National Teachers College
School of Arts, Science &Technology
629 J. Nepomuceno, Quiapo,Manila

“Smart pH Monitoring System for Water Quality Using IoT”

Submitted to:
Justin Neypes

Table of Contents:

I. Introduction

Project Overview & UN SDG Target

The United Nations Sustainable Development Goal 6 (SDG 6) on *Clean Water and Sanitation* claims that access to clean water and sanitation is indispensable for human life in this modern world. Good water quality is not only a human health safety measure but also a factor supporting the sustainable development of a region. One of the most effective ways to monitor the water quality is through the pH measurement which provides information on the water's acidity or alkalinity and consequently helps to detect pollution. This project aims to create a pH tester for water using the C++ programming language with the features allowing the users to input the readings, analyze them and make a decision on the water's potability. Ultimately, the C++ Water pH Tester project is not just a technological exercise but also a step towards improving clean water and sanitation, which is the whole point of the SDG goals. This project communicates to the communities the simultaneous nature of science, technology and social responsibility, thus through the provision of a reliable and easy way for water quality testing, the public health will be protected and the use of better water practices in the communities will be encouraged.

Problem Statement (What real-world problem does the app solve?)

Insufficient availability of pure drinking water: More than 2 billion individuals all over the globe do not have access to clean drinking water, thus, they are susceptible to diseases caused by water.

Health risks associated with unclean water: Consumption of water leading to diarrhea accounts for more than 485,000 deaths annually mainly affecting children under five years of age.

Testing water is costly: The standard testing kits for water are priced from $20 to $50, which is beyond the financial reach of a lot of people in poor economies.

Unsafe pH values in drinking water: Water with a pH below 6.5 or above 8.5 can affect human health and corrode pipes, however many communities lack simple equipment to detect these levels.

II. Requirements & Analysis

Functional Requirements and Non-Functional Requirements (List of features, e.g., FR1, FR2)

Functional Requirements (FR)

- FR1 — Add Test: Accept pH (double) and color code (int) inputs, create a `WaterTest` record with a unique `id`, classify it, store it in all modules (array, stack, linked list, BST, graph node, hash table), and display classification + tip.
- FR2 — Undo Last Test: Pop the last `WaterTest` from `TestStack` and remove the corresponding record from the array (and notify user).
- FR3 — List All Tests (Array): Display every `WaterTest` stored in the central array in insertion/index order.
- FR4 — Show Recent Tests (Stack): Display stack contents (most recent first).
- FR5 — Sort Tests by pH: Sort central array by pH (bubble sort) and allow viewing of sorted results.
- FR6 — Linked List View: Insert tests at head and display/traverse the linked list.
- FR7 — BST View: Insert by pH into a BST and display tests via inorder traversal (ascending pH).
- FR8 — Search by ID (Linear): Linear search over the array for a given ID.
- FR9 — Search by ID (Binary): Binary search over array (assumes appropriate sort order by the searched key).
- FR10 — Graph View: Add node for each test and display adjacency lists; allow manual edge creation.
- FR11 — Hash Lookup: Insert and retrieve `WaterTest` by `id` using `unordered_map` (O(1) average).

Non-Functional Requirements (NFR)

- NFR1 — Capacity Limit: System handles up to `MAX_TESTS = 60` test records (memory/array bounds).

- NFR2 — Responsiveness: Interactive CLI must respond immediately for operations on ≤60 items.
- NFR3 — Safety / Memory: All dynamic memory allocated by linked list and BST must be freed in destructors.
- NFR4 — Simplicity / Teachability: Implementations use simple, readable algorithms (bubble sort, arrays, pointer lists) for instructional clarity.
- NFR5 — Determinism: IDs are monotonically increasing and unique within a single run (`nextId`).

Data Requirements (Description of input data structure and size)

Input fields (per test)

- `pH` — `double` in range `0.0` to `14.0` (program prompts user; no enforced validation beyond prompt).
- `colorCode` — `int` with discrete values `1` (clear), `2` (slightly colored), `3` (very colored).
- `id` — `int` auto-assigned (`nextId` starting at 1).
- `status` — `string` derived from classification (`"Filtered"` or `"Not Filtered"`).

