# CS 3005: Programming in C++

## Instrument Designer

## Introduction

An instrument designer tool should allow a user to create an instrument by combining waveforms and envelopes to create an instrument's sound. The program should allow the user to also hear the sound of the instrument by writing the sound to a WAV file.

## Assignment

In this assignment, you will create the instrument designer program. This will be a text menu based system, where the user can create instances of waveforms, envelopes, and instruments. The user will also be able to create WAV files that can be played to demonstrate the sound of the instruments.

The commands this program will have are listed in this table.

| Command | Prefixable? | Function | Description |
|---|---|---|---|
| `help` | no | `menuUI` | Display help message. |
| `menu` | no | `menuUI` | Display help message. |
| `#` | yes | `commentUI` | Skip to end of line (comment). |
| `comment` | no | `commentUI` | Skip to end of line (comment). |
| `echo` | no | `echoUI` | Echo back the arguments given. |
| `quit` | no | `quitUI` | Terminate the program. |
| `list-waveforms` | no | `listWaveformsUI` | List waveforms in the inventory. |
| `add-waveform` | no | `addWaveformUI` | Add waveform to the inventory. |
| `edit-waveform` | no | `editWaveformUI` | Edit waveform in the inventory. |
| `list-envelopes` | no | `listEnvelopesUI` | List envelopes in the inventory. |
| `add-envelope` | no | `addEnvelopeUI` | Add envelope to the inventory. |
| `edit-envelope` | no | `editEnvelopeUI` | Edit envelope in the inventory. |
| `list-instruments` | no | `listInstrumentsUI` | List instruments in the inventory. |
| `add-instrument` | no | `addInstrumentUI` | Add instrument to the inventory. |
| `edit-instrument` | no | `editInstrumentUI` | Edit instrument in the inventory. |
| `record-instrument-note` | no | `recordInstrumentNoteUI` | Record a note for an instrument to a WAV file. |
| `configure-audio-track-and-wav-file` | no | `configure_audio_track_and_wav_file` | Configure meta data for the audio track and WAV file. |

- *Command* indicates the command the user types to request the command.
- *Prefixable?* indicates if the command should be searched for as a prefix (the first characters) of the command typed.
- *Function* indicates the function that the command will call when it is executed.
- *Description* indicates the description of the command.

Note that some of these commands are the same as were available in the previous assignment.

## Example Session

This sample session is kind of long. It builds several waveforms, envelopes, and instruments. It edits a waveform, an envelope, and an instrument. It displays the lists of waveforms, envelopes, and instruments. It also generates WAV files for a couple of instruments. Some of the assignment instructions will refer to this sample session to explain the output of several methods.

```
$ ./program-instrument-designer/instrument_designer
Choice? add-waveform
Waveform name: sine-loud
Waveform type: sine
```

```
Amplitude: 1.0
Choice? add-waveform
Waveform name: sine-soft
Waveform type: sine
Amplitude: 0.5
Choice? add-waveform
Waveform name: square-loud
Waveform type: square
Amplitude: 1.0
Choice? edit-waveform
Waveform name: sine-soft
Amplitude(0.5): 0.6
Choice? list-waveforms
sine-loud : sine-loud sine 1
sine-soft : sine-soft sine 0.6
square-loud : square-loud square 1
Choice? add-envelope
Envelope name: AD-quick
Envelope type: AD
Maximum amplitude: 1.0
Attack seconds: 0.01
Choice? add-envelope
Envelope name: AD-slow
Envelope type: AD
Maximum amplitude: 1.0
Attack seconds: 0.2
Choice? add-envelope
Envelope name: ADSR-1
Envelope type: ADSR
Maximum amplitude: 1.0
Attack seconds: 0.01
Decay seconds: 0.1
Sustain amplitude: 0.7
Release seconds: 0.02
Choice? edit-envelope
Envelope name: AD-slow
Maximum amplitude(1): 1.0
Attack seconds(0.2): 0.25
Choice? list-envelopes
AD-quick : AD-quick AD 1
AD-slow : AD-slow AD 1
ADSR-1 : ADSR-1 ADSR 1
Choice? add-instrument
Instrument name: quick-loud-square
Waveform name: square-loud
Envelope name: AD-quick
Choice? add-instrument
Instrument name: ADSR-sine
Waveform name: sine-soft
Envelope name: ADSR-1
Choice? edit-instrument
Instrument name: ADSR-sine
Waveform name: sine-loud
Envelope name: ADSR-1
Choice? list-instruments
ADSR-sine : sine-loud ADSR-1
quick-loud-square : square-loud AD-quick
Choice? configure-audio-track-and-wav-file
Samples/Second: 44100
Seconds: 1.0
Bits/Sample[8,16,24,32]: 32
Choice? record-instrument-note
Instrument name: ADSR-sine
Frequency: 261.626
WAV filename: middle-c-adsr-sine.wav
Choice? record-instrument-note
Instrument name: quick-loud-square
Frequency: 523.251
WAV filename: c5-loud-square.wav
Choice? quit
```

