EECS 293
Software Craftsmanship
2014 Fall Semester

# Programming Assignment 11

Due at your recitation session on November 10-14

## Reading

Read Chapter 6 in the textbook.

## Programming

In this assignment, you will design classes to model *filters*, a basic concept in signal processing. A filter repeatedly takes an input value and produces an output value. In the simplest case (*scalar filters*), the inputs and outputs are doubles, but filters can operate on different types (double vectors are popular), and in fact the type of the input and of the output may differ. The simplest filter, the *identity filter*, always produces as output the same value it has read as input:

| Input | Output |
|---|---|
| 3. | 3. |
| 0. | 0. |
| -1. | -1. |
| 2. | 2. |

More useful filters output the maximum, minimum, or the arithmetic mean of the input signal seen so far. The maximum and the minimum are defined according to some total order on the input values. These filters are useful to find extreme points in the signal (for example, the maximum round-trip time of a packet exchange over the Internet). The arithmetic mean filter is a scalar filter, and can be used to average a sequence of data readings.

Filters are typically operational for long periods, which means that the input sequence can be arbitrarily long. Therefore, it is impractical to store anywhere the full sequence of input values. Filters must be implemented so that they maintain only a short history of the input (or none at all). Furthermore, each input should be processed as quickly as possible.

Filters can support an optional reset operation. The reset uses a parameter of the same type as the input. The meaning of reset depends on the type of filter. For example, a max filter will return the maximum value seen since the last reset. In fact, max, min, and average filters come in two types:

- Filters that return the max, min, average input value since the beginning or since the last reset, whichever occurred last, and
- Filters that return the max, min, or average of the last *N* values (or less if less than *N* values are available since the beginning or the last reset).

For example,

| Input | Reset | Max ∞ | Max 3 |
|---|---|---|---|
| -1 | | -1 | -1 |
| 1 | | 1 | 1 |
| 2 | | 2 | 2 |
| | 0 | | |
| -1 | | 0 | 0 |
| -2 | | 0 | 0 |
| -3 | | 0 | -1 |
| 3 | | 3 | 3 |
| 1 | | 3 | 3 |
| 2 | | 3 | 3 |
| 1 | | 3 | 2 |

As you can see from this example, the reset value enters into the computation as if it were an input (the max of the reset value 0 and of -1, -2, and -3 is 0). The value of *N* (if any) is set up once before the filter starts accepting any input, and is never modified.

A *filter cascade* is a filter that consists of a sequence of filters: a filter in the sequence takes its input from the output of the preceding filter. For example, the cascade of a max2 filter followed by a min3 filter are:

| Input | Max2 Output | Cascade Output |
|---|---|---|
| -1 | -1 | -1 |
| 3 | 3 | -1 |
| 1 | 3 | -1 |
| 2 | 2 | 2 |
| 1 | 2 | 2 |

A cascade cannot be reset as a whole. Therefore, it does not support an aggregate reset method. However, if an individual filter in the cascade is resettable, then it can be reset independently of the other filters in the cascade.

A *scalar linear filter* is defined by the parameters $N$, $M$, $a_1$, $a_2$, …, $a_M$, $b_0$, $b_1$, …, $b_N$. These parameters are set up once before the filter starts accepting any input and are never modified. Then, at the $i$th iteration the output $y_i$ is calculated according to:

$$y_i + a_1 y_{i-1} + a_2 y_{i-2} + \ldots + a_M y_{i-M} = b_0 x_i + b_1 x_{i-1} + b_2 x_{i-2} + \ldots b_N x_{i-N}$$

where $x_i$ is the $i$th value of the input, $y_i$ is the $i$th value of the output, and the $a_1$, … , $a_M$ and $b_0$, … , $b_N$ are the filter parameters. A scalar linear filter can be reset to a value $r$, in which case the record of all of previous input values is set to $r$ and the record of all previous output values is set to $r\left(\sum_{i=0}^{N} b_i\right)/\left(1+\sum_{i=1}^{M} a_i\right)$ (can you see why?). If there are not enough inputs, the missing values of $x$ and $y$ are taken to be equal to their value as calculated during the last reset. If no reset ever occurred, the missing values are taken to be equal to 0. For example, if $M$=$N$=1, $a_1$=.1, $b_0$=$b_1$=.5, then

$$y_i=(x_i+x_{i-1})/2-y_{i-1}/10,$$

and so:

| Input | Reset | Output |
|---|---|---|
| -1 | | -.5 |
| 1 | | .05 |
| 2 | | 1.495 |
| | 0 | |
| -1 | | -.5 |
| 3 | | 1.05 |
| 1 | | 1.895 |
| 2 | | 1.3105 |
| 1 | | 1.36895 |

Scalar linear filters have a wide range of applications: for example, they can be used to smooth out the input signal, or to remove a periodic carrier signal from the input, or to determine the signal trend.

An important type of scalar linear filters is when $a_1$=$a_2$=…=$a_M$=0, which is called a *FIR filter*. A simple but useful case of a FIR filter always outputs the input multiplied by a constant factor: in this case, $b_1$=$b_2$=…=$b_N$=0 and $b_0$ is called the filter *gain*. Another useful FIR filter is the *binomial filter*, in which $b_i = \binom{N}{i}$ (binomial filters are often used to smooth out a window of data, for example in speech or image processing).

Your job is to create an object-oriented design for filters that follows the description above. Your classes will be used by other (hypothetical) programmers and experts in signal processing. The users will expect to have variables of at least the following types:

- `Filter`, with generic input and output types
- A generic `IdentityFilter`
- `ScalarFilter`
- A generic `MaxFilter`. In `MaxFilters`, either
    - The maximum follows the natural ordering of the input, or
    - The maximum is determined by a `Comparator`
- `MinFilter` (symmetric to the `MaxFilter`)
- `MaxFilterN`, `MinFilterN` (same as above, but uses at most *N* values).
- `AveragingFilter, AveragingFilterN`
- `FilterCascade`
- `ScalarLinearFilter`
- `FIRFilter`
- `GainFilter`
- `BinomialFilter`

The user will also need a way to cope with the potentially unsupported reset operation.

You can add as many additional interfaces, abstract classes, and concrete classes as you see fit. Types of filters should be represented by classes or interfaces, and commonality among filter types can be expressed through inheritance or containment. For each class or interface, you should describe at least the abstraction it captures, its position in the inheritance hierarchy, the signature of its constructors and public methods, its private data structures (if any) including but not limited to those that record previous inputs, and the pseudo-code of any complicated method. You can outline your architecture for error handling and your approach to testing.

Submit a design documents that describes your architectural decisions. Your document can optionally contain sketches and diagrams of your architecture. No implementation is required: you will implement your design in the next programming assignments. This assignment can be submitted on blackboard.

## Discussion Guidelines

The class discussion will focus on class design.