Programming Assignment #3

CS 246, Fall 2018

Due Wednesday, September 26

As stated in the previous assignment all of the filters that we will implement in the homework assignments, we will derive from the following interface class contained in the file `Filter.h`.

```
class Filter {
  public:
    virtual ~Filter(void) { }
    virtual float operator()(float x) = 0;
};
```

That is, any filter that we implement will be guaranteed to provide an overloaded `()` operator. The initial (0–th) call to this function will return the output $y = y_0$ of the filter at time $t = 0$, given an input value of $x = x_0$; i.e., returns y_0, with x_0 as input. The n–th call to this function will return the output of the filter $y = y_n$ at time $t_n = n/R$, given an input value of $x = x_n$. That is, it returns y_n, with x_n as input. Here R is the sampling rate.

An Improved Low Pass Filter

In class we obtained a digital first order low pass filter by discretizing the differential equation that governs an analog low pass filter circuit. This is sufficient for many applications. However, discretization behaves poorly for high frequencies. We will learn how to remedy this later on in the course. For right now, I will simply tell you the result. The transfer function for the improved first order low pass filter is

$$H_L(z) = \frac{a_L(1+z^{-1})}{1-b_L z^{-1}}, \quad \text{with} \quad a_L \doteq \frac{\theta}{1+\theta},\ b_L \doteq \frac{1-\theta}{1+\theta},\ \theta \doteq \tan\left(\frac{\pi f}{R}\right)$$

where f is the cutoff frequency.

Programming Task #1

I will give you the header file `LowPass.h`, which declares a class for our improved LP filter:

```
class LowPass : public Filter {
  public:
    LowPass(float f=0, float R=44100);
    void setFrequency(float f);
    float operator()(float x);
  private:
    double irate,
```

```
            a0, b1,
            x1, y1;
};
```

Where the member functions should do the following.

`LowPass(f,R)` — (constructor) creates a low pass filter with cutoff frequency f and sampling rate R.

`setFrequency(f)` — resets the cutoff frequency of the filter to the value f.

`operator()(x)` — returns the result of applying the low pass filter to the input sample value x.

Your submission for this part of the assignment will consist of a single file: the implementation file `LowPass.cpp`. You may only include the header files `Filter.h`, `LowPass.h`, and `cmath`.

An Improved High Pass Filter

Similarly, there is an improved version of the digital first order high pass filter discussed in class. The transfer function for it is:

$$H_H(z) = \frac{a_H(1 - z^{-1})}{1 - b_H z^{-1}}, \quad \text{with} \quad a_H \doteq \frac{1}{1+\theta},\ b_H \doteq \frac{1-\theta}{1+\theta},\ \theta \doteq \tan\left(\frac{\pi f}{R}\right)$$

where f is the cutoff frequency. One may show that $H_L(z) + H_H(z) = 1$, provided that both filters use the same cutoff frequency.

Programming Task #2

As with the LP filter, I will give you the header file `HighPass.h` which declares the following class for our improved first order high pass filter.

```
class HighPass : public Filter {
  public:
    HighPass(float f=0, float R=44100);
    void setFrequency(float f);
    float operator()(float x);
  private:
    double irate,
           a0, b1,
           x1, y1;
};
```

Where the member functions should do the following.

`HighPass(f,R)` — (constructor) creates a high pass filter with cutoff frequency f and sampling rate R.

`setFrequency(f)` — resets the cutoff frequency of the filter to the value f.

`operator()(x)` — returns the result of applying the high pass filter to the input sample value x.

Your submission for this part of the assignment will consist of the implementation file `HighPass.cpp`. You may only include the header files `Filter.h`, `HighPass.h`, and `cmath`.