UE TSM - Les Séries de Volterra
TD - Partie 2

KANG Jiale

November 15, 2024

Problem 1. *Euler implicite* is defined by

$$s = \frac{1 - z^{-1}}{T}$$

Question 7. Define $\nu = T\omega$, use *Euler implicite* to transform $F_1(s)$ to $F_1(z)$. Show the recurrence equation of input u_n and output y_n.

Solution. Recall that $F_1(s) = \frac{1}{1+s/\omega}$, in $\mathcal{Z}$-domain, $F_1(z) = \frac{\nu}{\nu+1-z^{-1}}$.

The relation between input and output is $Y(z) = F_1(z)U(z)$. Simplify this equation, we obtain

$$Y(z)(\nu + 1 - z^{-1}) = \nu U(z) \tag{1}$$

Transform equation (1) to time domain,

$$y_n = \frac{\nu}{\nu+1}u_n + \frac{1}{\nu+1}y_{n-1} \tag{2}$$

□

Problem 2. Program with FAUST.

Question 8. Implement F_1 with FAUST.

Solution. Equation (2) is a recurrent expression, we denote $a = \frac{\nu}{\nu+1}$, $b = \frac{1}{\nu+1}$. Code for the implementation is shown as Listing 1.

Listing 1: Code for the implementation of F_1

```
import("stdfaust.lib");

process = F1(1000);

F1(fc) = *(a) : + ~ *(b) with {
    nu = 2 * ma.PI * fc / ma.SR;
    a = nu / (nu + 1);
    b = 1 / (nu + 1);
};
```

□

Question 9. Implement a 1-layer 3-order none linear system.

Solution. Recall that block diagram of 3-order none linear system is shown as Figure 1.

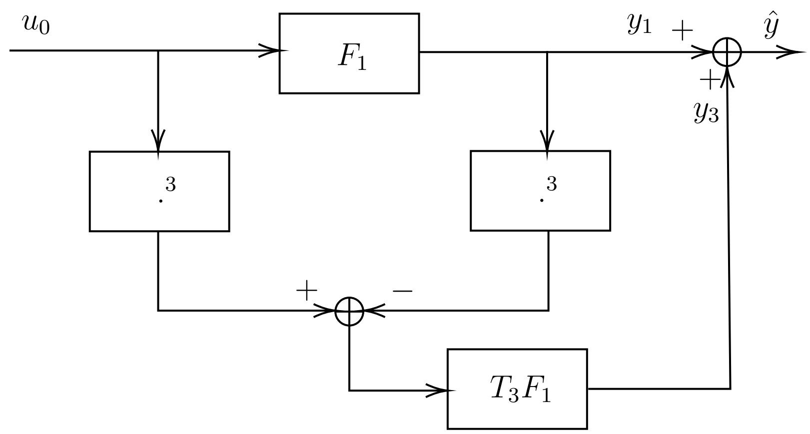

Figure 1: Block Diagram of 3-order None Linear System

According to this diagram , we could implement this system with help of Figure 2, the code of implementation for this system is shown as Listing 2.

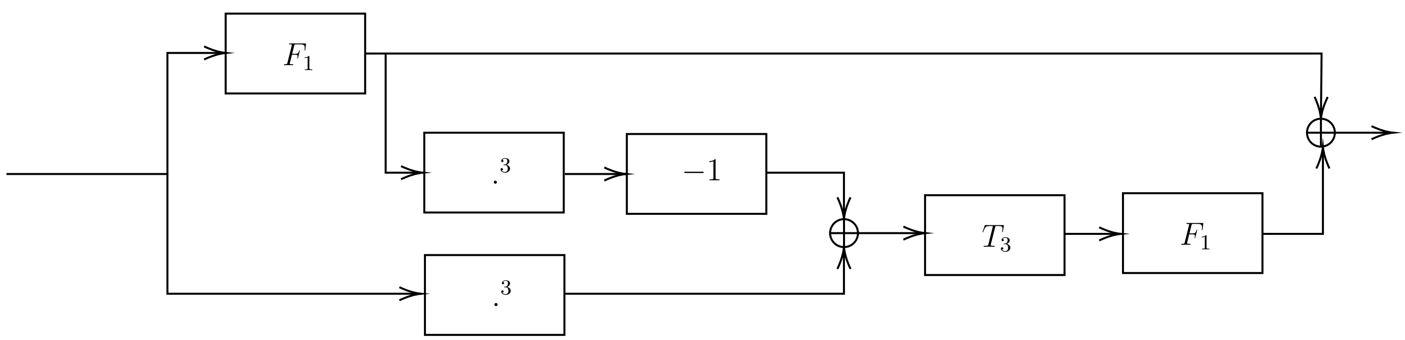

Figure 2: Diagram of Implementation with FAUST

Listing 2: Code for the implementation of 3-order None Linear System

```
import("stdfaust.lib");

process = E(1000);

F1(fc) = *(a) : + ~ *(b) with {
    nu = 2 * ma.PI * fc / ma.SR;
    a = nu / (nu + 1);
    b = 1 / (nu + 1);
};
```

```
NL(fc) =  _<: F1(fc), _             // devide into 2 parts in parallel
            : (_ <: _, _), _         // branch 1 devide into 2 parts in
               parallel and parallel with branch 2
            : _, ^(3), ^(3)          // identity, cubic, cubic in parallel
            : _, *(-1), _            // identity, (-1), identity in
               parallel
            : _, (+ : *(T3): F1(fc))// add with last two branches and
               follow by T3 F1
            : +;                     // two branches add
T3 = -1/3;
```

□

Question 10. Implement a 4-layer 3-order none linear system.

Solution. Recall that the kernel of a 4-layer 3-order none linear system could be expressed as

$$
\begin{aligned}
H_1(s_1) &= F_1(s_1)^4 \\
H_2(s_1, s_2) &= 0 \\
H_3(s_1, s_2, s_3) &= \sum_{k=0}^{3} F_1(s_1)^k F_1(s_2)^k F_1(s_3)^k F_3(s_1, s_2, s_3) F_1(s_1 + s_2 + s_3)^{3-k}
\end{aligned}
$$

Therefore, block diagram of 4-layer 3-order none linear system is shown as Figure 3. Obviously, it parallels 4 $\mathcal{M}$ and 1 $\mathcal{S}$ with input $\{u_0, 0\}$.

