**Week 6- Project 1**

**Tammy Hartline**

**SNHU-CS300**

**04/08/2023**

***Menu Output and Loop Pseudocode with File Open and Read Pseudocode***

CREATE function processFile() {

PROMPT user to enter *filename*

CHECK if *filename* exists and is a relevant path

IF *filename* exists and NO errors are found

OPEN *filename*

OUTPUT Menu()

ELSE

OUTPUT error

RETURN -1

WHILE the *filename* file IS OPEN AND file IS NOT EOF (at end of file)

GET file line

IF file line has LESS than 2 values (params)

PRINT “File “ line # “is missing vital course information.”

RETURN -1

CLOSE file

}

***Start of Menu code***

CREATE FUNCTION displayMenu() {

PRINT to screen

1. Load Data from File
2. Print all Course Info from File
3. Search if Course Exists

9. EXIT Program

PROMPT user to enter menu selection using cin

WHILE user entry is NOT equal to 9

CASE 1

CALL processFile()

BREAK

CASE 2

WHILE not at end of file

OUTPUT ALL sorted file lines

BREAK

CASE 3

IF user entry matches file line object

OUTPUT file line containing name, number, and prereqs if applicable

CASE 9

OUTPUT “Good-bye”

Return 0

DEFAULT

DisplayMenu()

***Vector Pseudocode***

IMPORT needed libraries for File PARSING, READ, WRITE, OPEN, CLOSE…

CALL processFileFunction();

CREATE struct courseInfo {

DECLARE int courseNumber

DECLARE string courseName

DECLARE vector<string> coursePrereq

INITIALIZE variable string infile and set = to open file function

INITIALIZE variable string readLine and set = read file function

DECLARE vector<Course>courses

}

WHILE file open fails

OUTPUT error “File not found”

RETURN -1

WHILE inFile is not null and not at EOF

THEN file opened successfully and can be read

LOAD file

READ file and set line = readLine

WHILE readLine is not at end of file and readLine contains objects

IF readline contains NO delimiters OR readLine does not contain courseName

and courseName and CourseNumber

PRINT Error

RETURN “File is missing data.”

IF readLine contains at least 1 delimiter(,)

USE sstream to store 1st object in line as courseNumber AND

MOVE to vector courses

THEN USE sstream to store 2nd object to courseName

AND MOVE to vector courses

ADD to vector coursePrereq

CONTINUE to next line

IF readLine contains MORE THAN 1 (,)

IF object 3 or GREATER EXISTS in courses

ADD each object to DECLARE string prereq

MOVE each object to coursePrereq

CONTINUE TO displayMenu()

IF user chooses Load Data (1)

OPEN file

PRINT “File has been loaded”

displayMenu()

IF user chooses to printALL (2)

READ each readLine and follow vecor code to pass in courses

ONCE each line has been read and program reaches EOF

SORT vector alphanumerically

PRINT all sorted lines to screen in order

displayMenu()

IF user chooses Find Course (3)

PROMPT user to enter courseName

IF users entry for courseName exists in vector coursePrereqs

PRINT line with that courses information

WHILE users entry does NOT exist in vector coursePrereq

PRINT “Sorry, that course was not found.”

BREAK

displayMenu()

IF user chooses EXIT (9)

RETURN Good-bye

ELSE

displayMenu()

| Vector | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| For each string | 1 | n | n |
| courseName-> num = first parsed string | 1 | 1 | 1 |
| courseName-> name = second parsed string | 1 | 1 | 1 |
| If less than 2 objects  Error  Return | 1 | 1 | 1 |
| For each remaining string | 1 | n | n |
| If prerequisite name found in courseName | 1 | n | n |
| Prerequisites = remaining parsed string | 1 | n | n |
| courseName->prerequisite = prerequisite | 1 | n | n |
| Add prerequisite(s)  Courses.push\_back(\*courseName) | 1 | n | n |
| **Total Cost** | | | 6n + 3 |
| **Runtime** | | | O(n) |

***Hash table Pseudocode***

CREATE Hash Table struct {

CREATE array named courseInfo

CREATE linked list courseName, courseNumber

CREATE linked list coursePrereq

ADD both linked lists (buckets) to courseInfo

}

DECLARE hash function {

INITIALIZE hash function as INPUT key = courseName

CREATE key using modulo % to GENERATE index for each key value

}

USE fstream to open and read file as described in vector pseudocode

FOR each readLine/ROW where readLine/ROW is NOT NULL

CALL hash function to calculate index

CREATE new node from courseInfo

INSERT new node into linked list where node = key index

CREATE findCourse() function for SEARCH

PROMPT user entry for courseName

IF courseName does not exist in courseInfo

PRINT ‘Course not found. ‘

displayMenu()

IF courseName exists in courseInfo

CALL has function to calculate index where key = courseName

TRAVERSE linked list where index = courseName

PRINT courseInfo from matching index

displayMenu()

CREATE printALL function for printing entire file

ASSUME file is already sorted

PRINT all lines from file

displayMenu()

