1 . Class Description

Goals of RandomAccessSet

- Manages the collections of items (such as tasks, tags, or changes) with the efficiency and uniqueness of a set, but with an indexable behavior
- Will combine the properties of a set (unique element storage) but with the ability to access elements by their positon (array-like indexing).

How

- Combining std::set for set operations and std::vector for random access container OR
- Combining std::unordered_map to store the elements and their indices (helpful for membership checking) with std::vector of actual element storage and random access/ordering.

Restrictions

- No duplicates (A task or tag shouldn't be added twice)
- Predictable order for UI rendering
- Test if a tag already exists (membership)
- Access random access for efficient retrieval

Important Features

- Support normal set behavior: Meaning make sure we are testing membership, adding or removing elements but keeping uniqueness, iteration over elements
- Add Element
- Removing an element
- Access by index
- Iteration
- Membership test (checking if the set already contains an element)

High-level Functionality

- Hybrid Data structure of a set and a list that allows unique element storage with random accessing through indexing with order with the goal of avoiding duplicates, accessing elements by position, and specific order preservation for efficient iteration.

2 . Similar classes

When mentioning "similar" classes, to achieve the goal of RandomAccessSet, we want to focus on uniqueness and index access support. Thus, a combination of classes will be needed but the ones listed here can be combined in several ways.

Std::set

- Unique elements
- Ascending order sort

Std::unordered_set

- Unique elements
- No specific order (hash table format)

Std::vector

- Random access support with index (O(1))
- Maintains insertion order
- Accepts duplicates

Std::deque

- Random access support with index
- Allows duplicates

Std::map

- Key-value pairing for sorted ordering of the keys
- Unique keys

Std::unordered_map

- Key-value pairing for sorted ordering of the keys
- Unique Keys
- No specific ordering

3 . Key Functions

- Add(&element) -> returns bool
- Remove(&element) -> returns bool
- Get(index) -> returns element at specific index
- Contains(&element) -> returns bool
- Size() -> returns number of elements in set
- Clear()
- getIndexOf(&element) -> returns index of an element
- Iteration capability

    Std::vector<type_here>::iterator begin()

    Std::vector<type_here>::iterator end()

4 . Error Conditions

- Programmer Error

    Catch with insertions ASSERT:

    EX: Out of bounds indexing/deletion, invalid parameter argument types, adding invalid items (null elements)

- Resource limitations EXCEPTION THROW:

EX: failure to allocate memory, concurrency issues

- User error catching
  Returning a special condition or throwing exception:
  EX: Element search that doesn't exist

5 . Expected Challenges

- Personal biggest challenge of knowing where/how to start. I can see finish line but not sure how to begin to get there.
- What test development approach do we have to use?

6 . Other class projects

- Group1 Stroustrup Random and WeightedSet
  Why Random: Could help with TagManager and DynamicString with randomization
  Why WeightedSet: TagManager could use the weighted relationship approach for tagging priority levels

- Group2 Ritchie ActionMap
  Why: Would help to dynamically link user actions to specific functions

- Group9 Liskov
  Why: Could help with having a system for scheduling and triggering events (task deadlines, reminders) allows for efficient task prioritization