DynamicString Class

1) Description:

An interspersed mix of std::strings and functions containing a ToString() function that appends each of these together, calling the functions and making the results into strings.

High-Level Goal

The DynamicString module provides functionality for building and managing strings that can update themselves on demand. Instead of storing only fixed text, DynamicString can contain functions or lambdas that produce text at runtime. When asked for its current contents (via ToString()), the module repeats calls to these function segments to account for any updates.

Example usage in Project:

The To-Do list items could each contain a status level variable to track progress from an options list of “Not Started, In Progress, Pending Review, Complete.” The name of the task can be stored as a fixed string while the status is dynamic. DynamicString can process both and return a a string containing both the task name and the most up-to-date status as one string.

2) Similar Classes in the Standard Library Usage

- `std::string:` The core string storage we will use in combination with dynamic elements.
- `std::stringstream:` Useful for building strings.
- `std::function:` We will store functions returning std::string for dynamic segments.
- `std::vect or std::list:` Use to maintain an ordered collection

  `std::format:` Could help with formatting logic. (C++20)

3) Key Functions

- `Constructor Overloads`
  - `DynamicString()` – Default constructor (no segments).
  - `DynamicString(const std::string&)` – Initialize with one static segment.

- `DynamicString(std::function<std::string()>)` – Initialize with one dynamic segment.

➢ `ToString()`

- **`Signature:`** `std::string ToString() const`
- **`Behavior:`** Iterates over all segments—static and dynamic—and concatenates them. The dynamic segments are functions invoked each time, ensuring the returned string reflects the **current** state of those functions.

➢ `Append(const std::string&)`

- Adds a new static string segment at the end of the current set of segments.

➢ `Append(std::function<std::string()>)`

- Adds a new dynamic segment at the end.

➢ `Insert / Remove / Replace (Optional)`

- For advanced control, we may allow adding/removing segments at arbitrary indices.

➢ `Clear()`

- Removes all segments, resetting the DynamicString to an empty state.

4) Error Conditions

➢ Out-of-Range Access (if Insert/Remove by index is supported)

- Source: Programmer error.
- Handling: Could use assert or throw std::out_of_range to catch misuse.

➢ Empty or Invalid Function

- std::function can be empty, which throws std::bad_function_call when invoked.
- Source: Programmer error.
- Handling: We might rely on std::function's native exception or add a check before invoking it.

➢ Memory/Resource Issues

- If extremely large strings or many dynamic segments are appended, you may run into out-of-memory issues.
- Source: Potentially recoverable system-level issue.
- Handling: Typically results in std::bad_alloc from the runtime. We rely on standard library behavior here.

- User/Data-Dependent Errors
  - If a dynamic function depends on external data (e.g., user input) that can be invalid, that logic must handle its own errors. DynamicString simply invokes that function.

5) Expected Challenges

- Frequent Recalculation
  - On every call to `ToString()`, we re-invoke all dynamic segments. This could impact performance.
- Ordering & Manipulation
  - Deciding how to insert, remove, or reorder segments.
- Integration
  - Making `DynamicString` easily usable within the broader project. Making sure that updating, and displaying the dynamic text remains intuitive.