PROGRAMMING ASSIGNMENT 2

Topics: AVL Trees, Black anmd Red Trees
Dynamic Memory Allocation, File I/O

Programming Language: C++11
Due Date: **Friday, 19/11/2025 (23:59:59)** over **https://submit.cs.hacettepe.edu.tr**

Introduction and Background

H.U. Department of Artifact Management (H.U.D.A.M.) has initiated an ambitious digital modernization effort aimed at preserving, analyzing, and distributing the country's most valuable research artifacts. Their vision is to build an intelligent Artifact Stewardship System capable of tracking ownership, regulating research access, and coordinating artifact circulation across Turkey's major academic institutions.

To achieve this, H.U.D.A.M. requires a fully functional simulation—a working prototype that demonstrates how artifacts and researchers interact inside this new system. And naturally, the department has turned to the brightest interns of Hacettepe University to make it happen.

In this simulation, artifacts of varying rarity, research importance, and historical significance will be cataloged, evaluated, and assigned to specialized researchers. Each artifact must be carefully indexed and balanced within the system, while researchers must be tracked according to their specialization fields, available capacity, and active assignments.

Once approved, researchers may request new artifacts for study, return completed ones, or temporarily lose access if they exceed capacity or leave the department. The system must also support advanced queries, generate detailed reports, and maintain strict consistency between both artifact and researcher registries.

The executives have given you, the interns, a demanding challenge: implement the core mechanisms of a real artifact stewardship environment, including:

- Balanced cataloging and retrieval of artifacts,
- Researcher indexing and assignment based on specialization and capacity,
- Consistent transfer of artifacts between researchers and storage, and
- Accurate reporting and comparison of system structures.

The H.U.D.A.M. leadership is counting on your implementation to demonstrate how interconnected data structures—AVL Trees and Red-Black Trees—can power a rule-based artifact management system that mirrors real academic research workflows.

The future of this modernization project, and the preservation of invaluable artifacts, rests on your code.

1 The Artifact Stewardship System

1.1 The Artifact Catalog (AVL Tree)

The Artifact Catalog is the core of the stewardship process. This structure stores every artifact currently managed by H.U.D.A.M., ensuring that all items can be located, updated, and verified efficiently. To support fast lookup and guaranteed worst-case performance, the catalog is implemented as an AVL Tree, where each node represents one artifact.

Each artifact in the catalog stores:

Every train stores:

- A unique artifacct ID
- A name,
- A rarity level (1–10),
- A research value, and
- An assignedTo field indicating the researcher currently responsible for it (or -1 if unassigned).

Because of the AVL Tree's strict balancing rules, insertions, deletions, and searches all operate in predictable logarithmic time. This is essential for maintaining responsiveness as the artifact repository grows.

Artifacts remain unassigned until claimed by a researcher. When a researcher is removed from the system, all artifacts assigned to them are immediately reset to unassigned.

1.2 The Researcher Directory (Red-Black Tree)

The Researcher Directory tracks all accredited specialists who may access or study artifacts. To support highly dynamic researcher turnover—where experts may be hired, reassigned, or removed frequently—this directory is implemented using a Red-Black Tree, which provides efficient insertion and deletion while maintaining overall balance.

Each researcher entry contains:

- Full name,
- A specialization field,
- A maximum assignment capacity, and
- A list of artifacts they are currently supervising

The Red-Black Tree ensures that researcher records remain balanced enough to support high-frequency operations. This is vital for large research programs where staffing changes occur daily.

Researchers cannot exceed their assignment capacity. Attempting to assign an additional artifact beyond this limit must be rejected by the system.

1.3 Artifact–Researcher Assignment System

Assignment is the central interaction between the AVL Catalog and the RBT Directory. When a researcher requests an artifact:

- The system checks whether the researcher exists (RBT).
- It checks whether the artifact exists and is unassigned (AVL).
- It checks whether the researcher has enough capacity.
- If valid, the artifact's assignedTo field is updated, and the researcher's assignment list is modified accordingly.

Returns work in the opposite direction: the system verifies that the artifact is indeed assigned to that researcher before unassigning it and updating all related records.

When a researcher is removed from the directory:

- Every artifact assigned to them is automatically set to unassigned,
- The researcher entry is deleted via Red-Black deletion rules,
- No dangling assignments remain.

1.4 The ArtifactManager (Controller)

The ArtifactManager class functions as the command executor and system controller. It manages:

- The AVL-based Artifact Catalog,
- The RBT-based Researcher Directory,
- Cross-structure synchronization,
- Command parsing,

1.5 Summary of System Flow

1. **Artifacts added** → Inserted into AVL Tree.
2. **Researchers hired** → Inserted into Red-Black Directory.
3. Requests → Researcher requests an artifact; validation and assignment occur.
4. Returns → Artifacts are returned and marked unassigned.
5. Deletions

2 Commands and Simulation Rules

This section describes all available commands, their expected formats, and the historical-archive rules governing how the system behaves. All commands are provided through an **input file**, which the program reads and processes sequentially, simulating the day-to-day operation of the Hacettepe Historical Archives.