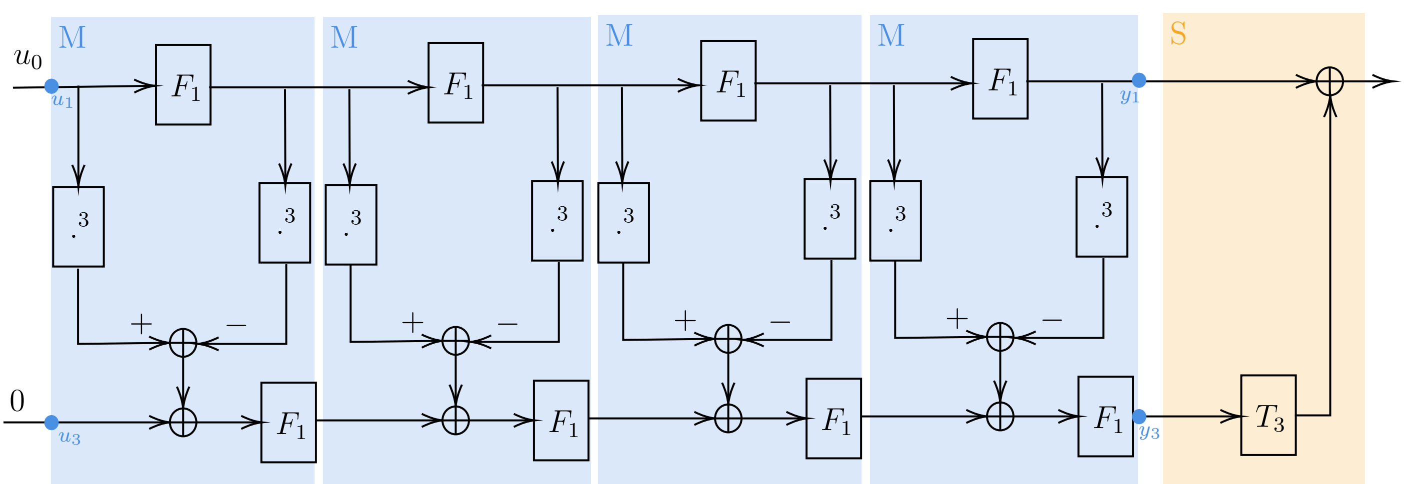

Figure 3: Block Diagram of 4-layer 3-order None Linear System

The code of implementation for this system is shown as Listing 3.

Listing 3: Code for the implementation of 4-layer 3-order None Linear System

```
import("stdfaust.lib");
```

```
process = fmoog(1000);

T3 = -1/3;
fmoog(fc) = _,0 : M : M : M : M : S with {
    F1 = *(a) : + ~ *(b) with {
        nu = 2 * ma.PI * fc / ma.SR;
        a = nu / (nu + 1);
        b = 1 / (nu + 1);
    };

    M(u1, u3) = F1(u1), F1(u3 + u1^3 - F1(u1)^3);

    S(y1, y3) =  y1 + T3 * y3;
};
```

□

Question 11. At what amplitude does distortion begin to be perceptible? To what order the sound seems to be real?

Solution. When we augment amplitude of the sound, the distortion begins to happen. We could perceive this distortion until its amplitude > 1.5. When its amplitude < 2, the sound seems to be real, because higher the amplitude could be, more non-harmonic frequencies will appear.

□

Question 12. When changing ω, is the stability of this filter still guaranteed?

Solution. As the cutoff frequency ω increases, the stability of the system can no longer be guaranteed. Because high frequencies non-harmonic components appear when we increase ω. This indicates that the system has entered the nonlinear region, resulting in non-harmonic distortion, and it may generate self-excited oscillations under the high cut off frequency. □

A Simulation code in FAUST

```
import("stdfaust.lib");

process = source(freq, delta) : drive(fmoog(fc));

T3 = -1/3 * checkbox("NL");

fmoog(fc) = _,0 : M : M : M : M : S with {
    F1 = *(a) : + ~ *(b) with {
        nu = 2 * ma.PI * fc / ma.SR;
        a = nu / (nu + 1);
        b = 1 / (nu + 1);
    };

    M(u1, u3) = F1(u1), F1(u3 + u1^3 - F1(u1)^3);
```

```
    S(y1, y3) =  y1 + T3 * y3;
};

drive(C) = *(g) : C : /(g) with {
    g = vslider("drive[style:knob]", 1, 0.1, 20, 0.1);
};

source(f1, df) = os.square(f1) + os.square(f1 + df) : *(0.5);
freq = vslider("freq[style:knob][scale:log][unit:Hz]", 330, 20, 5000, 1);
delta = vslider("delta[style:knob][unit:Hz]", 0.1, 0.05, 2, 0.05);
fc = vslider("fc[style:knob][scale:log][unit:Hz]", 1000, 20, 5000, 1);
```

B Auto Documentation

fmoog

November 15, 2024

compile˙options	-lang cpp -ct 1 -es 1 -mcd 16 -mdd 1024 -mdy 33 -single -ftz 0
filename	fmoog.dsp
maths.lib/author	GRAME
maths.lib/copyright	GRAME
maths.lib/license	LGPL with exception
maths.lib/name	Faust Math Library
maths.lib/version	2.8.0
name	fmoog
oscillators.lib/lf˙sawpos:author	Bart Brouns, revised by Stéphane Letz
oscillators.lib/lf˙sawpos:licence	STK-4.3
oscillators.lib/name	Faust Oscillator Library
oscillators.lib/sawN:author	Julius O. Smith III
oscillators.lib/sawN:license	STK-4.3
oscillators.lib/version	1.5.1
platform.lib/name	Generic Platform Library
platform.lib/version	1.3.0

This document provides a mathematical description of the Faust program text stored in the `fmoog.dsp` file. See the notice in Section 3 (page 4) for details.

1 Mathematical definition of `process`

The *fmoog* program evaluates the signal transformer denoted by `process`, which is mathematically defined as follows:

1. Output signal y such that

$$y(t) = p_{28}(t) \cdot (r_2(t) - p_{27}(t) \cdot r_1(t))$$

2. Input signal (none)

3. User-interface input signals u_{si} for $i \in [1, 4]$ and u_{c1} such that

"delta" (Hz)	$u_{s1}(t) \in [\,0.05, 2\,]$	(default value = 0.1)
"freq" (Hz)	$u_{s2}(t) \in [\,20, 5000\,]$	(default value = 330)
"drive"	$u_{s3}(t) \in [\,0.1, 20\,]$	(default value = 1)
"fc" (Hz)	$u_{s4}(t) \in [\,20, 5000\,]$	(default value = 1000)
"NL"	$u_{c1}(t) \in \{\,0, 1\,\}$	(default value = 0)

4. Intermediate signals p_i for $i \in [1, 28]$, s_i for $i \in [1, 12]$, r_i for $i \in [1, 11]$ and q_i for $i \in [1, 2]$ such that

