Lab session 2: Signals and Systems

Note: After some thought and discussion with assistants, I (Arnold Meijster) decided to have only 5 problems (not 6 as originally planned) in lab session 2. The originally planned problem 6 is moved to lab session 3. The reason for this shift is two-fold: from past experience we know that doing problem 5 correctly might involve quite some testing/debugging (i.e. takes a lot of time), and a correct implementation is essential for doing the originally planned problem 6 (which is about Pearson correlation). The plan is therefore to make, after the deadline of lab 2, the correct implementation for problem 5 publicly available such that it can be used as the reference implementation for lab session 3. Note that this means that there cannot be ANY extension of the deadline for lab 2. So, the deadline is really hard!

As a result of this plan change, the points per exercise have changed (probably in your advantage). The problems 1, 2, 3, and 4 are now worth 15 points. Problem 5 is worth 30 points. You get 10 points for free, totalling 100 points.

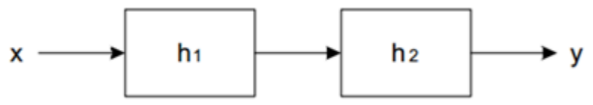
Problem 1: Linear filter
The input for this problem are a *kernel* $h[\,]$ of a FIR filter, and a discrete input signal $x[\,]$. Your output should be the signal $y[\,]$ that is obtained by feeding the FIR filter the input $x[\,]$. Your program should only show the samples of y where $x[\,]$ and $h[\,]$ have 'overlap' in the sliding window process (because you cannot print an infinite signal).

Example 1:
input:
```
2: [1,-1]
10: [0,0,0,1,1,1,1,0,0,0]
```
output:
```
11: [0,0,0,1,0,0,0,-1,0,0,0]
```

Example 2:
input:
```
10: [0,0,0,1,1,1,1,0,0,0]
2: [1,-1]
```
output:
```
11: [0,0,0,1,0,0,0,-1,0,0,0]
```

Example 3:
input:
```
3: [0,1,0]
3: [1,2,3]
```
output:
```
5: [0,1,2,3,0]
```

Problem 2: Cascade of two FIR filters
The input for this problem are the kernels $h_1[\,]$ and $h_2[\,]$ of two FIR filters, and a discrete input signal $x[\,]$. Your output should be the signal $y[\,]$ that is obtained by applying the filters as shown in the following diagram.



Example 1:
input:
```
2: [1,-1]
2: [1,-1]
10: [0,0,0,1,1,1,1,0,0,0]
```
output:
```
2: [0,0,0,1,-1,0,0,-1,1,0,0,0]
```

Example 2:
input:
```
2: [0,1]
3: [0,0,1]
4: [1,2,3,2]
```
output:
```
7: [0,0,0,1,2,3,2]
```

Example 3:
input:
```
1: [2]
2: [0,3]
4: [1,2,3,2]
```
output:
```
5: [0,6,12,18,12]
```

Problem 3: Series of FIR filters
This problem is similar to the previous problem, except that we place n FIR filters in a cascade (the previous exercise is the case $n = 2$). The first line of the input contains n. Next follow the n filter kernels h_1, h_2, .., h_n. The last input line is the input signal $x[\,]$. The output should be the signal $y[\,]$.

Example 1:

input:
```
2
2: [1,-1]
2: [1,-1]
10: [0,0,0,1,1,1,1,0,0,0]
```
output:
```
12: [0,0,0,1,-1,0,0,-1,1,0,0,0]
```

Example 2:

input:
```
5
2: [1,-1]
2: [1,-1]
2: [1,-1]
2: [1,-1]
2: [1,-1]
10: [0,0,0,1,1,1,1,0,0,0]
```
output:
```
15: [0,0,0,1,-4,6,-4,0,4,-6,4,-1,0,0,0]
```

Problem 4: Inverse of a FIR filter

The input of this problem consists of a discrete input signal $x[\,]$ followed by the output $y[\,]$ that is obtained by feeding $x[\,]$ to some unknown system. Your program should determine if the system might be FIR system. If it is not, the output should be `NO FIR`. If it is, your program should output the impulse reponse of the system.

Example 1:

input:
```
2: [1,-1]
5: [4,-2,2,-2,-2]
```
output:
```
4: [4,2,4,2]
```

Example 2:

input:
```
2: [1,-1]
5: [4,-2,1,-2,-2]
```
output:
```
NO FIR
```

Example 3:

input:
```
2: [1,-1]
4: [3,-1,-1,-1]
```
output:
```
3: [3,2,1]
```

Problem 5: Convolution via the convolution theorem

The input/output behaviour for this problem is the same as for problem 1. However, the length of the input signals $x[\,]$ and $h[\,]$ are chosen significantly larger. For many real-time applications, computing a convolution directly (as you did in problem 1) would be too slow. In this problem you will make a faster convolution algorithm. However, unfortunately the tests in Themis are still not big enough to reject the code from problem 1 as a solution for this problem (the time limit of 1 second is likely to be not exceeded). Still, submitting the code of problem 1 as a solution for this problem is considered cheating (and will be checked and reported by the teaching assistants).

In the 5th lecture (see slides on Nestor) the *convolution theorem* was presented. Implement yourself the version of the Discrete Fast Fourier Transform (FFT) that uses prime numbers (so no complex numbers!). This version of the algorithm should be faster than the direct implementation that you made for exercise 1. In your implementation, you should be using the prime 40961. Moreover, it is given that $\omega = 243$ is a *primitive 8192th root of unity* in the ring modulo 40961, i.e.

$$243^{8192} \textbf{ mod } 40961 = 1 \quad \text{and} \quad 243^i \textbf{ mod } 40961 \neq 243^j \textbf{ mod } 40961 \text{ for all } 0 \le i < j < 8192$$

There is a small demo program on Nestor (file `demopowmod.c`) which contains a handy routine to compute efficiently $a^b \textbf{ mod } c$. Once you implemented the transform, use the convolution theorem to implement the convolution algorithm which runs in $O(N \log N)$ time.

Note that the test sets in Themis are chosen such that overflow should not be an issue as long as you use `unsigned ints` (so not plain `ints`). Moreover, the length of the convolution does not exceed 8192, the inputs do not contain any negative numbers, and the output samples do not exceed 40960.

Example 1:

input:
```
3: [1,0,1]
5: [1,1,0,1,1]
```
output:
```
7: [1,1,1,2,1,1,1]
```

Example 2:

input:
```
4: [1,0,1,1]
5: [1,1,0,1,1]
```
output:
```
8: [1,1,1,3,2,1,2,1]
```

Example 3:
```
5: [1,1,0,1,1]
5: [1,1,0,1,1]
```
output:
```
9: [1,2,1,2,4,2,1,2,1]
```