CS 3005: Programming in C++

Musical Staves

Introduction

In this assignment you will add a collection of musical staff objects. (The plural of staff is staves.) In addition you will add commands to the score editor to add, edit, and display staves. The score reader and score writer classes will also be updated to fully support staves.

Syntax in the `.score` file for staff

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`.

There is no new syntax to the `.score` file format. We already added the `STAFF` in the previous assignment. This assignment will make it so the score reader actually keeps each staff it reads, and the score writer will write out all of the staves in the score.

Notes on the `ScoreReader::readScore` method

- After reading the staff object, add it to the staves collection.

Notes on the `ScoreWriter::writeScore` method

- After writing out all instruments, write out all staves.

Assignment

Here are the *new* commands that are required in the score editor program for this assignment. Previous commands are still required.

Command	Prefixable?	Function	Description
`score-staff-set-instrument`	no	`setStaffInstrumentUI`	Change instrument assigned to a staff in the score.
`score-list-staves`	no	`listScoreStavesUI`	List staves in the score.
`score-add-staff`	no	`addStaffUI`	Add a staff to the score.
`score-show-staff`	no	`showStaffUI`	Display staff details.
`score-staff-add-note`	no	`addStaffNoteUI`	Add a note to a staff.
`score-render`	no	`renderScoreUI`	Render score to wav file.

Example Session

```
$ ./program-score-editor/score_editor
Choice? score-add-waveform
Waveform name: w1
Waveform type: sine
Amplitude: 1.0
Choice? score-add-envelope
Envelope name: e1
Envelope type: AD
Maximum amplitude: 1.0
Attack seconds: 0.1
Choice? score-add-instrument
Instrument name: i1
Waveform name: w1
Envelope name: e1
Choice? score-add-staff
Staff name: s1
Instrument name: i1
Choice? score-list-staves
s1 i1
Choice? score-show-staff
```

```
Staff name: s1
s1 i1

Choice? score-staff-add-note
Staff name: s1
Start: 0.5
Duration: q
Note: C4
Choice? score-show-staff
Staff name: s1
s1 i1
0.5 0.25 C4

Choice? score-add-staff
Staff name: s2
Instrument name: i1
Choice? score-staff-add-note
Staff name: s2
Start: 0.25
Duration: h
Note: G4
Choice? score-show-staff
Staff name: s2
s2 i1
0.25 0.5 G4

Choice? score-list-staves
s1 i1
s2 i1
Choice? score-render
Filename: foo.wav
Samples per second: 1000
Bits per sample: 16
Choice? score-add-waveform
Waveform name: w2
Waveform type: square
Amplitude: 0.9
Choice? score-add-instrument
Instrument name: i2
Waveform name: w2
Envelope name: e1
Choice? score-staff-set-instrument
Staff name: s1
Instrument name: i2
Choice? score-list-staves
s1 i2
s2 i1
Choice? score-render
Filename: bar.wav
Samples per second: 1000
Bits per sample: 16
Choice? quit
```

Programming Requirements

Create `library-score/MusicalStaves.{h,cpp}`

`MusicalStaves` Class

This class will represent a collection of staff objects, with the key being the name of the staff and the value being the staff.

`protected` Data Members:

- `std::map<std::string, MusicalStaff> mStaves;` Map of staff names to staff objects.

`public` Methods:

- `MusicalStaves();` Does nothing.
- `virtual ~MusicalStaves();` Empty destructor.