$$
\begin{aligned}
p_1(t) &= \max\left(u_{s2}(t) + u_{s1}(t), 23.4489496824621\right)\\
p_2(t) &= \max\left(20, |p_1(t)|\right)\\
p_3(t) &= k_2 \cdot p_2(t)\\
p_4(t) &= \left(\frac{k_3}{p_2(t)}\right)\\
p_5(t) &= \max\left(0, \min\left(2047, \frac{k_4}{p_1(t)}\right)\right)\\
p_6(t) &= \mathrm{int}\left(p_5(t)\right)\\
p_7(t) &= \lfloor p_5(t) \rfloor\\
p_8(t) &= \left(p_7(t) + (1 - p_5(t))\right)\\
p_9(t) &= \max\left(u_{s2}(t), 23.4489496824621\right)\\
p_{10}(t) &= \max\left(20, |p_9(t)|\right)\\
p_{11}(t) &= k_2 \cdot p_{10}(t)\\
p_{12}(t) &= \left(\frac{k_3}{p_{10}(t)}\right)\\
p_{13}(t) &= \max\left(0, \min\left(2047, \frac{k_4}{p_9(t)}\right)\right)\\
p_{14}(t) &= \mathrm{int}\left(p_{13}(t)\right)\\
p_{15}(t) &= p_{14}(t) \oplus 1\\
p_{16}(t) &= \lfloor p_{13}(t) \rfloor\\
p_{17}(t) &= \left(p_{13}(t) - p_{16}(t)\right)\\
p_{18}(t) &= \left(p_{16}(t) + (1 - p_{13}(t))\right)\\
p_{19}(t) &= p_6(t) \oplus 1\\
p_{20}(t) &= \left(p_5(t) - p_7(t)\right)\\
p_{21}(t) &= \left(\frac{1}{p_2(t)}\right)\\
p_{22}(t) &= \left(\frac{1}{p_{10}(t)}\right)
\end{aligned}
$$

$$p_{23}(t) = k_5 \cdot u_{s4}(t) \cdot u_{s3}(t)$$
$$p_{24}(t) = k_6 \cdot u_{s4}(t)$$
$$p_{25}(t) = \left(\frac{1}{p_{24}(t)+1}\right)$$
$$p_{26}(t) = 0.5 \cdot u_{s3}(t)$$
$$p_{27}(t) = 0.333333333333333 \cdot u_{c1}(t)$$
$$p_{28}(t) = \left(\frac{1}{u_{s3}(t)}\right)$$

$$s_1(t) = 1 \ominus r_7(t{-}1)$$
$$s_2(t) = 2 \cdot r_6(t) + -1^2$$
$$s_3(t) = (s_2(t) - s_2(t{-}1))$$
$$s_4(t) = r_7(t{-}1)$$
$$s_5(t) = p_4(t) \cdot s_4(t) \cdot s_3(t)$$
$$s_6(t) = 2 \cdot r_{10}(t) + -1^2$$
$$s_7(t) = (s_6(t) - s_6(t{-}1))$$
$$s_8(t) = p_{12}(t) \cdot s_4(t) \cdot s_7(t)$$
$$s_9(t) = (k_3 \cdot s_4(t) \cdot (p_{22}(t) \cdot s_7(t) + p_{21}(t) \cdot s_3(t)) - p_{20}(t) \cdot s_5(t{-}p_{19}(t)) + p_{18}(t) \cdot s_8(t{-}p_{14}(t)) + p_{17}(t) \cdot s_8(t{-}p_{15}(t)) + p_8(t) \cdot s_5(t{-}p_6(t)))$$
$$s_{10}(t) = r_3(t)^3$$
$$s_{11}(t) = r_4(t)^3$$
$$s_{12}(t) = r_5(t)^3$$

$$r_7(t) = 1$$
$$r_6(t) = q_1(t) - \lfloor q_1(t) \rfloor$$
$$r_{10}(t) = q_2(t) - \lfloor q_2(t) \rfloor$$
$$r_5(t) = p_{25}(t) \cdot (r_5(t{-}1) + p_{23}(t) \cdot s_9(t))$$
$$r_4(t) = p_{25}(t) \cdot (r_4(t{-}1) + p_{24}(t) \cdot r_5(t))$$
$$r_3(t) = p_{25}(t) \cdot (r_3(t{-}1) + p_{24}(t) \cdot r_4(t))$$
$$r_2(t) = p_{25}(t) \cdot (r_2(t{-}1) + p_{24}(t) \cdot r_3(t))$$
$$r_{15}(t) = p_{25}(t) \cdot \left(r_{15}(t{-}1) + p_{24}(t) \cdot \left(p_{26}(t) \cdot s_9(t)^3 - s_{12}(t)\right)\right)$$
$$r_{14}(t) = p_{25}(t) \cdot (r_{14}(t{-}1) + p_{24}(t) \cdot (r_{15}(t) + s_{12}(t) - s_{11}(t)))$$
$$r_{13}(t) = p_{25}(t) \cdot (r_{13}(t{-}1) + p_{24}(t) \cdot (r_{14}(t) + s_{11}(t) - s_{10}(t)))$$
$$r_1(t) = p_{25}(t) \cdot \left(r_1(t{-}1) + p_{24}(t) \cdot \left(r_{13}(t) + s_{10}(t) - r_2(t)^3\right)\right)$$

$$q_1(t) = \begin{cases} p_3(t) + r_6(t-1) & \text{if } s_1(t) = 0 \\ 0 & \text{if } s_1(t) = 1 \end{cases} \quad q_2(t) = \begin{cases} p_{11}(t) + r_{10}(t-1) & \text{if } s_1(t) = 0 \\ 0 & \text{if } s_1(t) = 1 \end{cases}$$

5. Constants k_i for $i \in [1, 6]$ such that

$$k_1 = \min\left(192000, \max\left(1, f_S\right)\right)$$
$$k_2 = \left(\frac{1}{k_1}\right)$$
$$k_3 = 0.25 \cdot k_1$$
$$k_4 = 0.5 \cdot k_1$$
$$k_5 = \left(\frac{\pi}{k_1}\right)$$
$$k_6 = \left(\frac{2 \cdot \pi}{k_1}\right)$$

2 Block diagram of `process`

The block diagram of `process` is shown on Figure 1 (page 4).

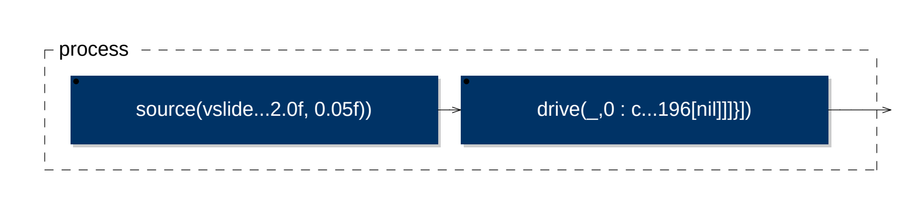

Figure 1: Block diagram of `process`

3 Notice

- This document was generated using Faust version 2.75.16 on November 15, 2024.
- The value of a Faust program is the result of applying the signal transformer denoted by the expression to which the `process` identifier is bound to input signals, running at the f_S sampling frequency.
- Faust (*Functional Audio Stream*) is a functional programming language designed for synchronous real-time signal processing and synthesis applications. A Faust program is a set of bindings of identifiers to expressions

that denote signal transformers. A signal s in S is a function mapping[1] times $t \in \mathbb{Z}$ to values $s(t) \in \mathbb{R}$, while a signal transformer is a function from S^n to S^m, where $n, m \in \mathbb{N}$. See the Faust manual for additional information (http://faust.grame.fr).

- Every mathematical formula derived from a Faust expression is assumed, in this document, to having been normalized (in an implementation-dependent manner) by the Faust compiler.

- A block diagram is a graphical representation of the Faust binding of an identifier I to an expression E; each graph is put in a box labeled by I. Subexpressions of E are recursively displayed as long as the whole picture fits in one page.