Structures & storage

- Central array: `WaterTest tests[MAX_TESTS]` (capacity 60). `testCount` tracks used entries.
- Stack history: fixed-size `TestStack.data[MAX_TESTS]` holding full `WaterTest` copies; `top` index.
- Linked list: dynamic `Node*` chain containing `WaterTest` copies.
- BST: dynamic `TreeNode*` nodes ordered by `pH`.
- Graph: `vector<WaterTest> nodes; vector<vector<int>> adj;` — nodes stored by index.

- Hash table: `unordered_map<int,WaterTest> ht;` keyed by `id`.

Size constraints

- Maximum of 60 active tests in array/stack/graph `nodes`. Linked list and BST will have up to 60 nodes. Hash table stores up to 60 entries. Adjacency lists can in worst case store $O(n^2)$ edges but the UI does not auto-create edges, so typical usage keeps edges sparse.

Complexity Analysis: Expected Time/Space complexity of the Core Algorithm (justify using Big O notation).

Add Test (create + propagate to all modules)

- *Time:* O(1) average per module insertion → overall O(1) amortized (array append O(1), stack push O(1), linked-list insert O(1), BST insert O(h) where h = tree height; worst-case h = O(n) but average for random pH is O(log n); graph addNode O(1); hash insert O(1) average). Because n is small and BST height is the only variable, we conservatively state O(h) worst-case and O(1) average.

- *Space:* storing one WaterTest in array + copies in other modules → O(1) additional space per insert; cumulative storage across modules is O(n).

Undo (stack pop + remove from array)

- *Time:* pop() O(1) + array removal requires shifting subsequent entries O(n) in worst case → O(n). Other modules are not removed in current implementation (so their stale copies remain), but a full undo with removals in all modules would require O(n) for array + O(n) or tree-removal complexities for BST.

- *Space:* no extra persistent space; O(1) auxiliary.

Display All (array traversal)

- *Time:* traverse array → O(n).

- *Space:* O(1) extra.

Display Stack (traverse stack array)

- *Time:* O(k) where k = top+1 ≤ n → O(n) worst-case.
- *Space:* O(1) extra.

Bubble Sort by pH

- *Time:* standard bubble sort → nested loops → $O(n^2)$ comparisons and swaps in worst and average case. With n ≤ 60, acceptable for pedagogy.
- *Space:* in-place swaps → O(1) extra.

Linked List Insert & Search

- *Insert at head:* O(1) time, O(1) space per insert (node allocation).
- *SearchById:* traverse list → O(n) time.
- *Space total:* O(n) nodes.

BST Insert & Inorder Display

- *Insert:* average O(log n) time if tree is balanced; worst-case O(n) (degenerate chain) because insertion compares pH down the tree. Current insertion has no balancing.
- *Inorder display:* O(n) time.
- *Space:* O(n) for nodes.

Graph addNode & addEdge & display

- *addNode:* push back into nodes and adj → O(1) amortized.
- *addEdge(u,v):* push into adjacency lists → O(1) per insertion.
- *display:* iterate all nodes and adjacency lists → O(n + m) where m = number of edges. In typical usage m is small; worst-case $m = O(n^2)$.

- *Space:* nodes O(n), adjacency lists O(n + m).

Hash Table Lookup / Insert / Erase

- *Time:* average O(1) for insert/lookup/erase; worst-case O(n) in degenerate hashing collisions but practically O(1).

- *Space:* O(n).

Linear Search & Binary Search (array)

- *Linear search:* O(n) time.

- *Binary search:* O(log n) time *only if* array is sorted by the search key (note: current bubble sort sorts by pH, while binary search as written compares id — requires array sorted by id to be correct).

- *Space:* O(1) for both.

Overall Space Complexity (system-wide)

- Each WaterTest is stored multiple times (array + stack + linked-list node + BST node + graph node + hash table entry). Per-test cumulative space is constant, so total memory is O(n). Adjacency lists add extra O(m) space for edges. With $n \leq 60$, memory usage is bounded and small.

Short final notes / caveats

- Because MAX_TESTS is a constant (60), many worst-case bounds are capped in practice, but algorithmic complexity still matters conceptually (e.g., bubble sort is $O(n^2)$).