- `void addStaff(const std::string& name, const MusicalStaff& staff);` Uses the name as the key and the staff as the value to store in the map.
- `MusicalStaff& getStaff(const std::string& name);` If name is a key, return the value associated with that key. If not, return a statically allocated `MusicalStaff` object. This is similar to `MenuData::getAction` returning a static object if the key is not valid. Except, this is more simple because we don't do any prefix matching.
- `const MusicalStaff& getStaff(const std::string& name) const;` Const version of `getStaff`.
- `unsigned int size() const;` Returns the number of items in the map.
- `typedef std::map<std::string, MusicalStaff>::iterator iterator;` Convenience typedef for iterator.
- `typedef std::map<std::string, MusicalStaff>::const_iterator const_iterator;` Convenience typedef for iterator.
- `iterator begin();` Returns the beginning iterator of the map.
- `const_iterator begin() const;` Returns the beginning iterator of the map.
- `iterator end();` Returns the ending iterator of the map.
- `const_iterator end() const;` Returns the ending iterator of the map.

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

We will add to the `MusicalScore` class by adding the `MusicalStaves` collection.

`MusicalScore` Class

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

Data Members:

- A `MusicalStaves` collection object for storing all staff objects that may be used in the music.

`public` Methods:

- `MusicalScore();` Allow the `MusicalStaves` object to be default constructed.
- `MusicalScore(const TimeSignature& time_signature, const double tempo);` Allow the `MusicalStaves` object to be default constructed.
- `void addStaff(const MusicalStaff& staff);` Adds staff to the collection of staves, using the staff's name as a key.
- `MusicalStaff& getStaff(const std::string& name);` Returns the staff object with the given name, from the collection.
- `const MusicalStaff& getStaff(const std::string& name) const;` Returns the staff object with the given name, from the collection.
- `MusicalStaves& getStaves();` Returns the data member.
- `const MusicalStaves& getStaves() const;` Returns the data member.
- `void renderStaff(const MusicalStaff& staff, const int samples_per_second, AudioTrack& track) const;` Uses the `MusicalStaff` `render` method to render the staff into the audio track.
- `void renderStaves(const int samples_per_second, std::map<std::string, AudioTrack>& tracks) const;` Renders each staff into a track, with the `renderStaff` method. Store each staff in the map, with the staff name as the key.
- `void renderWavChannels(const int samples_per_second, std::vector<AudioTrack>& channels) const;` Uses `renderStaves` to render all staves into a map of staves. Then adds all of the tracks to the `channels` vector, in the order obtained by looping over the map.

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

We will update the `ScoreReader` class by adding the ability to read store staves it reads.

`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;` When a staff is read, store it in the staves.

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

We will update the `ScoreWrite` class by adding the ability to write all staves.

`ScoreWriter` Class

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

Data Members:

No data members are required.

`public` Methods:

- `void writeScore(std::ostream& os, const MusicalScore& score) const;` After the instruments have been written, write all staves.

Create `library-score-io/WavWriter.{h,cpp}`

`WavWriter` Class

This class will render a score and save the rendered tracks to a WAV file.

`protected` Data Members:

None

`public` Methods:

- `WavWriter();` Empty constructor.
- `virtual ~WavWriter();` Empty destructor.
- `void writeWavFromScore(const MusicalScore& score, const int samples_per_second, const int bits_per_sample, const std::string& wav_filename) const;` Uses the `renderWavChannels` method on the `MusicalScore` object to render the score into channels. Uses a temporary `WAVFile` object to write the channels to a file.

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

Commands created for the score editor program will go here. We'll make a few more in this assignment.

Functions:

- `int register_score_editor_commands(ApplicationData& app_data);` Update to add the new commands specified for this assignment.
- `void setStaffInstrumentUI(ApplicationData& app);` Change the instrument assigned to a staff in the score. See example formatting above.
- `void listScoreStavesUI(ApplicationData& app);` List staves in the score. See example formatting above.
- `void addStaffUI(ApplicationData& app);` Add a staff to the score. See example formatting above.
- `void showStaffUI(ApplicationData& app);` Display the details of a staff. See example formatting above.
- `void addStaffNoteUI(ApplicationData& app);` Add a note to a staff. See example formatting above.
- `void renderScoreUI(ApplicationData& app)` Render the current score to a WAV file. Gathers required information from the user. Then uses a temporary `WavWriter` object to write a WAV file based on the score. See example formatting above.

Additional Documentation

-

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