- $\forall\, x \in \mathbb{R}$,

$$\operatorname{int}(x) = \begin{cases} \lfloor x \rfloor & \text{if } x > 0 \\ \lceil x \rceil & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases} .$$

- This document uses the following integer operations:

operation	*name*	*semantics*
$i \oplus j$	integer addition	$\text{normalize}(i + j)$, in $\mathbb{Z}$
$i \ominus j$	integer substraction	$\text{normalize}(i - j)$, in $\mathbb{Z}$

Integer operations in Faust are inspired by the semantics of operations on the n-bit two's complement representation of integer numbers; they are internal composition laws on the subset $[\,-2^{n-1}, 2^{n-1}-1\,]$ of $\mathbb{Z}$, with $n = 32$. For any integer binary operation $\times$ on $\mathbb{Z}$, the $\otimes$ operation is defined as: $i \otimes j = \text{normalize}(i \times j)$, with

$$\text{normalize}(i) = i - N \cdot \text{sign}(i) \cdot \left\lfloor \frac{|i| + N/2 + (\text{sign}(i) - 1)/2}{N} \right\rfloor,$$

where $N = 2^n$ and $\text{sign}(i) = 0$ if $i = 0$ and $i/|i|$ otherwise. Unary integer operations are defined likewise.

- The `fmoog-mdoc/` directory may also include the following subdirectories:
  - `cpp/` for Faust compiled code;
  - `pdf/` which contains this document;
  - `src/` for all Faust sources used (even libraries);
  - `svg/` for block diagrams, encoded using the Scalable Vector Graphics format (http://www.w3.org/Graphics/SVG/);
  - `tex/` for the LaTeX source of this document.

[1] Faust assumes that $\forall\, s \in S, \forall\, t \in \mathbb{Z}, s(t) = 0$ when $t < 0$.

4 Faust code listings

This section provides the listings of the Faust code used to generate this document, including dependencies.

Listing 1: fmoog.dsp

```
declare filename "fmoog.dsp";
declare name "fmoog";
import("stdfaust.lib");

process = source(freq, delta) : drive(fmoog(fc));

T3 = -1/3 * checkbox("NL");

fmoog(fc) = _,0 : M : M : M : M : S with {
    F1 = *(a) : + ~ *(b) with {
        nu = 2 * ma.PI * fc / ma.SR;
        a = nu / (nu + 1);
        b = 1 / (nu + 1);
    };

    M(u1, u3) = F1(u1), F1(u3 + u1^3 - F1(u1)^3);

    S(y1, y3) = y1 + T3 * y3;
};

drive(C) = *(g) : C : /(g) with {
    g = vslider("drive[style:knob]", 1, 0.1, 20, 0.1);
};

source(f1, df) = os.square(f1) + os.square(f1 + df) : *(0.5);
freq = vslider("freq[style:knob][scale:log][unit:Hz]", 330, 20, 5000, 1);
delta = vslider("delta[style:knob][unit:Hz]", 0.1, 0.05, 2, 0.05);
fc = vslider("fc[style:knob][scale:log][unit:Hz]", 1000, 20, 5000, 1);
```

Listing 2: stdfaust.lib

```
//################################ stdfaust.lib ##########################################
// The purpose of this library is to give access to all the Faust standard libraries
// through a series of environments.
//########################################################################################

aa = library("aanl.lib");
sf = library("all.lib");
an = library("analyzers.lib");
ba = library("basics.lib");
co = library("compressors.lib");
de = library("delays.lib");
dm = library("demos.lib");
dx = library("dx7.lib");
en = library("envelopes.lib");
fd = library("fds.lib");
fi = library("filters.lib");
ho = library("hoa.lib");
it = library("interpolators.lib");
ma = library("maths.lib");
mi = library("mi.lib");
ef = library("misceffects.lib");
os = library("oscillators.lib");
```

```
no = library("noises.lib");
pf = library("phaflangers.lib");
pl = library("platform.lib");
pm = library("physmodels.lib");
qu = library("quantizers.lib");
rm = library("reducemaps.lib");
re = library("reverbs.lib");
ro = library("routes.lib");
sp = library("spats.lib");
si = library("signals.lib");
so = library("soundfiles.lib");
sy = library("synths.lib");
ve = library("vaeffects.lib");
vl = library("version.lib");
wa = library("webaudio.lib");
wd = library("wdmodels.lib");
```