2.1 Command List

Command	Description
ADD_ARTIFACT <id> <name> <rarity> <value>	Registers a new historical artifact (e.g., manuscript, inscription, relic) in the AVL-based Artifact Catalog. Artifacts begin unassigned and are stored in ascending order of artifactID.
REMOVE_ARTIFACT <id>	Removes the artifact with the given ID from the catalog. If it is currently assigned to a scholar, the assignment must be revoked before removal.
HIRE_RESEARCHER <name> <capacity>	Admits a new scholar into the Historical Research Directory (Red-Black Tree), capable of supervising up to capacity artifacts.
FIRE_RESEARCHER <name>	Removes the scholar with the given ID from the directory and automatically unassigns all artifacts they were supervising.
REQUEST <researcherID> <artifactID>	A scholar requests permission to study an artifact. The system verifies existence, capacity, and assignment rules, then links the artifact to the scholar if valid.
RETURN <researcherID> <artifactID>	Returns an artifact previously assigned to the scholar, marking it as unassigned in the catalog.
RETURN_ALL <researcherID>	Returns all artifacts previously assigned to the scholar, marking it as unassigned in the catalog.
RESEARCHER_LOAD <id>	Prints how many artifacts the given scholar is currently supervising.
MATCH_RARITY <minRarity>	Searches the catalog and prints all artifacts with rarity at least minRarity that are either assigned to or suitable for scholars.
PRINT_UNASSIGNED	Prints all artifacts currently not assigned to any scholar.
PRINT_STATS	Prints a summary of total artifacts, total scholars, average artifact rarity, and average researcher load. Scholars should be printed sorted according to name using pre-order traversal. Post order traversal should be used for artifacts
CLEAR	Clears all data structures and resets the entire simulation.

2.2 Artifact Catalog Rules (AVL)

1. The Artifact Catalog is implemented as an AVL Tree, sorted by **artifactID**.
2. All insertions, removals, and updates must preserve AVL balance through rotations.
3. Each artifact stores its:
   - artifactID,
   - name,
   - rarityLevel,

- `researchValue`,
- `assignedTo` (or `-1` if unassigned).

4. When an artifact is removed, any scholar supervising it must also be updated so that the artifact is removed from their assignment list.
5. Inorder traversal of the AVL tree must reflect all artifacts accurately and in sorted order.

2.3 Researcher Directory Rules (Red-Black Tree)

1. Scholars are stored in a Red-Black Tree and sorted by **researcherID**.
2. All insertions and deletions must preserve Red-Black properties:
   - Root is always black,
   - Red nodes cannot have red children,
   - All root-to-leaf paths must have equal black-height.
3. Each researcher contains:
   - `researcherID`,
   - `fullName`,
   - `specialization`,
   - `capacity`,
   - A dynamic list of supervised artifact IDs.
4. Removing a scholar (`FIRE_RESEARCHER`) must:
   - Unassign all artifacts they supervise,
   - Properly delete the RBT node using the Red-Black delete fix-up.

2.4 Assignment Rules (Artifacts Scholars)

1. A scholar may supervise at most `capacity` artifacts.
2. An artifact may only be supervised by **one** scholar at a time.
3. `REQUEST` must validate:
   - The artifact exists and is unassigned,
   - The scholar exists and is not at capacity.
4. `RETURN` must validate:
   - Both entities exist,
   - The artifact is currently assigned to that scholar.
5. After succesful assignment,value of artifact should be increased by rarity * numberofsuccesful assignments

6. All assignments must be reflected consistently in **both** structures.

2.5 Output and Reporting Rules

1. All printed output must match the exact formatting shown in the assignment description.
2. `PRINT_ARTIFACTS_INORDER`, `PRINT_UNASSIGNED`, and `PRINT_RESEARCHERS_INORDER` print structured listings from the two trees.
3. `PRINT_RBT_LEVELS` prints the researcher directory level-by-level, with node colors appended to researcher IDs.
4. `PRINT_STATS` outputs four lines summarizing archive-wide statistics.
5. Any invalid command must produce the message:

   `Error: Unknown command '<command>'.`
6. Execution stops at end-of-file.

Implementation Notes

- You are provided with header and source templates defining all classes and structures required for the simulation (i.e., Starter Code).
- All tree-based data structures (`AVLTree`, `RedBlackTree`) **must be implemented manually**. Use of STL containers such as `std::vector`, `std::list`, `std::map`, etc. is **not allowed**.
- You must complete all functions marked with `// TODO` in the provided files without changing the specified class or function names. You may add helper functions as needed.

3 Files: Input/Output

3.0.1 Input Format

The program reads a sequence of archive-management commands from an input file supplied as a command-line argument. Each line in the file corresponds to a single operation within the Historical Research Stewardship System. Commands may reference artifact identifiers, researcher names, rarity levels, capacities, or thresholds. All command keywords are written in **UPPERCASE**, and all parameters must be separated by spaces.

