Joseph Silva

CS-300 Project 1

10/6/2025

**Pseudocode for creating and storing courses (Vector)**:

**NOTE:** Validation logic worked into this function to save from opening/parsing the file more than once and to validate prerequisites after everything is loaded.

//Function declaration, return a vector

FUNCTION vector generateCourseList(file)

//Initialize vector for storing the course objects

DECLARE vector <Course> courses

//Initialize vector for storing lines that have issues

DECLARE vector <String> issueLines

//Define a structure to store the data

STRUCTURE Course

STRING courseName

STRING courseIdNumber

Vector<string> prerequisites

END STRUCTURE

OPEN file

READ the file using the appropriate parser or function

//Start a while loop and go through the file line by line

WHILE the line read is not equal to the end of the file

//If there is no course ID or name

IF the current line does not have a course name and ID

PUSH line into issueLines vector

ENDIF

//Otherwise create a course structure, set the attributes and push to the vector

ELSE

DECLARE newCourse : Course

SET courseName TO corresponding field in file

SET courseNumber TO corresponding field in file

//If there are prerequisites, put them in the prerequisites vector

WHILE there is data after the course number

PUSH the prerequisites to the vector

ENDWHILE

ENDELSE

ENDWHILE

CLOSE file

//Check for lines with formatting issues and print them to the console

IF the issueLines vector is greater than 0

PRINT there were formatting issues found, some courses were not added

FOR the size of the issueLines vector

PRINT the line with issues

ENDFOR

ENDIF

//Validate the prerequisites after everything is loaded

FOR each course ID number in the prerequisites vector

FOR the size of the courses vector

IF the prerequisites course ID doesn’t exist in the courses vector

PRINT please check the file, course “x” has invalid prerequisites

ENDIF

ENDFOR

ENDFOR

RETURN courses vector

END FUNCTION

**Pseudocode for search and print course (Vector):**

//Declare a function that displays the course requested by the user

FUCNTION void (vector <Courses> courses, String CourseIdNumber)

//Loop through the courses vector looking for the given courseIdNumber

FOR the size of the course vector

//If the given course exists in the list, print the course and prerequisite information

IF a course ID number is equal to the given courseIdNumber

PRINT the course information to the console

IF the prerequisites vector is greater than 0

FOR the size of the course prerequisites vector

PRINT the prerequisite course name and ID to the console

ENDFOR

ENDIF

//Notify the user that the course doesn’t exist

ELSE

PRINT course does not exist

ENDELSE

ENDFOR

END FUNCTION

**Pseudocode for sorting a vector alphanumerically (vector):**

//Using selection sort to sort the vector

FUNCTION void selectionSort(vector courses)

//Variable declaration

SET integer minimum TO 0

//check the vector’s size if less than 1 nothing to do

IF the vector size is less than one

RETURN

ENDIF

SET variable size\_t n TO the size of the vector

//start outer for loop

FOR SET size\_t pos TO zero, loop until pos is less than n minus one, INCREMENT pos

//set variable min to pos

min equals pos

//start inner for loop for comparisons

FOR SET size\_t j TO zero, loop until j is equal to pos plus one, INCREMENET j

// check to see if the current element is the minimum

IF the bids title in position j is less the bids tile in position min

SET min equal TO j

ENDIF

ENDFOR

//swap the current minimum with the minimum found

IF the minimum element is not equal to pos

swap element min and pos

ENDIF

ENDFOR

END FUNCTION

**Pseudocode for printing a vector alphanumerically (vector):**

FUNCTION void printSortedCoursesVector(vector Course)

//Loop through sorted vector and print

FOR SET i TO 0 loop for the size of the courses vector and INCREMENT i

PRINT courses one at a time

//If there are prerequisites, print them for each course

IF the prerequisites vector is greater than 0

FOR SET j TO 0 for the size of the prerequisites vector INCREMENT j

PRINT the prerequisites for each course as well

ENDFOR

ENDIF

ENDFOR

END FUNCTION

**Pseudocode for creating and storing courses (Hash Table)**:

**NOTE:** Validation logic worked into this function to save from opening/parsing the file more than once and to validate prerequisites after everything is loaded.

