Project One

Jorge Fernando Moreno Jacob

Southern New Hampshire University

CS 300: DSA Analysis and Design

Professor Saba Jamalian

December 2, 2025

Pseudocode: Vector

Step 1: Load and Validate the File

```
Function LoadCourses()
    Ask user for the file name
    Try to open the file
    If file can’t be opened
        Print "Error: could not open file"
        Stop
    End If
    Create an empty vector called courses
    While there are still lines to read in the file
        Read the line
        Split the line by commas
        If the line has fewer than 2 items
            Print "Error: not enough fields"
            Skip to the next line
        End If
        Create a new Course object
        Set course.courseNumber to the first item
        Set course.courseTitle to the second item
        If there are more items
            Store the rest as course.prerequisites
        End If
```

```
        Add the new Course to the courses vector
    End While
    Close the file
    For each course in courses
        For each prerequisite in that course's prerequisites
            Check if that prerequisite exists as a courseNumber in courses
            If not
                Print an error message about the missing prerequisite course
            End If
        End For
    End For
    Return courses
End Function
```

Step 2: Course Structure

Each Course object has:

- courseNumber
- courseTitle
- prerequisites (a list of other courseNumbers)

All Course objects are stored in the vector courses.

Step 3: Search and Print Course Information

```
Function FindCourse(courseNumber)
```

```
    For each course in courses
        If course.courseNumber matches courseNumber
            Return course
        End If
    End For
    Return "not found"
End Function

Function PrintCourse(courseNumber)
    Set course = FindCourse(courseNumber)
    If course is "not found"
        Print "Course not found"
    Else
        Print the course number and title
        If the course has prerequisites
            Print "Prerequisites:" followed by the list
        Else
            Print "No prerequisites"
        End If
    End If
End Function

Function PrintAllCoursesSorted()
```

```
    // Assume courses is the vector that already holds all Course objects
    If courses is empty
        Print "No courses loaded."
        Return
    End If
    Create a list called sortedCourses and copy all items from courses into it
    // Sort the list by courseNumber in ascending (alphanumeric) order
    Sort sortedCourses by courseNumber from lowest to highest
    // Print each course in the sorted list
    For each course in sortedCourses
        Print course.courseNumber + " - " + course.courseTitle
        If course has prerequisites
            Print "Prerequisites: " followed by the list of prerequisite courseNumbers
        Else
            Print "Prerequisites: None"
        End If
    End For
End Function
```

Step 4: Main Program (Menu)

```
Start
    Create an empty vector called courses
    Set choice = 0
```

```
    While choice is not 9
        Print "Menu:"
        Print " 1. Load Courses"
        Print " 2. Display All Courses (sorted)"
        Print " 3. Find a Course"
        Print " 9. Exit"
        Print "Enter choice:"
        Read choice
        If choice == 1
            Set courses = LoadCourses()
        Else if choice == 2
            Call PrintAllCoursesSorted()
        Else if choice == 3
            Ask user for a course number
            Read courseNumber
            Call PrintCourse(courseNumber)
        Else if choice == 9
            Print "Goodbye."
        Else
            Print "Invalid option. Please try again."
        End If
    End While
End
```

Pseudocode: Hash Table

Step 1: Load and Validate the File

```
Start
    Ask user for the course data file name
    Try to open the file
    If file cannot be opened
        Print "Error: could not open file"
        Stop
    End If
    Create an empty list called allCourseLines
    // each item will hold courseNumber, courseTitle, and prereq list
    While there are still lines to read in the file
        Read the line
        Trim spaces
        If the line is empty
            Continue to next line
        End If
        Split the line by commas into a list called pieces
        If number of pieces is fewer than 2
            Print "Error: missing course number or title"
            Continue to next line
        End If
```

```
    Create a temporary record tempRecord
    Set tempRecord.courseNumber = first piece
    Set tempRecord.courseTitle = second piece
    Set tempRecord.prereqList = all remaining pieces after index 1
    Add tempRecord to allCourseLines
End While
Close the file
// Validate that every prerequisite exists as a course in the file
For each record in allCourseLines
    For each prereq in record.prereqList
        Set foundMatch = FALSE
        For each otherRecord in allCourseLines
            If otherRecord.courseNumber == prereq
                Set foundMatch = TRUE
                Break out of this inner loop
            End If
        End For
        If foundMatch == FALSE
            Print "Error: prerequisite " + prereq + " does not exist as a course in the file"
            Stop
        End If
    End For
End For
```