`Example Input:`

```
ADD_ARTIFACT 501 HittiteTablet 8 1400
HIRE_RESEARCHER HaletCilingiroglu 4
REQUEST HaletCilingiroglu 501
RETURN HaletCilingiroglu 501
PRINT_UNASSIGNED
```

Blank lines and lines containing only whitespace should be ignored. Processing continues until the end of the input file is reached.

3.0.2 Output Format

All output must be written to standard output (`stdout`) using **`std::cout`**. Do not use C-style output functions such as `printf()` or `fprintf()`.

Each command produces a specific message or structured listing. These outputs form the official administrative log of the archive's operations and will be verified through automated testing. **Any deviation in text, spacing, punctuation, capitalization, or formatting may result in point deductions.**

`Example Outputs (Not connected to example inputs above):`

```
Artifact 501 added.
Researcher HaletCilingiroglu hired.
Artifact 501 assigned to HaletCilingiroglu.
Artifact 501 returned by HaletCilingiroglu.
RBT LEVEL ORDER:
HaletCilingirogluB
Error: Artifact 999 not found.
Error: Researcher Tarkan is at full capacity.
Exported to artifacts.txt and researchers.txt.
Error: Unknown command 'EXCAVATE_ARTIFACT'
```

Structured print commands such as `PRINT_UNASSIGNED`, `PRINT_STATS`, or tree traversals must follow the detailed formatting rules provided in the assignment description.

The use of additional debug output is strictly forbidden. Only the specified messages may be printed.

Must-Use Starter Code

You MUST use the provided starter code. All classes (`Artifact`, `Researcher`, `AVLTree`, `RedBlackTree`, `ArtifactManager`, etc.) must appear exactly as given and must be placed directly inside your **zip** archive.

You are allowed to add helper methods, but you may not rename or remove any existing classes, functions, or member variables marked in the template. Automated grading depends on these specifications.

Build and Run Configuration

Your code will be compiled in an environment similar to the following (note that instead of `main.cpp`, our test files will be used):

```
$ g++ -std=c++11 *.cpp *.h -o HUArchive
```

You may optionally use the provided `Makefile` or `CMakeLists.txt`:

```
$ make all
```

or using CMake:

```
$ mkdir HUArchive_build
$ cmake -S . -B HUArchive_build/
$ make -C HUArchive_build/
```

After compilation, you can run your program (with test input redirected) using:

```
$ ./bin/HUArchive < input.txt
```

Grading Policy

- **No memory leaks and runtime errors: 10%**
  - No memory leaks: 5%
  - No memory errors / invalid reads: 5%
- **Correct implementation of core data structures: 45%**
  - Correct implementation of the AVL Tree (artifact catalog): 15%
  - Correct implementation of the Red-Black Tree (scholar directory): 15%
  - Correct implementation of traversal rules (postorder for artifacts, preorder for researchers): 15%
- **Correct implementation of commands and system actions: 35%**
  - Correct execution of `ADD_ARTIFACT` / `REMOVE_ARTIFACT`: 5%
  - Correct execution of `HIRE_RESEARCHER` / `FIRE_RESEARCHER`: 6%
  - Correct execution of `REQUEST`, `RETURN`, `RETURN_ALL`: 10%
  - Correct execution of `MATCH_RARITY`: 5%
  - Enforcement of system consistency (capacity rules, uniqueness constraints, assignment consistency): 9%
- **Output tests: 10%**

  - Exact matching of required output format, spacing, and punctuation.

Note: You must receive a NON-ZERO grade from the assignment to have your submission accepted. Submissions graded as 0 will be treated as NO SUBMISSION.

Important Notes

- Do not miss the deadline: **Friday, 07/11/2025 (23:59:59)**.
- Keep backups of your work until grading is complete.
- Your submitted solution must be entirely your own work.
- All discussions must remain high-level; exchanging code or pseudocode is forbidden.
- Questions should be posted on Piazza: `https://piazza.com/hacettepe.edu.tr/fall2025/bbm203`.
- You MUST test your code on **Tur^{3}Bo Grader**: `https://test-grader.cs.hacettepe.edu.tr/` (This does *not* count as submission.)
- Submit your final work at: `https://submit.cs.hacettepe.edu.tr/` Submission format:
  - **b<studentID>.zip**
    - Artifact.cpp <FILE>
    - Artifact.h
    - Researcher.cpp
    - Researcher.h
    - AVLTree.cpp
    - AVLTree.h
    - RedBlackTree.cpp
    - RedBlackTree.h
    - ArtifactManager.cpp
    - ArtifactManager.h
    - main.cpp
- **You MUST use the starter code.** All files must be included exactly as instructed.
- Only submit **.cpp** and **.h** files — no binaries or object files.

Academic Integrity Policy

All work on this assignment must be done individually. You may discuss general concepts with classmates, but sharing code, pseudocode, or detailed solutions is strictly prohibited. Copying or adapting code from any online source (including AI/LLM tools such as ChatGPT, Gemini, or Copilot) is considered an academic integrity violation.

All submissions will be checked for similarity. Any submission that fails the similarity check will NOT be graded and will be reported to the ethics committee. Violations may result in disciplinary action, including suspension. Do not copy code from any source.