//Function declaration, populate and return a hash table

FUNCTION Hashtable generateCourseList(file file, Hashtable hashtable)

//Initialize vector for storing lines that have issues

DECLARE vector <String> issueLines

//Define a structure to store the course data.

STRUCTURE Course

STRING courseName

STRING courseIdNumber

Vector<string> prerequisites

END STRUCTURE

OPEN file

READ the file using the appropriate parser or function

//Start a while loop and go through the file line by line

WHILE the line read is not equal to the end of the file

//If there is no course ID or name

IF the current line does not have a course name and ID

PUSH line into issueLines vector

ENDIF

//Otherwise create a course structure, and set the attributes

ELSE

DECLARE newCourse : Course

SET courseName TO corresponding field in file

SET courseNumber TO corresponding field in file

//If there are prerequisites, put them in the prerequisites vector.

WHILE there is data after the course number

PUSH the prerequisites to the vector

ENDWHILE

//Now we need logic to insert the structure into the hash table.

//Use the course number, convert it from string to integer and do //modulus division with the hash table size to set a key.

SET key TO a value based on modulus division

//Start at the node corresponding to the key

CREATE a pointer to the node at the key, call It node

//If the node doesn’t exist, add a node with a course and key

IF node is NULL

CREATE a new node, add a course and a key

ENDIF

ELSE

//This will handle the case where we are inserting a new head

//into a node that exists.

IF the node exists but the key is a default value

SET the key TO the key

SET the course TO the course

SET the next node to NULL

ENDIF

//This block handles the collision case

ELSE

//Loop through the linked list looking for a NULL next node

WHILE the next node is not NULL

Go to the next node

ENDWHILE

//When a NULL next node is found, create a node and populate it

CREATE a new node at next with a course and a key

ENDELSE

ENDELSE

ENDWHILE

CLOSE file

//Check for lines with formatting issues and print them to the console

IF the issueLines vector is greater than 0

PRINT there were formatting issues found, some courses were not added

FOR the size of the issueLines vector

PRINT the line with issues

ENDFOR

ENDIF

//Validate the prerequisites after everything is loaded

FOR each course ID number in the prerequisites vector

//Use the prerequisite course number, convert it from string to integer and do //modulus division with the hash table size to set a key.

CREATE a preReqKey from the course ID number in the prerequisites vector

CREATE a pointer to the node in the hashtable that corresponds with preReqKey

//Check the node for the key

IF the node is null or the key is a default value

PRINT please check the file, course “x” has invalid prerequisites

ENDIF

ENDFOR

RETURN the hashtable

END FUNCTION

**Pseudocode for search and print course (Hash Table):**

//Declare a function that displays the course requested by the user

FUNCTION void (Hashtable hashtable , String CourseIdNumber)

//Create a key for the course ID that is passed in

CREATE a key for the course ID passed in

//Create a pointer to the node that corresponds with the key

CREATE a pointer to the node that corresponds to the key

//If the node is NULL or the key doesn’t exist, return there is nothing to do

IF the node is NULL or the key in the node is a default value

RETURN

ENDIF

//This will handle the case where the course we are looking for is the head

//If the course number is found print

IF the courseIdNumber that corresponds to the key is found

PRINT the courseName and courseIdNumber

//If there are prerequisites, loop through the prerequisites vector and

//print them

FOR the size of the prerequisites vector

PRINT the prerequisites courseName and courseIdNumber

ENDFOR

ENDIF

RETURN

//This will handle the case where the course we are looking for is a member in the //linked list (collision occurred).

