Programming Assignment #1

CS 246, Fall 2018

Due Wednesday, September 12

Audio input and output

In this course, we will use PortAudio for audio input and output. You can download the source from here

    http://www.portaudio.com/

However, you will need to compile and install it on your system. For your convenience, I have pre–compiled the code using MSVC 2015 (it should work with MSVC 2017 as well) for use on Windows machines. You can get the pre–compiled libraries from the CS 246 Moodle page.

Note that PortAudio is a cross–platform low–level audio API. In fact, I will compile and run your code on my Linux box. This means that you should not include any header files in your code that are platform–specific, such as `windows.h` or `conio.h`.

User interface

For real–time audio processing, we will also need a way of enabling user interaction. Although we could use keyboard input for this, it is more sensible to use a GUI interface. Unfortunately, there are no simple cross–platform APIs available for this. There are cross–platform general graphics APIs, such as FLTK or GTK. However, the time and effort needed for learning and using them would be more than is appropriate for our simple needs. For this reason, I have created a very basic (read: usable but visually unappealing) API for simple audio controls, which takes the form of a class named `Control`.

The public interface to the class is

```
class Control {
  public:
    Control(unsigned n, const char *title="Control Class 0.2");
    virtual ~Control(void);
    void setRange(unsigned index, int min, int max);
    void setLabel(unsigned index, const char *label);
    void setValue(unsigned index, int value);
    void show(bool visible=true);
    void setTitle(const char *title);
    virtual void valueChanged(unsigned index, int value) {};
};
```

`Control(n,title)` — (constructor) creates a window with n audio controls (sliders). By default, the window is not visible. The `show` function should be called to make the window visible (see below). The sliders are indexed from 0 to $n-1$. The window title may be (optionally) specified using the variable `title` (the window title may also be changed at any time by calling `setTitle`).

`~Control()` — (destructor) destroys the audio control class instance. The window of audio controls will be destroyed as a result.

`setRange(index,min,max)` — sets the range of slider values to the range `min`...`max` (inclusive). The values of `min` and `max` must be nonnegative, and `min` must be less than `max`. The function call will only effect the slider whose index is given by the value of `index`. The default slider range is $0 \ldots 100$.

`setLabel(index,label)` — sets the slider label the character string pointed to by `label`. The function call will only effect the slider whose index is given by the value of `index`.

`setValue(index,value)` — sets the slider value to a specified value. The value of `value` must be within the range specified by the call to `setRange`. The function call will only effect the slider whose index is given by the value of `index`. The default value is 0.

`show(visible)` — hides or shows the audio control window. If `visible` is set to `true`, the window will be made visible. A value of `false` will hide the window. By default, the window is hidden (not visible).

`setTitle(title)` — sets the title of the control window to the string specified by `title`.

`valueChanged(index,value)` — hook function called whenever the user changes a slider value. The value of `index` gives the index of the slider that has been changed, and `value` gives the new value of the slider. The default implementation of this function is trivial: it does nothing. To change the default behavior, the function must be overridden in a derived class.

To use this class, you need to (1) use `Control` as a base class for some derived class, (2) use the derived class constructor to initialize the controls, and (3) implement the `valueChanged` virtual function to respond to user input.

Programming task

Write a program that plays a tone and allows the user to alter the pitch and amplitude of the tone in real time.

- You may output any type of tone: a sine wave, triangle wave, et cetera. However, you may *not* use the quadratic spline tone that I use in the PortAudio demo program, and you may not use a square wave (this includes a non–symmetric square wave). If you use a sine wave, you do not have to use a look–up table.
- Three controls are required: (1) octave, (2) pitch offset, and (2) amplitude/volume.
- The octave control should allow the user to change the output tone by a factor of 2^m, where m is an integer in the range $-2 \ldots 2$ (a total of 5 octaves). The initial value of the control should be 0.
- The pitch offset control should allow the user to continuously change the pitch of the output tone by any value between -1200 cents and 1200 cents. The initial value of the control should be 0 cents.
- The volume control should allow the user to continuously change the volume to any value between -24 dB and 24 dB. If the volume is over 0 dB (maximum volume), clipping should be performed. The initial value of the control should be 0 dB.
- The initial tone should be 220 Hz. The current frequency of the tone that is playing should be displayed in the window title.
- When the user changes parameter values, the audio output should change in a smooth manner, with little to no audible clicks.

Your program should be called `ToneOut.cpp`. You may include the header files `portaudio.h` and `Control.h`, as well as any *standard* C++ header file.