- Binary search is correct only if the array is sorted by the search key; reconcile the sort-by-pH vs. binary-search-by-id mismatch to avoid incorrect results.

- To make undo fully consistent, each module needs a removeById() and the stack should store richer operation records (op type + payload) so a single undo can update array, list, BST, graph (edges), and hash table.

III. Design Specifications

Justification: Why was this specific DSA chosen for its role?

(Array)
The decision was made to use an array as it is the easiest of all methods to keep a number of water test records in a fixed-size system. The array allows quick access to the individual items through indexing which makes data display, sorting, and traversal very fast. It is also suitable for implementing basic searching algorithms like binary and linear search due to its sequential memory structure.

(Stack)
The Undo functionality was supported inherently by the Last-In-First-Out (LIFO) method of a stack, so it was decided to use that data structure. This data structure mirrors the operation of the undo feature in the real world by only allowing the removal of the most recent test. It makes sure that the users can undo the last entry without it affecting the previous data. The stack is also giving more structure to the program by allowing the controlled addition and removal of test records.

(String)
For holding legible data, such as water classification and comprehensive safety advice, strings were selected. By using human-readable communications rather than numerical codes, they improve the system's usability. Depending on the test outcome, the strings offer the flexibility of dynamic text. Besides that, they provide useful labels such as "Filtered" and "Not Filtered," which are ways to enhance the user's clarity.

(Sorting)
Sorting was the reason that led to the water test results being logically ordered, that is, starting from the lowest to it being the highest in terms of pH values. It is obvious that consumers will find water quality comparisons easier because of the structured data. Bubble Sort was the algorithm of choice because it is user-friendly and very effective in the case of small datasets, such as the 60 test records that this application will work with.

(Trees)
To present the water testing based on pH levels in a sorted hierarchical manner, the Binary Search Tree was picked. BSTs have better searching and sorting compared to linear structures since they allow and promote sorted traversal, which in turn, the system can display records from the lowest to the highest pH.

Implementation Details: How did you implement it (e.g., adjacency list for Graph, array for Heap)?

(Array)
All the test items that were ever tested have been recorded and kept in an array of max size called WaterTest tests[MAX_TESTS]. A variable of type integer is used to store the value of the new test and it is added to the next available free index. After that, the bubble sort is applied to this array in order to organize the tests considering their pH level.

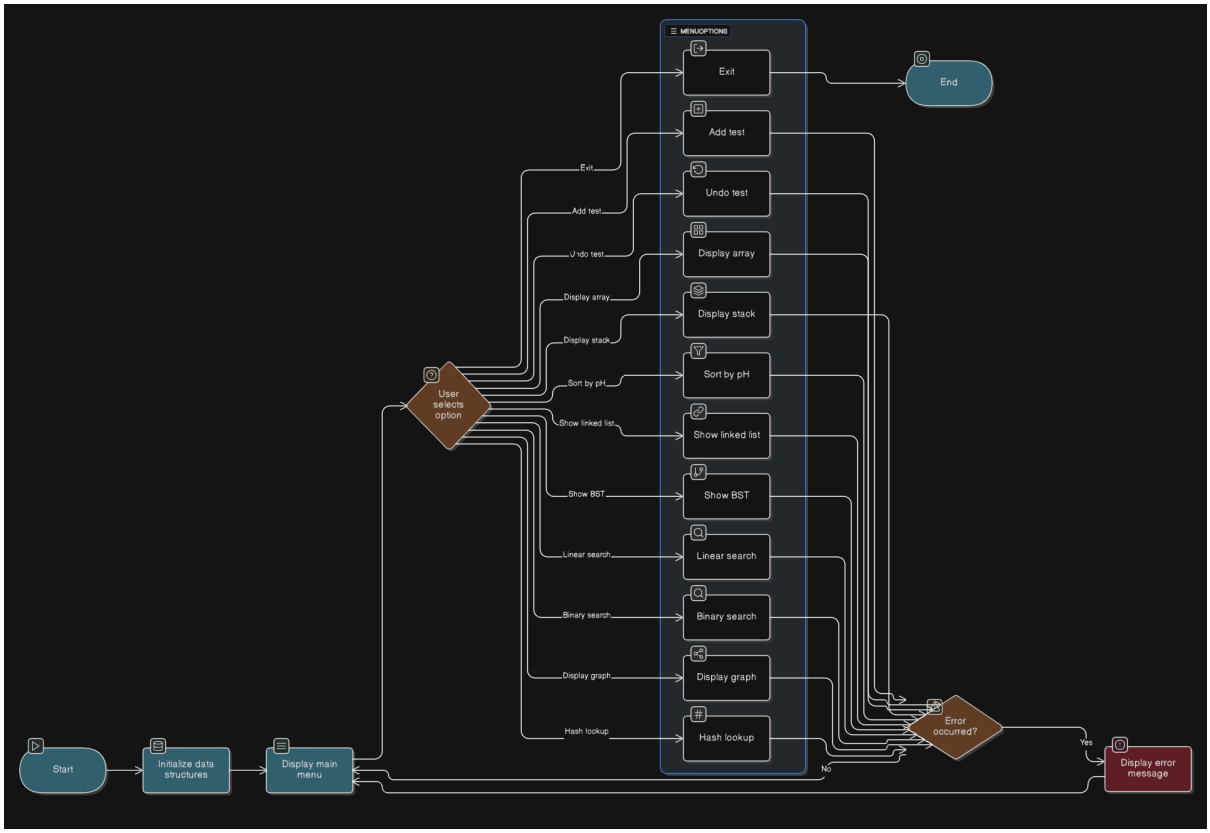
(Stack)
The custom stack TestStack structure had a stack that was provided with an array and a top index. The most recent WaterTest record is put in and the top is increased through the push() method. The pop() method decreases the top pointer while returning the most recent element.