SET node to the next node

//Loop through the linked list looking for the courseIdNumber

WHILE the node is not NULL

//If the key is found print

IF the courseIdNumber that corresponds to the key is found

PRINT the courseName and courseIdNumber

//If there are prerequisites, loop through the prerequisites vector and

//print them

FOR the size of the prerequisites vector

PRINT the prerequisites courseName and courseIdNumber

ENDFOR

ENDIF

ENDWHILE

END FUNCTION

**Pseudocode for moving hash table to vector (Hash Table):**

//Move all course objects to a vector so we cab sort using selection sort

FUNCTION vector hashTableToVector(hash table)

//Declare a vector to hold all course objects

DECLARE vector <Course> courseVector

//Loop through the nodes vector and extract the courses

FOR SET intetger i TO 0 loop until i is less than nodes INCREMENT i

//Start at the first node

CREATE a pointer to the first node in the table

IF the node is not NULL and the key is not a default value

PUSH BACK course to courseVector

WHILE the next node is not null

PUSH BACK course to courseVector

ENDWHILE

ENDIF

ENDFOR

RETURN courseVector

END FUNCTION

**Pseudocode for sorting a vector alphanumerically (Hash Table):**

//Using selection sort to sort the vector

FUNCTION void selectionSort(vector courseVector)

//Variable declaration

SET integer minimum TO 0

//check the vector’s size if less than 1 nothing to do

IF the vector size is less than one

RETURN

ENDIF

SET variable size\_t n TO the size of the vector

//start outer for loop

FOR SET size\_t pos TO zero, loop until pos is less than n minus one, INCREMENT pos

//set variable min to pos

min equals pos

//start inner for loop for comparisons

FOR SET size\_t j TO zero, loop until j is equal to pos plus one, INCREMENET j

// check to see if the current element is the minimum

IF the bids title in position j is less the bids tile in position min

SET min equal TO j

ENDIF

ENDFOR

//swap the current minimum with the minimum found

IF the minimum element is not equal to pos

swap element min and pos

ENDIF

ENDFOR

END FUNCTION

**Pseudocode for printing a vector alphanumerically (Hash Table):**

FUNCTION void printSortedCoursesVector(vector courseVector)

//Loop through sorted vector and print

FOR SET i TO 0 loop for the size of the courses vector and INCREMENT i

PRINT courses one at a time

//If there are prerequisites, print them for each course

IF the prerequisites vector is greater than 0

FOR SET j TO 0 for the size of the prerequisites vector INCREMENT j

PRINT the prerequisites for each course as well

ENDFOR

ENDIF

ENDFOR

END FUNCTION

**Pseudocode for creating and storing courses (Binary Tree)**:

**NOTE:** Validation logic worked into this function to save from opening/parsing the file more than once and to validate prerequisites after everything is loaded.

FUNCTION Binary tree generateCourseList(file file, Tree<Course> courses)

//Initialize vector for storing lines that have issues

DECLARE vector <String> issueLines

//Define a structure to store the course data.

STRUCTURE Course

STRING courseName

STRING courseIdNumber

Vector<string> prerequisites

END STRUCTURE

OPEN the file

READ the file using the appropriate parser or function

//Loop through the file line by line

WHILE the line read is not equal to the end of the file

//If there is no course ID or name

IF the current line does not have a course name and ID

PUSH line into issueLines vector

ENDIF

//Create a course structure and set the attributes

ELSE

DECLARE newCourse : Course

SET courseName TO corresponding field in file

SET courseNumber TO corresponding field in file

//If there are prerequisites, put them in the prerequisites vector.

WHILE there is data after the course number

PUSH the prerequisites to the vector

ENDWHILE

INSERT the course object into the binary tree

ENDELSE

ENDWHILE

CLOSE file

//Check for lines with formatting issues and print them to the console

IF the issueLines vector is greater than 0

PRINT there were formatting issues found, some courses were not added

FOR the size of the issueLines vector

PRINT the line with issues

ENDFOR

ENDIF

//Validate the prerequisites after everything is loaded

FOR each course ID number in the prerequisites vector

SET the current node to the root node

WHILE the current node is not NULL

TRAVERSE the tree searching for the node with the prerequisite courseID

IF the courseID is not found

PRINT please check the file, course “x” has invalid prerequisites

ENDIF

ENDWHILE

ENDFOR

RETURN the binary tree

END FUNCTION

**Pseudocode for inserting courses (Binary Tree)**:

FUNCTION void addCourse(Tree<Course> courses, course)

