CS 3005: Programming in C++

Score Editor

Introduction

We are adding support for envelopes to the score editor program now. This includes user commands to add, edit, and list envelopes. It also includes a command to read a score file that contains waveforms and envelopes.

Future assignments will continue to add functionality to the program.

Syntax in the `.score` file for an envelope

This format was designed to be read using the C++ standard library's `>>` operator. All values are whitespace delimited. By context, your code should be able to determine whether the next value is a `std::string` or a `double`.

The ENVELOPE keyword must always be followed by the envelope name and the envelope type.

Below you will see the possible fields that can be present for various envelope types. Note that the parameters, such as DECAY-SECONDS and SUSTAIN-AMPLITUDE, are all optional, and order does not matter. If not specified, then the value set in the default constructor should be left in place. If accidentally specified more than once, the last value should be used.

If there is no `ENVELOPE-END` keyword, then the envelope is invalid, and should not be added to the score.

ADSR Envelope

```
ENVELOPE name1 ADSR
  MAXIMUM-AMPLITUDE 0.25
  ATTACK-SECONDS 0.01
  DECAY-SECONDS 0.02
  SUSTAIN-AMPLITUDE 0.5
  RELEASE-SECONDS 0.03
ENVELOPE-END
```

AD Envelope

```
ENVELOPE name2 AD
  MAXIMUM-AMPLITUDE 0.25
  ATTACK-SECONDS 0.01
ENVELOPE-END
```

Other Envelope

If you have other envelope types implemented, please add them to the reader.

Notes on the `ScoreReader::readEnvelope` method

- The method is called *after* the `ENVELOPE` keyword has been read, so it will start reading the name and type.
- If the envelope name refers to an envelope already existing in the score, the method still needs to read until the `ENVELOPE-END` keyword, but it should not make any changes to the existing envelope.
- If the envelope name does not exist in the score, the method should use the factory to create an envelope and add it to the score.
- Each of the keywords encountered should be consumed by the method, and set the appropriate fields in the envelope, if the envelope is new.
- Any word that is read in a position that should be a keyword, but is not a recognized keyword, should cause the envelope to not be added to the score. The method should return immediately if this happens.
- If the ending keyword is not found before the end of the input stream, the envelope should not be added to the score.
- Under normal operation, the method returns the envelope pointer, whether it is a new pointer that has

been added to the score, or it is an already existing envelope in the score.
- If an error occurs, such as reading non-keyword or end of input stream without ending keyword, the method should return a null pointer.

Assignment

Here are the *new* commands that are required for this assignment. Previous commands are still required.

Command	Prefixable?	Function	Description
`score-read`	no	`readScoreUI`	Read score from file.
`score-list-envelopes`	no	`listScoreEnvelopesUI`	List envelopes in the score.
`score-add-envelope`	no	`addScoreEnvelopeUI`	Add envelope to the score.
`score-edit-envelope`	no	`editScoreEnvelopeUI`	Edit envelope in the score.

Example Session

```
$ ./program-score-editor/score_editor
Choice? menu
Options are:
  # - Skip to end of line (comment).
  comment - Skip to end of line (comment).
  echo - Echo back the arguments given.
  help - Display help message.
  menu - Display help message.
  quit - Terminate the program.
  score-add-envelope - Add envelope to the score.
  score-add-waveform - Add waveform to the score.
  score-edit-envelope - Edit envelope in the score.
  score-edit-waveform - Edit waveform in the score.
  score-list-envelopes - List envelopes in the score.
  score-list-waveforms - List waveforms in the score.
  score-read - Read score from file.

Choice? score-add-envelope
Envelope name: adsr1
Envelope type: ADSR
Maximum amplitude: 0.1
Attack seconds: 0.2
Decay seconds: 0.3
Sustain amplitude: 0.4
Release seconds: 0.5
Choice? score-add-envelope
Envelope name: ad1
Envelope type: AD
Maximum amplitude: 0.6
Attack seconds: 0.7
Choice? score-edit-envelope
Envelope name: ad1
Maximum amplitude(0.6): 0.11
Attack seconds(0.7): 0.12
Choice? score-edit-envelope
Envelope name: adsr1
Maximum amplitude(0.1): 0.21
Attack seconds(0.2): 0.22
Decay seconds(0.3): 0.23
Sustain amplitude(0.4): 0.24
Release seconds(0.5): 0.25
Choice? score-list-envelopes
ad1 : ad1 AD 0.11
adsr1 : adsr1 ADSR 0.21
Choice? quit
```