(String)
The status field of each WaterTest contains the categorization results, which are recorded as C++ string objects. In case of different pH and color conditions, the text tips were kept in a 2D vector of strings. The strings describing the water quality are given back by the categorization function.

(Sorting)
To carry out the sorting, a bubbleSortByPH() method was made which took the WaterTest records array and the current number of tests as parameters. Using nested loops, the method compared neighboring items over and over again and swapped them if their pH values were not in order. The highest pH values will gradually "bubble up" to the end of the array, hence sorting the whole list.

(Trees)
The WaterTest and the pointers to the left and right descendants were kept in a TreeNode struct. The insert() method places new nodes by comparing the pH value with the current node. All the tests were displayed in ascending order of pH levels through inorder traversal.

Algorithm Flowchart: Include the Flowchart for the system's most complex function (the core algorithm using a Finals concept).

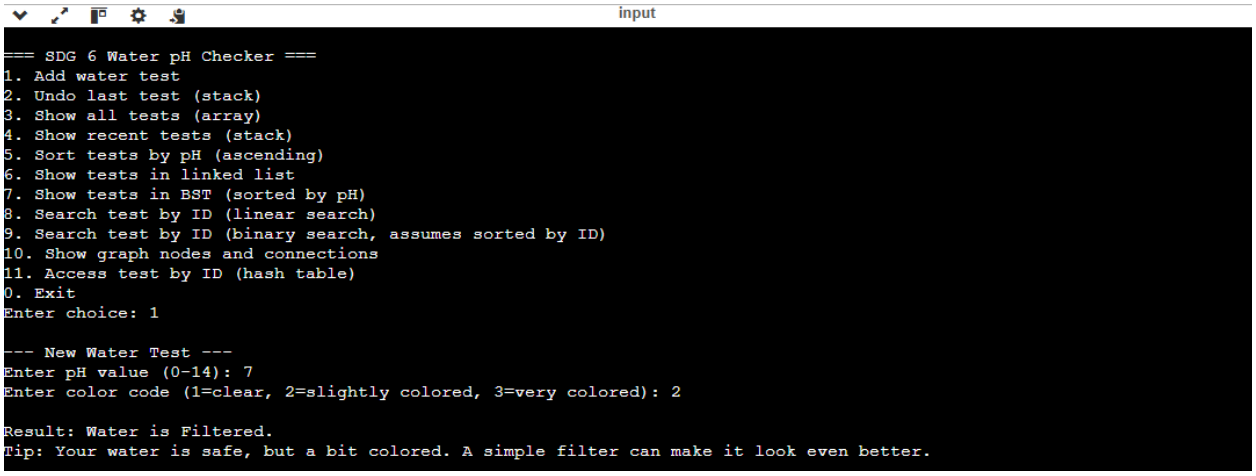


Module Breakdown: Define the custom C++ classes and how they interact.