//Check if the root node is NULL, create it if it is

IF the root node is NULL

CREATE the root node and add a course object

ENDIF

//When we have a root node, handle the left subtree

ELSE IF the course ID passed in is less than the value in the current node

//Add a node to the left with the course object

IF the left node is NULL

CREATE a leftnode and add the course object

ENDIF

//When the left node is not NULL, traverse the left subtree

//recursively until a NULL node is found

ELSE

CALL addCourse, pass the left node (current) and course

ENDELSE

//Left side tree handling ends here

ENDELSE IF

//handling for the right subtree

ELSE

//Add a node to the right with the course object

IF the right node is NULL

CREATE a right node and add the course object

ENDIF

//When the right node is not NULL, traverse the right subtree

//recursively until a NULL node is found

ELSE

CALL addCourse, pass the right node and course

ENDELSE

//Right tree handling ends here

ENDELSE

END FUNCTION

**Pseudocode for searching courses (Binary Tree)**:

FUNCTION void (Tree<Course> courses, String CourseIdNumber)

SET the current node to root

//Start a loop to find the course ID that matched the one being searched

WHILE the current node is not NULL

//Check to see if the course ID matches, if it does print

IF the course ID in the current node is equal to the one being searched

PRINT the courseName and courseIdNumber

//If there are prerequisites, loop through the prerequisites vector and print

FOR the size of the prerequisites vector

PRINT the prerequisites courseName and courseIdNumber

ENDFOR

ENDIF

//Traverse the left subtree when the course ID in the current node is greater //than the one being searched.

IF the course ID in the current node is greater than the one being searched

SET the current node to its left node

ENDIF

//Traverse the right subtree when the course ID in the current node is less //than the one being searched.

ELSE

SET the current node TO its right node

ENDELSE

ENDWHILE

//If we get here the course wasn’t found

PRINT a message stating the course wasn’t found

END FUNCTION

**Pseudocode for sorting a Binary tree in order (Binary Tree):**

//Declare the inOrder function, we take the root node as input

FUNCTION void inOrder(Node node)

IF the node is not NULL

//Traverse the left subtree recursively

CALL inOrder and pass the left node

PRINT the course information

//Check the prerequisites vector

IF the prerequisites vector is greater than 0

FOR the size of the prerequisites vector

PRINT the prerequisites

ENDFOR

ENDIF

//Traverse the right subtree recursively

CALL inOrder and pass the right node

ENDIF

END FUNCTION

**Pseudocode for main menu:**

//Declare variable to take the user’s input

SET integer choice TO 0

//Loop until the use exits the program

WHILE choice is not equal to 9

PRINT menu

PRINT 1. Load courses

PRINT 2. Print all courses in alpha numeric order

PRINT 2. Print course and prerequisites for course

PRINT 9. Exit

PRINT Enter choice

//Get user input

INPUT choice

//Create a switch case

SWITCH choice

CASE 1:

Call function to load courses

BREAK

CASE: 2

CALL function to print all courses

BREAK

CASE 3:

CALL function to print course and prerequisites

BREAK

END SWITCH

ENDWHILE

//Exit on input of 9

PRINT you exited the program

**Big O analysis:**

Vector

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector Analysis** | | | |
|  | **Line cost** | **Time executed** | **Total cost** |
| **Load Course File** | O(1) | O(1) | O(1) |
| **Read each line** | O(1) | O(1) | O(n) |
| **Create Course Object** | O(n) | O(n) | O(n) |
| **Push Back to Vector** | O(1) | O(1) | O(1) |
|  |  | **Total cost** | 2n + 2 |
|  |  | **Run Time** | O(n) |