Listing 3: `oscillators.lib`

```
//############################## oscillators.lib ######################################
// This library contains a collection of sound generators. Its official prefix is `os`.
//
// The oscillators library is organized into 9 sections:
//
// * [Wave-Table-Based Oscillators](#wave-table-based-oscillators)
// * [Low Frequency Oscillators](#low-frequency-oscillators)
// * [Low Frequency Sawtooths](#low-frequency-sawtooths)
// * [Alias-Suppressed Sawtooth](#alias-suppressed-sawtooth)
// * [Alias-Suppressed Pulse, Square, and Impulse Trains](#alias-suppressed-pulse-square-and-
    impulse-trains)
// * [Filter-Based Oscillators](#filter-based-oscillators)
// * [Waveguide-Resonator-Based Oscillators](#waveguide-resonator-based-oscillators)
// * [Casio CZ Oscillators](#casio-cz-oscillators)
// * [PolyBLEP-Based Oscillators](#polyblep-based-oscillators)
//
// #### References
// * <https://github.com/grame-cncm/faustlibraries/blob/master/oscillators.lib>
//########################################################################################

/************************************************************************
************************************************************************
FAUST library file, GRAME section

Except where noted otherwise, Copyright (C) 2003-2017 by GRAME,
Centre National de Creation Musicale.
----------------------------------------------------------------------
GRAME LICENSE

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation; either version 2.1 of the
License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with the GNU C Library; if not, write to the Free
Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
02111-1307 USA.

EXCEPTION TO THE LGPL LICENSE : As a special exception, you may create a
```

```
larger FAUST program which directly or indirectly imports this library
file and still distribute the compiled code generated by the FAUST
compiler, or a modified version of this compiled code, under your own
copyright and license. This EXCEPTION TO THE LGPL LICENSE explicitly
grants you the right to freely choose the license for the resulting
compiled code. In particular the resulting compiled code has no obligation
to be LGPL or GPL. For example you are free to choose a commercial or
closed source license or any other license if you decide so.
************************************************************************
************************************************************************/

ma = library("maths.lib");
ba = library("basics.lib");
fi = library("filters.lib");
si = library("signals.lib");

declare name "Faust Oscillator Library";
declare version "1.5.1";

// This library contains platform specific constants
pl = library("platform.lib");

//=====================Oscillators based on mathematical functions===============
//
// Note that there is a numerical problem with several phasor functions built using the internal
// `phasor_imp`. The reason is that the incremental step is smaller than `ma.EPSILON`, which happens with very small frequencies,
// so it will have no effect when summed to 1, but it will be enough to make the fractional function wrap
// around when summed to 0. An example of this problem can be observed when running the following code:
//
// `process = os.phasor(1.0, -.001);`
//
// The output of this program is the sequence 1, 0, 1, 0, 1... This happens because the negative incremental
// step is greater than `-ma.EPSILON`, which will have no effect when summed to 1, but it will be significant
// enough to make the fractional function wrap around when summed to 0.
//
// The incremental step can be clipped to guarantee that the phasor will
// always run correctly for its full cycle, otherwise, for increments smaller than `ma.EPSILON`,
// phasor would initially run but it'd eventually get stuck once the output gets big enough.
//
// All functions using `phasor_imp` are affected by this problem, but a safer
// version is implemented, and can be used alternatively by setting `SAFE=1` in the environment using
// [explicit sustitution](https://faustdoc.grame.fr/manual/syntax/#explicit-substitution) syntax.
//
// For example: `process = os[SAFE=1;].phasor(1.0, -.001);` will use the safer implementation of `phasor_imp`.
//================================================================================

//========================Wave-Table-Based Oscillators==================================
// Oscillators using tables. The table size is set by the
// [pl.tablesize](https://github.com/grame-cncm/faustlibraries/blob/master/platform.lib) constant.
//========================================================================================

// Global parameter to use the safer version of `phasor_imp`, but which
// could be used in other functions as well.

SAFE = 0; // 0: use the faster version, 1: use the safer version

//-----------------------`(os.)sinwaveform`------------------------
```

```
// Sine waveform ready to use with a ‘rdtable‘.
//
// #### Usage
//
// ‘‘‘
// sinwaveform(tablesize) : _
// ‘‘‘
//
// Where:
//
// * ‘tablesize‘: the table size
//------------------------------------------------------------
sinwaveform(tablesize) =
    sin(float(ba.period(tablesize)) * (2.0 * ma.PI) / float(tablesize));


//----------------------‘(os.)coswaveform‘-----------------------
// Cosine waveform ready to use with a ‘rdtable‘.
//
// #### Usage
//
// ‘‘‘
// coswaveform(tablesize) : _
// ‘‘‘
//
// Where:
//
// * ‘tablesize‘: the table size
//------------------------------------------------------------
coswaveform(tablesize) =
    cos(float(ba.period(tablesize)) * (2.0 * ma.PI) / float(tablesize));

// Possibly faster version using integer arithmetic
phasor_env(freq, N) = environment {

    //------- GLOBAL PARAMS
    nbits = 31;
    tablesize = 1<<N;
    accuracy = int(nbits - N);
    mask = (1<<nbits)-1;
    inc(step) = int(tablesize * step * (1<<accuracy));

    //------- LAMBDA DSP CASE
    lambda(inc_op) = (inc_op : &(mask)) ~ _ : >>(accuracy) : /(tablesize);

    //------- MINIMAL CASE
    hsp(0,0) = lambda(+(inc(freq/ma.SR)’));

    //------- GENERAL CASE
    hsp(reset,phase) = lambda(select2(hard_reset,+(inc(freq/ma.SR)),inc(phase)))
    with {
        hard_reset = (1-1’)|reset;
    };
};
declare phasor_env author "Pierre Mascarade Relano, Maxime Sirbu, Stéphane Letz";

// Generic phasor with ‘reset‘ and ‘phase‘ parameters to be specialised in concrete use-cases
    .
phasor_imp(freq, reset, phase) = (select2(hard_reset, +(incr(SAFE)), phase) : ma.decimal) ~
    _
with {
    incr_aux = freq/ma.SR;

    // Faster but less accurate version
    incr(0) = incr_aux;

    // To make sure that the incremental step is greater or equal to EPSILON or
    // less than or equal to -EPSILON to avoid numerical problems.
```

```
    // A frequency of 0Hz can still be used to freeze the phasor.
    incr(1)= (freq != 0) * ba.if(freq < 0, min(-1.0 * ma.EPSILON, incr_aux), max(ma.EPSILON,
        incr_aux));

    // To correctly start at `phase` at the first sample
    hard_reset = (1-1')|reset;
};

// Possibly faster version using integer arithmetic
// phasor_imp(freq, reset, phase) = phasor_env(freq, 16).hsp(reset, phase);

// Version to be used with tables
phasor_table(tablesize, freq, reset, phase) = phasor_imp(freq, reset, phase) : *(float(
    tablesize));


