disadvantages:**
  + Large memory
  + Not efficient for sorting and grouping

**Recommended Data Structure:**

The data structure that I believe will be most efficient in my code is the hash table. The hash table is the best option for this program because it allows you to search and retrieve classes from the structure faster than the other two. Hash tables will also allow you to use less memory to store the data by compressing the data into smaller sizes. Being able to access the course information at a quick rate and sort and group the data is the main focus when determining the data structure that would work best for this project, and the structure that met these requirements the best is the hash table.