Hash Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Hash Table Analysis** | | | |
|  | **Line cost** | **Time executed** | **Total cost** |
| **Load Course File** | O(1) | O(1) | O(1) |
| **Read each line** | O(1) | O(1) | O(n) |
| **Create Course Object** | O(n) | O(n) | O(n) |
| **Insert Worst case** | O(n) | O(n) | O(n) |
|  |  | **Total cost** | 3n + 1 |
|  |  | **Run Time** | O(n) |

Binary Tree

|  |  |  |  |
| --- | --- | --- | --- |
| **Binary Tree Analysis** | | | |
|  | **Line cost** | **Time executed** | **Total cost** |
| **Load Course File** | O(1) | O(1) | O(1) |
| **Read each line** | O(1) | O(1) | O(n) |
| **Create Course Object** | O(n) | O(n) | O(n) |
| **Insert Worst case** | O(n) | O(n) | O(n) |
|  |  | **Total cost** | 3n + 1 |
|  |  | **Run Time** | O(n) |

**Advantages, disadvantages and recommendation:**

**Vector** – Some advantages of using a vector are its dynamic resizing which removes the burden of manual memory management and there are many built-in functions available for vectors in C++ which simplifies data manipulation. In addition, the at() function provides bounds checking which helps prevent common programming errors and a vector uses contiguous memory allocation which can result in better overall cache performance.

Some disadvantages are the overhead involved with dynamic resizing, the poor performance when inserting or deleting middle elements and the overall memory overhead.

**Hash Table** – Some advantages for using a hash table are a fast lookup mechanism due to the use of key and value pairs and flexibility as hash tables can store all kinds of data types and be efficient for simple or complex lookups. Hash tables can also be designed with a resizing algorithm built in adding another layer of efficiency to the implementation.

Some disadvantages are that hash tables can be inefficient when there are lots of collisions, they can be complicated to implement, and they do not maintain any order which makes retrieving elements in sorted order difficult.

**Binary tree** – Some advantages of using a binary search tree are efficient searching capabilities due to the tree being kept in order. This makes it easier to retrieve the nodes in sorted order and print them when needed. Nodes can be added and deleted dynamically, and a balanced tree adds efficiency.

Some disadvantages are that trees are not self-balancing and worst-case time complexity can be costly. They also consume lots of memory, are not suited for larger data sets and offer limited functionality outside of search, add and delete.

**Recommendation**

For the implementation of the ABCU computer science course catalog I think the best choice is using a binary search tree to store the data. The data set is small, and this data structure will keep the catalog in sorted order which will give the users an efficient search mechanism and allow for all courses to be easily retrieved and printed in sorted order. I don’t think the cost of creating the course objects and populating the data structure is a big factor in the decision. Creating the course objects will have a worst case of O(n) depending on the number of prerequisites and while a vector would offer O(1) efficiency when inserting the object, that’s assuming it never has to be resized. I didn’t think the cost advantage of inserting into a vector outweighs the sort and search efficiency of the binary tree especially considering the courses will only be inserted once per run.

The most important functionality that ABCU requires is being able to look up courses and print the entire course list in alpha numeric order. This data structure is the most efficient in accomplishing these tasks. If the computer science catalog is expanded in the future, we may need to reassess this choice depending on how large the data set grows.

**References**

GeeksforGeeks. (2025, July 12). Advantages of vector over array in C++. [https://www.geeksforgeeks.org/cpp/advantages-of-vector-over-array-in-c/](https://www.geeksforgeeks.org/cpp/advantages-of-vector-over-array-in-c/%20%20%20)

GeeksforGeeks. (2025, July 23). Applications, advantages and disadvantages of hash data structure. <https://www.geeksforgeeks.org/dsa/applications-advantages-and-disadvantages-of-hash-data-structure/>

GeeksforGeeks. (2025, July 23). Applications, advantages and disadvantages of binary search tree. <https://www.geeksforgeeks.org/dsa/applications-advantages-and-disadvantages-of-binary-search-tree/>