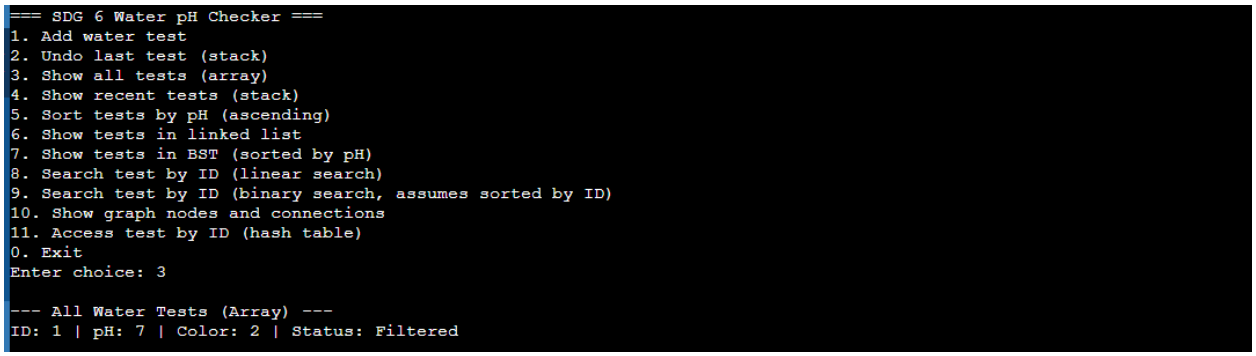
The whole system is composed of various separate C++ modules, each designed for a specific task, which together provide the processing and then the analyzing of the WaterTest data. The WaterTest struct, which is the key part of the software, holds the test's ID, pH level, color code and the condition of the test. Since every module gets and keeps a copy of this struct, it acts as the shared data unit. The TestStack operates on the principle of enabling the program to reverse its actions by pushing the new test on the top and popping the most recent one, thus serving as an undo tool with LIFO style. The LinkedList, while managing dynamic storage, takes care of quickly adding new tests at the top for O(1) insertion and also the BST does its part with simple sorted inorder navigation without sorting the array while pH tests get arranged. The Graph module symbolizes relations by means of adjacency lists and, despite being grounded on array indices that may become altered in the event of deletions, it still considers nodes as WaterTest objects in memory. Moreover, the HashTable (executed as unordered_map) not only gives rapid O(1) ID-based lookup but also employs the fixed-size array for sorting and searching.

IV. Testing and Result

Test Cases



Explanation: The user added one test with pH = 7 and colorCode = 1. The classifier returns "Filtered" and the program prints the matching tip from tipsMatrix



By enumerating every test's ID, pH level, color code, and status, this output provides a great convenience to the user since they can view all the saved results in a single spot. The array being the main storage unit, hence all the updates like additions, deletions (from undo), or sorting are reflected instantly when this option is used.

V. Conclusion and Contribution

Conclusion

The development of the Water pH Tester programmatically in C++ shows how technology could address global problems like water safety and public health. Users of such instruments must be no brainer since the problem of water quality monitoring for communities is still a global issue. The project, by creating an efficient and easy-to-use pH test, has already realized one of the targets under the United Nations Sustainable Development Goal 6 (SDG 6)—Clean Water and Sanitation. Therefore, people would quickly and accurately identify pH—a key indicator of water quality—so that they could decide whether to drink the water based on the knowledge of its condition. Moreover, the research that is being talked about shows that technology has the potential to significantly contribute to the adoption of sustainable practices and the increase of the environmental awareness level. It acts as a reminder for the users and the students to recognize the importance of water quality and to understand how even the most basic programming tools can support the earth's needs.

Individual Contributions (Detailed breakdown of each member's assigned module/class.)

Arandia, Jervy -

Cayetano, Christian James -

Ferrer, Alquin -

Nario, Angelo -

Swing, Jhon Seena -