**CS-300 DSA: Analysis and Design**

**Module 3**

**Project One Milestone One**

**Vector Data Structure**

**Submitted By:**

HAMNA KHALID

ID: 2902671

[hamna.khalid@snhu.edu](mailto:hamna.khalid@snhu.edu)

**Submitted To:**

PROFESSOR SATHISH GOPALAKRISHNAN

s.gopalakrishnan1@snhu.edu

Southern New Hampshire University

Wednesday, September 17, 2025

**Vector Data Structure**

# **PSEUDOCODE**

Load, Store, and Print Course Data (Vector)

# **GOAL**

Design pseudocode that loads course data from a file, validates format & prerequisites, stores each course as an object in a **vector**, and supports printing a specific course’s info and prerequisites.

# **AUTHOR**

HAMNA KHALID

ID: 2902671

**Department Of Computer Science**

**ABC University**

# **Introduction**

For this milestone, I created pseudocode for ABC University’s Computer Science department. The goal is to help advisors quickly access student course information, including course numbers, titles, and prerequisites. All of this data is stored in a simple text file (CSV format), and my pseudocode demonstrates how to import the file, verify the format, and then store the information in a vector of course objects.

A vector is the chosen data structure because it is dynamic (can grow automatically as more courses are added), provides fast access by index, and is easy to traverse sequentially. Each course is represented as an object with fields for the number, title, and prerequisites. Once stored in the vector, the courses can be searched by number and printed out so advisors can view details and verify prerequisites.

This milestone is important because it establishes the foundation for Project One. The pseudocode illustrates a design plan that I can later convert into a full C++ implementation with classes, methods, and working file I/O.

# **Pseudocode**

## **File Input and Validation**

This part of the pseudocode explains how the program will open and read the file. Because data can sometimes be messy, validation is important to make sure the file is usable.

function loadCoursesFromFile(fileName):

try:

open file with name fileName

catch error:

print "Error: unable to open file."

exit program

create empty Vector<Course> courses

create empty Set courseNumbers // used to check prerequisites later

// -------- First Pass: Collect all course numbers --------

for each line in file:

split line into tokens by comma

if tokens.size < 2 then:

print "Error: Line does not contain both course number and title"

continue

// Note: some courses may not have prerequisites; in that case, only course number and title are required.

courseNumber ← tokens[0]

add courseNumber to courseNumbers

reset file pointer to beginning

// -------- Second Pass: Create course objects --------

for each line in file:

split line into tokens by comma

courseNumber ← tokens[0]

courseTitle ← tokens[1]

prerequisites ← remaining tokens after index 1

// Validate prerequisites

for each prereq in prerequisites:

if prereq not in courseNumbers:

print "Warning: prerequisite " + prereq + " does not exist in file"

skip this course

newCourse ← Course(courseNumber, courseTitle, prerequisites)

append newCourse to courses

close file

return courses

* The file is read twice. In the first pass, I collect all course numbers so that prerequisites can be checked later. This ensures that no prerequisite points to a non-existent course. In the second pass, I create actual course objects and append them to the vector. If a prerequisite is missing from the course list, the program warns the user instead of crashing.

## **Course Object**

struct Course:

courseNumber : string

title : string

prerequisites : Vector<string>

* The Course struct is a simple container for the course’s data. Each course object represents exactly one line from the input file. The prerequisites are stored as a vector of strings since each course can have zero, one, or many prerequisites. This design makes it easy to loop over the prerequisites later when printing or validating.

## **Print All Courses**

function printAllCourses(courses):

if courses is empty:

print "No courses found in the file."

return

for each course in courses:

print "Course Number: " + course.courseNumber

print "Title: " + course.title

if course.prerequisites is not empty:

print "Prerequisites:"

for each prereq in course.prerequisites:

print " " + prereq

else:

print "Prerequisites: None"

* This function prints out all courses in the order they were loaded into the vector. It shows the course number, title, and prerequisites. The function also checks if the course has no prerequisites and prints “None” instead of leaving it blank. This makes the output more user-friendly for advisors.

## **Search for a Specific Course**

function searchCourse(courses, searchNumber):

for each course in courses:

if course.courseNumber = searchNumber:

print "Course Number: " + course.courseNumber

print "Title: " + course.title

if course.prerequisites is not empty:

print "Prerequisites:"

for each prereq in course.prerequisites:

print " " + prereq

else:

print "Prerequisites: None"

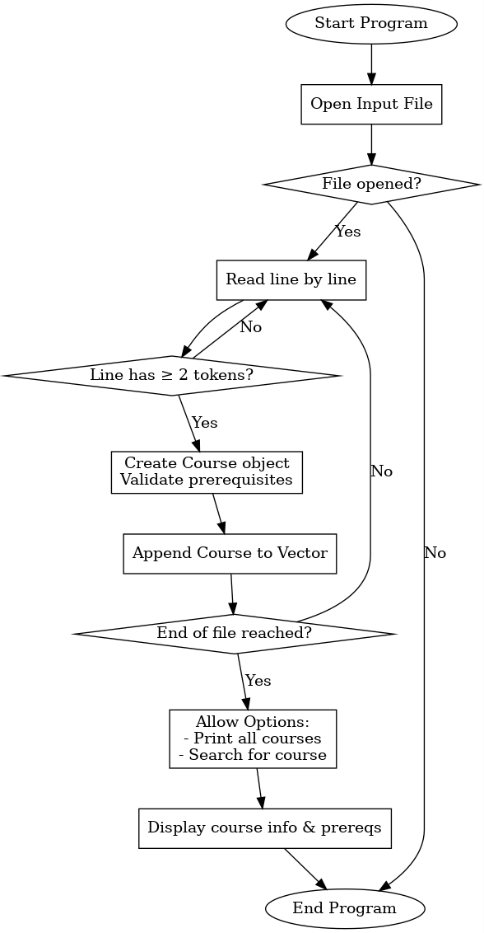
return

print "Error: Course not found in the data."

* The search function allows advisors to look up a single course by its number. It traverses the vector one course at a time (linear search). For a small dataset, this is fast and efficient. If the course is found, it prints all details, including prerequisites. If not, it gives a clear error message.

# **Flowchart**

The following flowchart illustrates the overall logic of the program. It shows how the system opens the course data file, validates each line, creates course objects, stores them in the vector, and then allows advisors to either print all courses or search for a specific course. This visual complements the pseudocode by making the step-by-step flow clearer and easier to follow.



**Figure 1: Flowchart of the course data program**

# **Runtime Analysis**

* **Loading courses**

O(n), where *n* is the number of courses. Each line is read once and processed.

* **Validating prerequisites**

O(n·m), where *m* is the average number of prerequisites. This is because each prerequisite must be checked against the set of valid course numbers.

* **Appending to vector**

Amortized O(1) for each course.

* **Printing all courses**

O(n), since each course must be printed once.

* **Searching for a course**

O(n), because the vector is traversed until a match is found.

* Even though searching and printing are linear, this is not a problem for ABC University because the number of Computer Science courses is relatively small (dozens, not thousands). The vector is therefore the right data structure: it balances simplicity, speed, and flexibility.

# **Conclusion**

This pseudocode offers a comprehensive design for how ABCU’s advisors could access course information. The program opens the file, verifies that each line has a valid format, creates course objects, and stores them in a vector. Once the vector is filled, the advisor can either print all courses or search for a specific course to view its details and prerequisites.

Working on this milestone gave me hands-on practice with **file validation, data structure selection, and thinking about runtime efficiency**. It also highlighted the importance of planning before coding. With this pseudocode complete, I now have a clear roadmap for implementing the full program in C++ in Project One.