Programming Requirements

Update `library-envelope/Envelope.h`

These constants should be used by the `ScoreReader`, instead of using string literals.

- `const std::string MAXIMUM_AMPLITUDE = "MAXIMUM-AMPLITUDE";` A global constant. This can be used by the

`ScoreReader` to reduce the number of “magic” strings.
- `const std::string ATTACK_SECONDS = "ATTACK-SECONDS";` A global constant. This can be used by the `ScoreReader` to reduce the number of “magic” strings.
- `const std::string DECAY_SECONDS = "DECAY-SECONDS";` A global constant. This can be used by the `ScoreReader` to reduce the number of “magic” strings.
- `const std::string SUSTAIN_AMPLITUDE = "SUSTAIN-AMPLITUDE";` A global constant. This can be used by the `ScoreReader` to reduce the number of “magic” strings.
- `const std::string RELEASE_SECONDS = "RELEASE-SECONDS";`

Update `library-score/MusicalScore.{h,cpp}`

We will add to the `MusicalScore` class by adding the `Envelopes` collection. Future assignments will add more to the class.

`MusicalScore` Class

This class will store all of the information for a piece of music.

Data Members:

- An `Envelopes` collection object for storing all envelopes that may be used in the music.

`public` Methods:

- `MusicalScore();` Allow the `Envelopes` object to be default constructed. Should not require any changes.
- `Envelopes& getEnvelopes();` Return the data member.
- `const Envelopes& getEnvelopes() const;` Return the data member.

Update `library-score-io/ScoreReader.{h,cpp}`

We will update the `ScoreReader` class by adding the ability to read `Envelope` objects. Future assignments will add more to the class.

`ScoreReader` Class

This class will eventually read all of the information for a piece of music from the `.score` file format.

Data Members:

No data members are required.

`public` Methods:

- `void readScore(std::istream& input_stream, MusicalScore& score) const;` Add the ability to recognize the ENVELOPE keyword, and call `readEnvelope()` to read the envelope.
- `std::shared_ptr<Envelope> readEnvelope(std::istream& is, MusicalScore& score) const;` This method reads an envelope formatted with the format described above.

Update `library-commands/score_editor_aux.{h,cpp}`

Commands created for the score editor program will go here. We’ll make a few in this assignment, and add more in future assignments.

Functions:

- `void readScoreUI(ApplicationData& app);` Asks the user for a filename that holds a score. Opens the file. If successfully opened, uses a temporary `ScoreReader` object to read the score into the `ApplicationData` object’s `MusicalScore`. If not opened successfully, gives an error message. See example above for formatting details.
- `void listScoreEnvelopesUI(ApplicationData& app);` Displays all of the envelopes currently stored in the score. See example above for formatting details.
- `void addScoreEnvelopeUI(ApplicationData& app);` Asks the user for required information. Uses the factory to create a new envelope. Based on the type prompts the user for additional information and configures

the envelope accordingly. If the creation and configuration is successful, adds the envelope to the score. Otherwise, gives an error message. If you have additional envelope types, please make this method handle them correctly. See example above for formatting details.
- `void editScoreEnvelopeUI(ApplicationData& app);` Ask the user for required information. If the envelope name identifies an existing envelope, prompt the user for required information to reconfigure the envelope, based on its type. When editing envelope attributes, display the current value of the attribute in the prompt. If there is an error with the envelope name, display an error message. See example above for formatting details.
- `int register_score_editor_commands(ApplicationData& app_data);` Update to also add the new commands specified for this assignment.

Additional Documentation

- None

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