//-----------------------`(os.)phasor`------------------------
// A simple phasor to be used with a `rdtable`.
// `phasor` is a standard Faust function.
//
// #### Usage
//
// ```
// phasor(tablesize,freq) : _
// ```
//
// Where:
//
// * `tablesize`: the table size
// * `freq`: the frequency in Hz
//
// Note that `tablesize` is just a multiplier for the output of a unit-amp phasor
// so `phasor(1.0, freq)` can be used to generate a phasor output in the range [0, 1[.
//-----------------------------------------------------------
phasor(tablesize, freq) = phasor_table(tablesize, freq, 0, 0);


//-----------------------`(os.)hs_phasor`------------------------
// Hardsyncing phasor to be used with a `rdtable`.
//
// #### Usage
//
// ```
// hs_phasor(tablesize,freq,reset) : _
// ```
//
// Where:
//
// * `tablesize`: the table size
// * `freq`: the frequency in Hz
// * `reset`: a reset signal, reset phase to 0 when equal to 1
//---------------------------------------------------------
declare hs_phasor author "Mike Olsen, revised by Stéphane Letz";

hs_phasor(tablesize, freq, reset) = phasor_table(tablesize, freq, reset, 0);


//-----------------------`(os.)hsp_phasor`------------------------
// Hardsyncing phasor with selectable phase to be used with a `rdtable`.
//
// #### Usage
//
// ```
// hsp_phasor(tablesize,freq,reset,phase)
// ```
//
// Where:
//
```

```
// * `tablesize`: the table size
// * `freq`: the frequency in Hz
// * `reset`: reset the oscillator to phase when equal to 1
// * `phase`: phase between 0 and 1
//----------------------------------------------------------
declare hsp_phasor author "Christophe Lebreton, revised by Stéphane Letz";

hsp_phasor(tablesize, freq, reset, phase) = phasor_table(tablesize, freq, reset, phase);


//-----------------------`(os.)oscsin`------------------------
// Sine wave oscillator.
// `oscsin` is a standard Faust function.
//
// #### Usage
//
// ```
// oscsin(freq) : _
// ```
//
// Where:
//
// * `freq`: the frequency in Hz
//------------------------------------------------------------
oscsin(freq) = rdtable(tablesize, sinwaveform(tablesize), int(phasor(tablesize,freq)))
with {
    tablesize = pl.tablesize;
};


//-----------------------`(os.)hs_oscsin`------------------------
// Sin lookup table with hardsyncing phase.
//
// #### Usage
//
// ```
// hs_oscsin(freq,reset) : _
// ```
//
// Where:
//
// * `freq`: the frequency in Hz
// * `reset`: reset the oscillator to 0 when equal to 1
//---------------------------------------------------------
declare hs_oscsin author "Mike Olsen";

hs_oscsin(freq,reset) = rdtable(tablesize, sinwaveform(tablesize), int(hs_phasor(tablesize,
      freq,reset)))
with {
    tablesize = pl.tablesize;
};


//-----------------------`(os.)osccos`------------------------
// Cosine wave oscillator.
//
// #### Usage
//
// ```
// osccos(freq) : _
// ```
//
// Where:
//
// * `freq`: the frequency in Hz
//------------------------------------------------------------
osccos(freq) = rdtable(tablesize, coswaveform(tablesize), int(phasor(tablesize,freq)))
with {
```

```
    tablesize = pl.tablesize;
};


//-----------------------`(os.)hs_osccos`------------------------
// Cos lookup table with hardsyncing phase.
//
// #### Usage
//
// ```
// hs_osccos(freq,reset) : _
// ```
//
// Where:
//
// * `freq`: the frequency in Hz
// * `reset`: reset the oscillator to 0 when equal to 1
//----------------------------------------------------------
declare hs_osccos author "Stéphane Letz";

hs_osccos(freq,reset) = rdtable(tablesize, coswaveform(tablesize), int(hs_phasor(tablesize,
      freq,reset)))
with {
    tablesize = pl.tablesize;
};


//-----------------------`(os.)oscp`------------------------
// A sine wave generator with controllable phase.
//
// #### Usage
//
// ```
// oscp(freq,phase) : _
// ```
//
// Where:
//
// * `freq`: the frequency in Hz
// * `phase`: the phase in radian
//------------------------------------------------------------
oscp(freq,phase) = oscsin(freq) * cos(phase) + osccos(freq) * sin(phase);


//-----------------------`(os.)osci`------------------------
// Interpolated phase sine wave oscillator.
//
// #### Usage
//
// ```
// osci(freq) : _
// ```
//
// Where:
//
// * `freq`: the frequency in Hz
//------------------------------------------------------------
osci(freq) = s1 + d * (s2 - s1)
with {
    tablesize = pl.tablesize;
    i = int(phasor(tablesize,freq));
    d = ma.decimal(phasor(tablesize,freq));
    s1 = rdtable(tablesize+1,sinwaveform(tablesize),i);
    s2 = rdtable(tablesize+1,sinwaveform(tablesize),i+1);
};


//-----------------------`(os.)osc`------------------------
```

```
// Default sine wave oscillator (same as [oscsin](#oscsin)).
// `osc` is a standard Faust function.
//
// #### Usage
//
// ```
// osc(freq) : _
// ```
//
// Where:
//
// * `freq`: the frequency in Hz
//------------------------------------------------------------
osc = oscsin;


//----------------------`(os.)m_oscsin`-----------------------
// Sine wave oscillator based on the `sin` mathematical function.
//
// #### Usage
//
// ```
// m_oscsin(freq) : _
// ```
//
// Where:
//
// * `freq`: the frequency in Hz
//------------------------------------------------------------
m_oscsin(freq) = lf_sawpos(freq) : *(2*ma.PI) : sin;


//----------------------`(os.)m_osccos`-----------------------
// Sine wave oscillator based on the `cos` mathematical function.
//
// #### Usage
//
// ```
// m_osccos(freq) : _
// ```
//
// Where:
//
// * `freq`: the frequency in Hz
//------------------------------------------------------------
m_osccos(freq) = lf_sawpos(freq) : *(2*ma.PI) : cos;


// end GRAME section
//#########################################################################################
/************************************************************************
FAUST library file, jos section

Except where noted otherwise, The Faust functions below in this
section are Copyright (C) 2003-2022 by Julius O. Smith III <jos@ccrma.stanford.edu>
([jos](http://ccrma.stanford.edu/~jos/)), and released under the
(MIT-style) [STK-4.3](#stk-4.3-license) license.

The MarkDown comments in this section are Copyright 2016-2017 by Romain
Michon and Julius O. Smith III, and are released under the
[CCA4I](https://creativecommons.org/licenses/by/4.0/) license (TODO: if/when Romain agrees)

************************************************************************/

//==============================Low Frequency Oscillators==============================
// Low Frequency Oscillators (LFOs) have prefix `lf_`
// (no aliasing suppression, since it is inaudible at LF).
// Use `sawN` and its derivatives for audio oscillators with suppressed aliasing.
```

```
//===============================================================

//--------`(os.)lf_imptrain`----------
// Unit-amplitude low-frequency impulse train.
// `lf_imptrain` is a standard Faust function.

// #### Usage
//
// ```
// lf_imptrain(freq) : _
// ```

// Where:
//
// * `freq`: frequency in Hz
//------------------------------------------------------------
lf_imptrain(freq) = lf_sawpos(freq)<:-(mem)<0; // definition below


//--------`(os.)lf_pulsetrainpos`----------
// Unit-amplitude nonnegative LF pulse train, duty cycle between 0 and 1.
//
//
// #### Usage
//
// ```
// lf_pulsetrainpos(freq, duty) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
// * `duty`: duty cycle between 0 and 1
