CS 3005: Programming in C++

Collections

Introduction

It is common to want to keep a collection of instances of items of the same type. This will allow to the same item to be reused, rather than create another item that is identically configured.

In particular, we will want to keep a collection of waveforms, a collection of envelopes, and a collection of instruments.

Our collections will require a unique name for each item in a collection. For example, “Cello” and “Violin” are different names we could use for two separate instruments. However, if we tried to store another “Violin” in the collection, it would overwrite the old item with the new one.

Assignment

In this assignment, you will create classes to represent collections of waveforms, envelopes, and instruments. These classes will be capable of storing and retrieving items. They will also allow for the collection to be iterated over.

Iterators

For each of our collections we will provide a `begin` and `end` method to allow for iteration over the collection. These methods will use the underlying iterators for the standard library class we are using for storage.

Programming Requirements

Create `library-waveform/Waveforms.{h,cpp}`

`Waveforms` Class

Data Members:

The `Waveforms` class should contain data members to track the following information. These data members should be `protected` or `private`. They are not allowed to be `public`.

- `std::map<std::string, std::shared_ptr<Waveform>>` a map of names to waveforms; Used to store all waveforms.

`public` Type Definitions:

- `typedef std::map<std::string, std::shared_ptr<Waveform>>::iterator iterator;`
- `typedef std::map<std::string, std::shared_ptr<Waveform>>::const_iterator const_iterator;`

`public` Methods:

- `Waveforms();` Default constructor, does not need to do anything.
- `virtual ~Waveforms();` Required, but empty.
- `void addWaveform(const std::string& name, std::shared_ptr<Waveform> waveform);` Stores `waveform` in the map, using `name` as the key.
- `std::shared_ptr<Waveform> getWaveform(const std::string& name);` If `name` exists in the map, returns the waveform associated with `name`. Otherwise, returns `nullptr`.
- `iterator begin();` Returns an iterator to the first item in the collection.
- `const_iterator begin() const;` Returns a constant iterator to the first item in the collection.
- `iterator end();` Returns an iterator to the end item in the collection. This is the item that doesn’t exist.
- `const_iterator end() const;` Returns a constant iterator to the end item in the collection. This is the item that doesn’t exist.

Update `library-waveform/Makefile`

Add `Waveforms.{h,cpp}` in the appropriate places to add them to the library and install the header file.

Create `library-envelope/Envelopes.{h,cpp}`

`Envelopes` Class

Data Members:

The `Envelopes` class should contain data members to track the following information. These data members should be `protected` or `private`. They are not allowed to be `public`.

- `std::map<std::string, std::shared_ptr<Envelope>>` a map of names to envelopes; Used to store all envelopes.

`public` Type Definitions:

- `typedef std::map<std::string, std::shared_ptr<Envelope>>::iterator iterator;` Makes Envelopes::iterator be an alias for the map iterator.
- `typedef std::map<std::string, std::shared_ptr<Envelope>>::const_iterator const_iterator;` Makes Envelopes::iterator be an alias for the map iterator.

`public` Methods:

- `Envelopes();` Default constructor, does not need to do anything.
- `virtual ~Envelopes();` Required, but empty.
- `void addEnvelope(const std::string& name, std::shared_ptr<Envelope> envelope);` Stores `envelope` in the map, using `name` as the key.
- `std::shared_ptr<Envelope> getEnvelope(const std::string& name);` If `name` exists in the map, returns the envelope associated with `name`. Otherwise, returns `nullptr`.
- `iterator begin();` Returns an iterator to the first item in the collection.
- `const_iterator begin() const;` Returns a constant iterator to the first item in the collection.
- `iterator end();` Returns an iterator to the end item in the collection. This is the item that doesn’t exist.
- `const_iterator end() const;` Returns a constant iterator to the end item in the collection. This is the item that doesn’t exist.

Update `library-envelope/Makefile`

Add `Envelopes.{h,cpp}` in the appropriate places to add them to the library and install the header file.

Create `library-instrument/Instrumentarium.{h,cpp}`

`Instrumentarium` Class

Data Members:

The `Instrumentarium` class should contain data members to track the following information. These data members should be `protected` or `private`. They are not allowed to be `public`.

- `std::map<std::string, std::shared_ptr<Instrument>>` a map of names to instruments; Used to store all instruments.

`public` Type Definitions:

- `typedef std::map<std::string, std::shared_ptr<Instrument>>::iterator iterator;` Makes Instrumentarium::iterator be an alias for the map iterator.
- `typedef std::map<std::string, std::shared_ptr<Instrument>>::const_iterator const_iterator;` Makes Instrumentarium::iterator be an alias for the map iterator.

`public` Methods:

- `Instrumentarium();` Default constructor, does not need to do anything.
- `virtual ~Instrumentarium();` Required, but empty.

- `void addInstrument(const std::string& name, std::shared_ptr<Instrument> instrument);` Stores `instrument` in the map, using `name` as the key.
- `std::shared_ptr<Instrument> getInstrument(const std::string& name);` If `name` exists in the map, returns the instrument associated with `name`. Otherwise, returns `nullptr`.
- `iterator begin();` Returns an iterator to the first item in the collection.
- `const_iterator begin() const;` Returns a constant iterator to the first item in the collection.
- `iterator end();` Returns an iterator to the end item in the collection. This is the item that doesn't exist.
- `const_iterator end() const;` Returns a constant iterator to the end item in the collection. This is the item that doesn't exist.

Update `library-instrument/Makefile`

Add `Instrumentarium.{h,cpp}` in the appropriate places to add them to the library and install the header file.

Additional Documentation

TBA

Grading Instructions

To receive credit for this assignment:

- your code must be pushed to your repository for this class on GitHub
- all unit tests must pass
- all acceptance tests must pass
- all programs must build, run, and execute as described in the assignment descriptions.

Extra Challenges (Not Required)

TBA