The WAV files created are available for download. You should compare the WAV files created by your program with these.

- c5-loud-square.wav
- middle-c-adsr-sine.wav

## Menu Example

```
$ ./program-instrument-designer/instrument_designer
Choice? menu
Options are:
# - Skip to end of line (comment).
add-envelope - Add envelope to the inventory.
add-instrument - Add instrument to the inventory.
add-waveform - Add waveform to the inventory.
comment - Skip to end of line (comment).
configure-audio-track-and-wav-file - Configure meta data for the audio track and WAV file.
echo - Echo back the arguments given.
edit-envelope - Edit envelope in the inventory.
edit-instrument - Edit instrument in the inventory.
edit-waveform - Edit waveform in the inventory.
help - Display help message.
list-envelopes - List envelopes in the inventory.
list-instruments - List instruments in the inventory.
list-waveforms - List waveforms in the inventory.
menu - Display help message.
quit - Terminate the program.
record-instrument-note - Record a note for an instrument to a WAV file.

Choice? quit
```

## Error Messages Example

```
$ ./program-instrument-designer/instrument_designer
Choice? add-waveform
Waveform name: fred
Waveform type: triangle
Unable to create a waveform of type 'triangle'.
Choice? add-envelope
Envelope name: wilma
Envelope type: ADAD
Unable to create an envelope of type 'ADAD'.
Choice? edit-waveform
Waveform name: barney
Unable to find a waveform with name 'barney'.
Choice? edit-envelope
Envelope name: betty
Unable to find an envelope with name 'betty'.
Choice? add-waveform
Waveform name: fred
Waveform type: sine
Amplitude: 1.0
Choice? add-envelope
Envelope name: wilma
Envelope type: AD
Maximum amplitude: 1.0
Attack seconds: 0.1
Choice? add-instrument
Instrument name: barney
Waveform name: bambam
Envelope name: dino
bambam does not name a waveform in this application.
Choice? add-instrument
Instrument name: flintstone
Waveform name: fred
Envelope name: dino
dino does not name an envelope in this application.
Choice? edit-instrument
Instrument name: flintstone
Waveform name: fred
Envelope name: wilma
flintstone does not name an instrument in this application.
Choice? record-instrument-note
Instrument name: flintstone
flintstone does not name an instrument in this application.
```

```
Choice? quit
```

## Programming Requirements

## Update `library-application/ApplicationData.{h,cpp}`

We will update the `ApplicationData` class to support collections of `Waveform`, `Envelope`, and `Instrument` objects.

### `ApplicationData` Class

#### Data Members:

These additional data members should be in the `private` or `protected` section of the class. No `public` data members are allowed. The data members allow the program to keep track of collections of `Waveform`, `Envelope`, and `Instrument` objects.

- `Waveforms` a collection of waveforms.
- `Envelopes` a collection of envelopes.
- `Instrumentarium` a collection of instruments.

#### `public` Methods:

We need to provide non-`const` versions of these methods to support changing the contents of the collections. We also need `const` versions for some read-only access to the contents of the collections.