//------------------------------------------------------------
lf_pulsetrainpos(freq,duty) = float(lf_sawpos(freq) <= duty);

//pulsetrainpos = lf_pulsetrainpos; // for backward compatibility


//--------`(os.)lf_pulsetrain`----------
// Unit-amplitude zero-mean LF pulse train, duty cycle between 0 and 1.
//
// #### Usage
//
// ```
// lf_pulsetrain(freq,duty) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
// * `duty`: duty cycle between 0 and 1
//------------------------------------------------------------
lf_pulsetrain(freq,duty) = 2.0*lf_pulsetrainpos(freq,duty) - 1.0;


//--------`(os.)lf_squarewavepos`----------
// Positive LF square wave in [0,1]
//
// #### Usage
//
// ```
// lf_squarewavepos(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
```

```
//-----------------------------------------------------------
lf_squarewavepos(freq) = lf_pulsetrainpos(freq,0.5);
// squarewavepos = lf_squarewavepos; // for backward compatibility


//--------`(os.)lf_squarewave`----------
// Zero-mean unit-amplitude LF square wave.
// `lf_squarewave` is a standard Faust function.
//
// #### Usage
//
// ```
// lf_squarewave(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//-----------------------------------------------------------
lf_squarewave(freq) = 2.0*lf_squarewavepos(freq) - 1.0;
// squarewave = lf_squarewave; // for backward compatibility


//--------`(os.)lf_trianglepos`----------
// Positive unit-amplitude LF positive triangle wave.
//
// #### Usage
//
// ```
// lf_trianglepos(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//-----------------------------------------------------------
lf_trianglepos(freq) = 1.0-abs(saw1(freq)); // saw1 defined below


//----------`(os.)lf_triangle`----------
// Zero-mean unit-amplitude LF triangle wave.
// `lf_triangle` is a standard Faust function.
//
// #### Usage
//
// ```
// lf_triangle(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//-----------------------------------------------------------
declare lf_triangle author "Bart Brouns";
declare lf_triangle licence "STK-4.3";

lf_triangle(freq) = 2.0*lf_trianglepos(freq) - 1.0;


//================== Low Frequency Sawtooths ====================
// Sawtooth waveform oscillators for virtual analog synthesis et al.
// The 'simple' versions (`lf_rawsaw`, `lf_sawpos` and `saw1`), are mere samplings of
// the ideal continuous-time ("analog") waveforms. While simple, the
// aliasing due to sampling is quite audible. The differentiated
// polynomial waveform family (`saw2`, `sawN`, and derived functions)
// do some extra processing to suppress aliasing (not audible for
// very low fundamental frequencies). According to Lehtonen et al.
// (JASA 2012), the aliasing of `saw2` should be inaudible at fundamental
```

```
// frequencies below 2 kHz or so, for a 44.1 kHz sampling rate and 60 dB SPL
// presentation level; fundamentals 415 and below required no aliasing
// suppression (i.e., `saw1` is ok).
//=====================================================================

//-----------------`(os.)lf_rawsaw`--------------------
// Simple sawtooth waveform oscillator between 0 and period in samples.
//
// #### Usage
//
// ```
// lf_rawsaw(periodsamps) : _
// ```
//
// Where:
//
// * `periodsamps`: number of periods per samples
//---------------------------------------------------------
lf_rawsaw(periodsamps) = (_,periodsamps : fmod) ~ +(1.0);


//-----------------`(os.)lf_sawpos`--------------------
// Simple sawtooth waveform oscillator between 0 and 1.
//
// #### Usage
//
// ```
// lf_sawpos(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//
//---------------------------------------------------------
declare lf_sawpos author "Bart Brouns, revised by Stéphane Letz";
declare lf_sawpos licence "STK-4.3";

lf_sawpos(freq) = phasor_imp(freq, 0, 0);


//-----------------`(os.)lf_sawpos_phase`--------------------
// Simple sawtooth waveform oscillator between 0 and 1
// with phase control.
//
// #### Usage
//
// ```
// lf_sawpos_phase(freq, phase) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
// * `phase`: phase between 0 and 1
//---------------------------------------------------------
declare lf_sawpos_phase author "Bart Brouns, revised by Stéphane Letz";
declare lf_sawpos_phase licence "STK-4.3";

lf_sawpos_phase(freq,phase) = phasor_imp(freq, 0, phase);


//-----------------`(os.)lf_sawpos_reset`--------------------
// Simple sawtooth waveform oscillator between 0 and 1
// with reset.
//
// #### Usage
//
```

```
// ```
// lf_sawpos_reset(freq,reset) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
// * `reset`: reset the oscillator to 0 when equal to 1
//
//---------------------------------------------------------
declare lf_sawpos_reset author "Bart Brouns, revised by Stéphane Letz";
declare lf_sawpos_reset licence "STK-4.3";

lf_sawpos_reset(freq,reset) = phasor_imp(freq, reset, 0);


//-----------------`(os.)lf_sawpos_phase_reset`--------------------
// Simple sawtooth waveform oscillator between 0 and 1
// with phase control and reset.
//
// #### Usage
//
// ```
// lf_sawpos_phase_reset(freq,phase,reset) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
// * `phase`: phase between 0 and 1
// * `reset`: reset the oscillator to phase when equal to 1
//
//---------------------------------------------------------
declare lf_sawpos_phase_reset author "Bart Brouns, revised by Stéphane Letz";
declare lf_sawpos_phase_reset licence "STK-4.3";

lf_sawpos_phase_reset(freq,phase,reset) = phasor_imp(freq, reset, phase);


//-----------------`(os.)lf_saw`--------------------
// Simple sawtooth waveform oscillator between -1 and 1.
// `lf_saw` is a standard Faust function.
//
// #### Usage
//
// ```
// lf_saw(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//---------------------------------------------------------
declare saw1 author "Bart Brouns";
declare saw1 licence "STK-4.3";

saw1(freq) = 2.0 * lf_sawpos(freq) - 1.0;
lf_saw(freq) = saw1(freq);

//================= Alias-Suppressed Sawtooth ===================
//-----------------`(os.)sawN`--------------------
// Alias-Suppressed Sawtooth Audio-Frequency Oscillator using Nth-order polynomial
      transitions
// to reduce aliasing.
//
// `sawN(N,freq)`, `sawNp(N,freq,phase)`, `saw2dpw(freq)`, `saw2(freq)`, `saw3(freq)`,
// `saw4(freq)`, `sawtooth(freq)`, `saw2f2(freq)`, `saw2f4(freq)`
//
```

```
// #### Usage
//
// ```
// sawN(N,freq) : _     // Nth-order aliasing-suppressed sawtooth using DPW method (see
     below)
// sawNp(N,freq,phase) : _ // sawN with phase offset feature
// saw2dpw(freq) : _    // saw2 using DPW
// saw2ptr(freq) : _    // saw2 using the faster, stateless PTR method
// saw2(freq) : _       // DPW method, but subject to change if a better method emerges
// saw3(freq) : _       // sawN(3)
// saw4(freq) : _       // sawN(4)
// sawtooth(freq) : _   // saw2
// saw2f2(freq) : _     // saw2dpw with 2nd-order droop-correction filtering
// saw2f4(freq) : _     // saw2dpw with 4th-order droop-correction filtering
// ```
//
// Where:
//
// * `N`: polynomial order, a constant numerical expression between 1 and 4
// * `freq`: frequency in Hz
// * `phase`: phase between 0 and 1
//
// #### Method
// Differentiated Polynomial Wave (DPW).
//
// ##### Reference
// "Alias-Suppressed Oscillators based on Differentiated Polynomial Waveforms",
// Vesa Valimaki, Juhan Nam, Julius Smith, and Jonathan Abel,