CREATE exitProgram() function

Return Good-Bye

USE THE ABOVE CODE to implement displayMenu() based on user selection by calling the appropriate function created above

| Hashtable | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| vector<Node> nodes(store course info) | 1 | n | n |
| hash(key) | 1 | n | n |
| return key | 1 | n | N |
| Get course information | 1 | n | N |
| Insert(course) | 1 | n | N |
| Key = courseName | 1 | n | N |
| Determine correct node position and place | 1 | n | N |
| **Total Cost** | | | 7n |
| **Runtime** | | | O(n) |

***Binary Search Tree Pseudocode***

**IMPORT needed libraries for File PARSING, READ, WRITE, OPEN, CLOSE…**

**IN MAIN**

PROMPT user to enter *filename*

CHECK if *filename* exists and is a relevant path

IF *filename* exists and NO errors are found

OPEN *filename*

WHILE the *filename* file IS OPEN AND file IS NOT EOF (at end of file)

GET file line

IF file line has LESS than 2 values (params)

OUTPUT Error “File “ line # “is missing vital information.”

RETURN

IF file line has 2 parameters

CONTINUE to next STRUCT

IF file line has MORE than 2 parameters

CONTINUE to next STRUCT

ELSE

OUTPUT ERROR message informing user file was not opened successfully

RETURN ERROR

CLOSE file;

END

**CREATE STRUCT with Binary Search Tree for all course objects**

INITIALIZE STRUCT called Course

CREATE int courseNumber variable;

CREATE string courseName variable;

CREATE string preReqCourse variable;

DECLARE Course::CourseID();

DECLARE Course::CourseName();

DECLARE Course::CoursePreReqs();

DECLARE Course::NoPreReqCourseInfo with (CourseID, CourseName);

DECLARE Course::PreReqCourseInfo with (CourseID, CourseName, and CoursePreReqs);

WHILE NOT EOF(end of file)

CREATE delimiter(,) to separate each cell (row and column) in file.

FOR each line’s first container (content in column 1, per row),

SET container EQUAL to courseNumber;

CALL CourseID and pass in courseNumber;

FOR each line’s second container (content in column 2, per row),

SET container EQUAL to courseName;

CALL CourseName and pass in courseName;

IF line only has 2 containers (params)

CALL NoPreReqCourseInfo and PASS in

CourseID and CourseName;

ELSE

BREAK

FOR each line that contains MORE THAN 2 parameters

SET all containers that are NOT null,

AND AFTER delimiter 2 EQUAL to preReqCourse;

CALL CoursePreReqs and PASS in preReqCourse;

CALL PreReqCourseInfo and PASS in

CourseID, CourseName and CoursePreReqs;

DECLARE CourseInfoOutput function;

DECLARE SEARCH function;

IF user enters search for course number not found while reading file

OUTPUT ERROR course not found

IF user enters search for course number that has linked PreReqs

OUTPUT PreReqCourseInfo with line passed in

ELSE

OUTPUT NoPreReqCourseInfo with line passed in

CREATE PrintAllCourseInfo function

IF user selects to view all courses

OUTPUT all lines from PreReqCourseInfo & NoPreReqCourseInfo

RETURN

CLOSE FILE;

END;

| **Binary Search Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Getline** | 1 | n | n |
| **SET courseNumber = 1st object** | 1 | n | n |
| **SET courseName = 2nd object** | 1 | n | n |
| **SET coursePrereq = objects in index > 2** | 1 | n | n |
| **Insert (Course course)** | 1 | n | n |
| **Insert(course)** | 1 | n | n |
| **If course is null** | 1 | n | n |
| **course = new node(courseName)** | 1 | N | n |
| **addNode(course,courseName)** | 1 | n | n |
| **Total Cost** | | | 9n |
| **Runtime** | | | O(n) |

**Advantages and Disadvantages of Vectors, Hash Tables, and Binary Search Trees**

***Vectors***

Advantages:

* Expandable storage
* Automated storage management
* Easily processes changing data
* Offers efficiency without sacrificing flexibility

Disadvantages:

* Can take up more memory than an array or other method
* Is mainly useful when storing elements that are not predetermined

***Hash Table***

Advantages:

* Insert, search, and delete can be done in 0(1)
* Can store large amounts of data

Disadvantages:

* Duplicate keys, causing collisions and value errors
* Can be expensive
* Difficult to implement

***Binary Search Tree***

Advantages:

* Better representation of relationships between data
* Stores arbitrary number of data values

Disadvantages:

* Deleting nodes can be complex
* Insertion, search and deletion are dependent on the height of search tree

***Recommendation***

Based on the runtime analysis using Big O, the obvious choice seems to be a vector as it appears to have less line and time cost. However, I like the idea of using a Binary Search Tree over a vector, mainly because I am more familiar with the structure, and the advantage it provides related to “connecting the dots,” so to speak, and recognizing relationships between data. While I know most find a vector more user friendly and can be considered as the more efficient way to go, I think that whenever BST are done correctly and taking all optimizations into consideration, it can have the better performance and better features when looking at the life of the program, and the way it will operate. I find BST to be less prone to mixing up values or mismatching data, which ultimately in the long run, will prove to be a more effective structure and solution than a vector or hash table.