- Constructor: No changes needed. The new data members all have default constructors that create empty collections, as we desire.
- `Waveforms& getWaveforms();` Returns the data member.
- `const Waveforms& getWaveforms() const;` Returns the data member.
- `Envelopes& getEnvelopes();` Returns the data member.
- `const Envelopes& getEnvelopes() const;` Returns the data member.
- `Instrumentarium& getInstrumentarium();` Returns the data member.
- `const Instrumentarium& getInstrumentarium() const;` Returns the data member.

## Create `library-commands/instrument_designer_aux.{h,cpp}`

These files will contain the necessary user interaction commands to create the functionality of the instrument designer program.

#### Functions:

- `void listWaveformsUI(ApplicationData& app_data);` Loops through the waveforms in the `Waveforms` collection from the `ApplicationData` object and displays them as shown in the example above.
- `void addWaveformUI(ApplicationData& app_data);` Prompts the user for data necessary to add a waveform to the collection. Uses the `WaveformFactory` to create a `Waveform` object and adds it to the collection. See the example above for details of the user interaction.
- `void editWaveformUI(ApplicationData& app_data);` Retrieves the user selected waveform from the collection, and allows the user to change the parameters of the waveform. See the example above for details of the user interaction.
- `void listEnvelopesUI(ApplicationData& app_data);` Displays the envelopes in the format shown in the example above.
- `void addEnvelopeUI(ApplicationData& app_data);` Prompts the user for data necessary to add an envelope to the collection. Uses the factory to create an envelope object and adds it to the collection. See the example above for details of the user interaction.
- `void editEnvelopeUI(ApplicationData& app_data);` Retrieves the user selected envelope from the collection, and allows the user to change the parameters of the envelope. See the example above for details of the user interaction.
- `void listInstrumentsUI(ApplicationData& app_data);` Displays the instruments in the format shown in the example above.
- `void addInstrumentUI(ApplicationData& app_data);` Prompts the user for data necessary to add an instrument to the collection. Uses the factory to create an instrument object and adds it to the collection. See the example above for details of the user interaction.
- `void editInstrumentUI(ApplicationData& app_data);` Retrieves the user selected instrument from the collection, and allows the user to change the parameters of the instrument. See the example above for

details of the user interaction.
- `void put_frequency_in_register(ApplicationData& app_data);` Prompts the user for a double, “Frequency: “. Stores it in register `0` of the application data.
- `void recordInstrumentNoteUI(ApplicationData& app_data);` Allows the user to select an instrument. Then uses `put_frequency_in_register` to allow the user to select a frequency. Uses `fill_audio_track_from_instrument` to fill the audio track with audio data from the instrument. Configures the channels in the application data to have two tracks, then copies the audio track into both audio tracks. Uses `save_wav_file`, to create a WAV file.
- `int register_instrument_designer_commands(ApplicationData& app_data);` Registers all of the commands in the table above to the application data. The commands that are the same as the menu test program can be added by calling `register_menu_test_commands`. Returns 0.
- `int instrument_designer(ApplicationData& app_data);` Registers the commands for the instrument designer, then starts the main loop. Returns 0.

## Update `library-commands/Makefile`

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

## Create `program-instrument-designer/instrument_designer.cpp`

### Functions:

- `int main();` Entry point to the instrument designer program. Should create an `ApplicationData` and pass it to the `instrument_designer` function and return the result of that function call.

## Create `program-instrument-designer/Makefile`

This file must contain rules such that any of the following commands will build the `instrument_designer` program:

- `make`
- `make all`
- `make instrument_designer`

## Create `program-instrument-designer/.gitignore`

The file needs to store one line of text:

```
instrument_designer
```

This will prevent the executable program from being committed to the repository. It is a *derived file*.

## Update `Makefile`

- Update the project-level Makefile so that `make` and `make all` in the project directory will call `make` in the `program-instrument-designer` directory.
- If necessary, make sure the order of make commands is correct to build prerequisite libraries in the correct order.

## 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