End

Step 2: Create Course Objects and Store Them in the Hash Table

Start

Define a Course structure with:

courseNumber

courseTitle

prerequisites (a list of course numbers)

Create an empty hash table called courseTable

For each record in allCourseLines

Create a new Course object called newCourse

Set newCourse.courseNumber = record.courseNumber

Set newCourse.courseTitle = record.courseTitle

Set newCourse.prerequisites = record.prereqList

Compute a hash index from newCourse.courseNumber

Insert newCourse into courseTable at that index

// if there is already a course there, add to the linked list (chaining)

End For

End

Step 3: Search and Print Course Information

Function FindCourse(courseNumber)

```
    Compute hash index from courseNumber
    Set current = first node in the bucket at that index
    While current is not null
        If current.courseNumber == courseNumber
            Return current.Course
        End If
        Move current to next node in the chain
    End While
    Return "not found"
End Function
```

Function PrintAllCourses()

```
    For each bucket in courseTable
        Set current = first node in this bucket
        While current is not null
            Print current.courseNumber + " - " + current.courseTitle
            If the course has prerequisites
                Print "Prerequisites:" followed by each prereq in the list
            Else
                Print "Prerequisites: None"
            End If
```

```
        Move current to next node in the chain
      End While
    End For
End Function
```

Function PrintSingleCourse()

```
    Ask user for a course number to look up
    Read input into searchNumber
    Set foundCourse = FindCourse(searchNumber)
    If foundCourse is "not found"
      Print "Course not found."
    Else
      Print "Course Number: " + foundCourse.courseNumber
      Print "Title: " + foundCourse.courseTitle
      If foundCourse.prerequisites list is empty
        Print "Prerequisites: None"
      Else
        Print "Prerequisites:" and list all prereqs on one line
      End If
    End If
End Function
```

Pseudocode: Binary Search Tree
Step 1: Load and Validate Course File

```
Start

    Ask user for the course file name
    Try to open the file

    IF the file cannot be opened THEN
        Display "Error: could not open course file."
        Stop the program
    END IF

    Create an empty list called allCourseLines
        // each item will hold: raw line text + split tokens

    WHILE there are still lines to read in the file DO
        Read the next line as a string
        Split the line by commas into a list called tokens

        // ---- Format check A: at least two fields ----
        IF number of tokens < 2 THEN
            Display "Format error: not enough fields on this line."
            Display the line that caused the error
            Stop the program
        END IF
```

```
    // Save this parsed line so we can do prereq checks later
    Create a record tempRecord
    Set tempRecord.courseNumber = tokens[0]
    Set tempRecord.courseTitle  = tokens[1]
    Set tempRecord.prereqList  = any remaining tokens after index 1
    Add tempRecord to allCourseLines

END WHILE

Close the file

// ---- Format check B: every prerequisite exists as a course ----
FOR each record in allCourseLines DO

    FOR each prereq in record.prereqList DO

        Set foundMatch = FALSE

        FOR each otherRecord in allCourseLines DO
            IF otherRecord.courseNumber == prereq THEN
                Set foundMatch = TRUE
                BREAK out of this inner loop
```

```
            END IF
        END FOR

        IF foundMatch == FALSE THEN
            Display "Format error: prerequisite " + prereq +
                " does not exist as a course in the file."
            Stop the program
        END IF

    END FOR

END FOR

// If we reach this point, the file is valid and ready to load

End
```

Create Course Objects and Store in Binary Search Tree

```
Start

    Define a Course struct with:
        - courseNumber
        - courseTitle
```

