**ABCU Advising Program**

**Pseudocode & Runtime Analysis**

**Course Object**

STRUCT Course: number (string), title (string), prereqs (list<string>)

**Main Menu**

* LOOP: print options [1=Load, 2=Print All (sorted), 3=Print One, 9=Exit]
* READ choice
* IF choice==1: call LoadData()
* ELSE IF choice==2: call PrintAll()
* ELSE IF choice==3: READ courseNum; call PrintOne(courseNum)
* ELSE IF choice==9: BREAK
* ELSE: print 'Invalid choice'

**Vector Version**

LoadData\_Vector(file):

OPEN file

FOR each line in file:

tokens = split(line, ',')

IF len(tokens) < 2: report format error; CONTINUE

c.number = trim(tokens[0]); c.title = trim(tokens[1]);

c.prereqs = tokens[2..] (trim each)

PUSH\_BACK vectorCourses, c

CLOSE file

PrintAll\_Vector():

SORT vectorCourses by c.number (ascending, alphanumeric)

FOR c in vectorCourses: PRINT c.number, c.title

PrintOne\_Vector(courseNum):

FOR c in vectorCourses: IF c.number==courseNum:

PRINT c.number, c.title, 'Prereqs:', join(c.prereqs); RETURN

PRINT 'Course not found'

**Hash Table Version (chaining)**

LoadData\_Hash(file):

OPEN file

FOR each line: parse into Course c (as above)

INSERT hashmap[c.number] = c (create list for collisions)

CLOSE file

PrintAll\_Hash():

temp = VALUES(hashmap) // gather all courses

SORT temp by c.number; FOR c in temp: PRINT c.number, c.title

PrintOne\_Hash(courseNum):

IF courseNum in hashmap:

c = hashmap[courseNum]; PRINT c.number, c.title, 'Prereqs:', join(c.prereqs)

ELSE PRINT 'Course not found'

**Binary Search Tree Version (key = course number)**

LoadData\_BST(file):

OPEN file

FOR each line: parse into Course c

INSERT bst with key=c.number, value=c

CLOSE file

PrintAll\_BST():

IN\_ORDER\_TRAVERSE(bst.root): for each node print node.course.number, node.course.title

PrintOne\_BST(courseNum):

node = BST\_SEARCH(bst.root, courseNum)

IF node != NULL: PRINT node.course.number, node.course.title, 'Prereqs:', join(node.course.prereqs)

ELSE PRINT 'Course not found'

**Validation During Load**

* - Ensure each line has at least 2 tokens (course number, title).
* - Collect all course numbers in a set S during first pass; after load, for each course prereq p, verify p in S; report any missing as format errors.

**Runtime & Memory Analysis (worst-case, n = number of courses)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Structure | Load (Parse + Insert) | Find One | Print All (sorted) | Memory |
| Vector | O(n) parse + O(1) append → O(n) | O(n) | O(n log n) sort + O(n) print → O(n log n) | O(n) |
| Hash Table | O(n) avg (O(1) insert); worst O(n^2) with collisions | Avg O(1), worst O(n) | Gather O(n) + sort O(n log n) → O(n log n) | O(n) + buckets |
| BST (unbalanced) | Avg O(n log n), worst O(n^2) | Avg O(log n), worst O(n) | In-order O(n) (already sorted by key) | O(n) |

**Advantages & Disadvantages**

* **Vector:**
* Advantage - Simple and memory efficient, fast sequential print.
* Disadvantage - Linear search, must sort for ordered output.
* **Hash Table:**
* Advantage – Constant time average lookup for a course, good for frequent searches.
* Disadvantage - Needs extra space, still must sort to print all in order.
* **BST:**
* Advantage - Naturally ordered by course number, in order traversal prints in O(n), good search performance.
* Disadvantage – Be sure to prevent tree from coming unbalanced, since that can cause performance drop in the worst case.

**Recommendation**

Use a balanced BST (e.g., Red-Black/AVL) keyed by course number. It provides O(log n) inserts/searches, and prints the entire catalog in order with a single O(n) traversal, meeting both advising requirements efficiently. If a library balanced BST is unavailable, a hash table is a good backup for fast lookups, paired with sorting a temporary list for the “print all” feature.