- listOfPrereqs

Create an empty BinarySearchTree called courseTree

FOR each record in allCourseLines DO

Create a new Course object called newCourse

Set newCourse.courseNumber = record.courseNumber

Set newCourse.courseTitle = record.courseTitle

Set newCourse.listOfPrereqs = record.prereqList

Insert newCourse into courseTree

// Use the courseNumber as the key when inserting

END FOR

End

Print Course Information from Binary Search Tree

Start

// To print all courses in sorted order:

```
    Call InOrderTraversal(courseTree.root)

End


Procedure InOrderTraversal(node)

    IF node is not null THEN
        InOrderTraversal(node.left)

        Display node.course.courseNumber + " - " + node.course.courseTitle

        InOrderTraversal(node.right)
    END IF

End Procedure

Start

    Ask user for a course number to look up
    Read user input into searchNumber

    Search the binary search tree for searchNumber
```

```
    Store the result in foundCourse

    IF foundCourse is empty THEN
        Display "Course not found."
    ELSE
        Display "Course Number: " + foundCourse.courseNumber
        Display "Title: " + foundCourse.courseTitle

        IF foundCourse.listOfPrereqs is empty THEN
            Display "Prerequisites: None"
        ELSE
            Display "Prerequisites: "

            FOR each prereq in foundCourse.listOfPrereqs DO
                Display prereq (all on one line, separated by commas)
            END FOR
        END IF
    END IF

End
```

Runtime Analysis

This runtime analysis evaluates the worst-case running time for reading the file, parsing each line, and creating Course objects for the three data structures: vector, hash table, and binary search tree. As required, the analysis does not include menu logic or search/print operations.

Vector

When using a vector, the program performs the following steps:

1. Open the file.
2. For each of the *n* lines:
    - Read the line
    - Split the line into fields
    - Create a Course object
    - Insert the Course object into the vector using push_back

Example breakdown:

Operation	**Cost**	**Repetitions**	**Total Cost**
Open file	1	1	1
Read a line	1	n	n
Split the line into fields	1	n	n
Create a Course object	1	n	n
Insert Course into vector	1	n	n

Total cost: $1 + 4n \rightarrow O(n)$

Worst-case runtime: O(n)

Hash Table

The hash table follows the same initial steps:

1. Open the file
2. Read and parse each of the *n* lines
3. Create a Course object
4. Insert the object into the hash table using a hash function

Inserting into a hash table has different cases:

- Average case**:** Constant time O(1) for hashing and inserting
- Worst case**:** All values collide into the same bucket, creating a long linked list

Worst-case total:

- Insert one course: O(n)
- Insert all *n* courses: $O(n^2)$

Worst-case runtime: $O(n^2)$

Average expected runtime: O(n) (assuming good hash distribution)

Binary Search Tree

The steps for the BST are:

1. Open the file
2. Read and parse each line
3. Create a Course object
4. Insert it into the BST based on courseNumber

BST insert performance depends on tree height:

- Balanced BST**:** Insert = O(log n), total load = O(n log n)
- Degenerate BST (becomes a linked list): Insert = O(n), total load = $O(n^2)$

Worst-case runtime: $O(n^2)$

Expected runtime if reasonably balanced: O(n log n)

Advantages and Disadvantages of Each Data Structure
Vector

Advantages

- Very easy to implement
- Stores data contiguously in memory, making iteration fast
- Good for operations that access all courses

Disadvantages

- Searching for a specific course is O(n)
- Printing all courses in alphanumeric order requires sorting (O(n log n))
- Insertions in the middle are expensive because elements must shift

Hash Table

Advantages

- Extremely fast lookups in average case (O(1))
- Ideal for frequent searches by courseNumber
- Efficient insertion and retrieval

Disadvantages

- Does not maintain sorted order
- To print all courses alphabetically, an extra sort step is required (O(n log n))
- Collisions may degrade performance to O(n) per operation
- Uses more memory due to buckets and pointers

Binary Search Tree

Advantages

- Searching for a course is O(log n) when tree is balanced
- In-order traversal naturally prints the courses in alphanumeric order with no sorting required
- Efficient for both searching and ordered output
- Structure reflects hierarchical course relationships

Disadvantages

- More complex to implement compared to vectors and hash tables
- If the tree becomes unbalanced, worst-case performance becomes O(n)
- Removing nodes requires careful handling

Recommended Data Structure

The advisors at ABCU require the program to:

1. Print a list of all Computer Science courses in alphanumeric order, and
2. Display a course's title and prerequisites when searching for a specific course.

While the vector and hash table are effective in certain cases, the binary search tree best satisfies both requirements:

- It supports efficient searching for a single course
- It prints courses in sorted order directly through an in-order traversal
- It avoids the extra sorting step required by vectors and hash tables

The Binary Search Tree (BST) is the best data structure for this advising program because it

provides efficient search capabilities and outputs the list of courses naturally in alphanumeric order, fully matching the advisors' needs.