// IEEE Tr. Audio, Speech, and Language Processing (IEEE-ASLP),
// Vol. 18, no. 5, pp 786-798, May 2010.
// 10.1109/TASL.2009.2026507.
//
// #### Notes
// The polynomial order `N` is limited to 4 because noise has been
// observed at very low `freq` values. (LFO sawtooths should of course
// be generated using `lf_sawpos` instead.)
//------------------------------------------------------------------
declare sawN author "Julius O. Smith III";
declare sawN license "STK-4.3";
// --- sawN for N = 1 to 4 ---
// Orders 5 and 6 have noise at low fundamentals: MAX_SAW_ORDER = 6; MAX_SAW_ORDER_NEXTPOW2
     = 8;
MAX_SAW_ORDER = 4;
MAX_SAW_ORDER_NEXTPOW2 = 8; // par cannot handle the case of 0 elements
sawN(N,freq) = saw1l : poly(Nc) : D(Nc-1) : gate(Nc-1)
with {
  Nc = max(1,min(N,MAX_SAW_ORDER));
  clippedFreq = max(20.0,abs(freq)); // use lf_sawpos(freq) for LFOs (freq < 20 Hz)
  saw1l = 2*lf_sawpos(clippedFreq) - 1; // zero-mean, amplitude +/- 1
  poly(1,x) = x;
  poly(2,x) = x*x;
  poly(3,x) = x*x*x - x;
  poly(4,x) = x*x*(x*x - 2.0);
  poly(5,x) = x*(7.0/3 + x*x*(-10.0/3.0 + x*x));
  poly(6,x) = x*x*(7.0 + x*x*(-5.0 + x*x));
  p0n = float(ma.SR)/clippedFreq; // period in samples
  diff1(x) = (x - x')/(2.0/p0n);
  diff(N) = seq(n,N,diff1); // N diff1s in series
  factorial(0) = 1;
  factorial(i) = i * factorial(i-1);
  D(0) = _;
  D(i) = diff(i)/factorial(i+1);
  gate(N) = *(1@(N)); // delayed step for blanking startup glitch
};

//-----------------`(os.)sawNp`--------------------------------
// Same as `(os.)sawN` but with a controllable waveform phase.
//
```

```
// #### Usage
//
// ```
// sawNp(N,freq,phase) : _
// ```
//
// where
//
// * `N`: waveform interpolation polynomial order 1 to 4 (constant integer expression)
// * `freq`: frequency in Hz
// * `phase`: waveform phase as a fraction of one period (rounded to nearest sample)
//
// #### Implementation Notes
//
// The phase offset is implemented by delaying `sawN(N,freq)` by
// `round(phase*ma.SR/freq)` samples, for up to 8191 samples.
// The minimum sawtooth frequency that can be delayed a whole period
// is therefore `ma.SR/8191`, which is well below audibility for normal
// audio sampling rates.
//
//-----------------------------------------------------------------
declare sawNp author "Julius O. Smith III";
declare sawNp license "STK-4.3";
// --- sawNp for N = 1 to 4 ---
// Phase offset = delay (max 8191 samples is more than one period of audio):
sawNp(N,freq,phase) = sawN(N,freq) : @(max(0,min(8191,int(0.5+phase*ma.SR/freq))));

//-----------------`(os.)saw2, (os.)saw3, (os.)saw4`-------------
// Alias-Suppressed Sawtooth Audio-Frequency Oscillators of order 2, 3, 4.
//
// #### Usage
//
// ```
// saw2(freq) : _
// saw3(freq) : _
// saw4(freq) : _
// ```
//
// where
//
// * `freq`: frequency in Hz
//
// ##### References
// See `sawN` above.
//
// #### Implementation Notes
//
// Presently, only `saw2` uses the PTR method, while `saw3` and `saw4` use DPW.
// This is because PTR has been implemented and tested for the 2nd-order case only.
//
//------------------------------------------------------------------
saw2 = saw2ptr; // "faustlibraries choice"
saw3 = sawN(3); // only choice available right now
saw4 = sawN(4); // only choice available right now

//-------------------------`(os.)saw2ptr`--------------------------
// Alias-Suppressed Sawtooth Audio-Frequency Oscillator
// using Polynomial Transition Regions (PTR) for order 2.
//
// #### Usage
//
// ```
// saw2ptr(freq) : _
// ```
//
// where
//
// * `freq`: frequency in Hz
```

```
//
// ##### Implementation
//
// Polynomial Transition Regions (PTR) method for aliasing suppression.
//
// ##### References
//
// * Kleimola, J.; Valimaki, V., "Reducing Aliasing from Synthetic Audio
//    Signals Using Polynomial Transition Regions," in Signal Processing
//    Letters, IEEE , vol.19, no.2, pp.67-70, Feb. 2012
// * <https://aaltodoc.aalto.fi/bitstream/handle/123456789/7747/publication6.pdf?sequence=9>
// * <http://research.spa.aalto.fi/publications/papers/spl-ptr/>
//
// ##### Notes
//
// Method PTR may be preferred because it requires less
// computation and is stateless which means that the frequency `freq`
// can be modulated arbitrarily fast over time without filtering
// artifacts. For this reason, `saw2` is presently defined as `saw2ptr`.
//
//---------------------------------------------------------
declare saw2ptr author "Julius O. Smith III";
declare saw2ptr license "STK-4.3";
// specialized reimplementation:
saw2ptr(freq) = y with { // newer PTR version (stateless - freq can vary at any speed)
  p0 = float(ma.SR)/float(max(ma.EPSILON,abs(freq))); // period in samples
  t0 = 1.0/p0; // phase increment
  p = ((_<:(-(1)<:_,_),_) <: selector1,selector2) ~(+(t0)):!,_;
  selector1 = select2(<(0)); // for feedback
  selector2 = select2(<(0), (_<:_,(*(1-p0):+(1)):+), _); // for output
  y = 2*p-1;
};

//----------------------`(os.)saw2dpw`---------------------
// Alias-Suppressed Sawtooth Audio-Frequency Oscillator
// using the Differentiated Polynomial Waveform (DWP) method.
//
// #### Usage
//
// ```
// saw2dpw(freq) : _
// ```
//
// where
//
// * `freq`: frequency in Hz
//
// This is the original Faust `saw2` function using the DPW method.
// Since `saw2` is now defined as `saw2ptr`, the DPW version
// is now available as `saw2dwp`.
//---------------------------------------------------------
declare saw2dpw author "Julius O. Smith III";
declare saw2dpw license "STK-4.3";
saw2dpw(freq) = saw1(freq) <: * <: -(mem) : *(0.25'*ma.SR/freq);

//------------------`(os.)sawtooth`--------------------------------
// Alias-suppressed aliasing-suppressed sawtooth oscillator, presently defined as `saw2`.
// `sawtooth` is a standard Faust function.
//
// #### Usage
//
// ```
// sawtooth(freq) : _
// ```
//
// with
//
// * `freq`: frequency in Hz
```

```
//-----------------------------------------------------------
sawtooth = saw2; // default choice for sawtooth signal - see also sawN

//-----------------`(os.)saw2f2, (os.)saw2f4`---------------------------------
// Alias-Suppressed Sawtooth Audio-Frequency Oscillator with Order 2 or 4 Droop Correction Filtering.
//
// #### Usage
//
// ```
// saw2f2(freq) : _
// saw2f4(freq) : _
// ```
//
// with
//
// * `freq`: frequency in Hz
//
// In return for aliasing suppression, there is some attenuation near half the sampling rate.
// This can be considered as beneficial, or it can be compensated with a high-frequency boost.
// The boost filter is second-order for `saw2f2` and fourth-order for `saw2f4`, and both are designed
// for the DWP case and therefore use `saw2dpw`.
// See Figure 4(b) in the DPW reference for a plot of the slight droop in the DPW case.
//-----------------------------------------------------------
declare saw2f2 author "Julius O. Smith III";
declare saw2f2 license "STK-4.3";
// --- Correction-filtered versions of saw2: saw2f2, saw2f4 -----
saw2f2 = saw2dpw : cf2 with {
  cf2 = fi.tf2(1.155704605878911, 0.745184288225518,0.040305967265900,
       0.823765146386639, 0.117420665547108);
};
declare saw2f4 author "Julius O. Smith III";
declare saw2f4 license "STK-4.3";
saw2f4 = saw2dpw : cf4 with {
  cf4 = fi.iir((1.155727435125014, 2.285861038554662,
       1.430915027294021, 0.290713280893317, 0.008306401748854),
       (2.156834679164532, 1.559532244409321, 0.423036498118354,
       0.032080681130972));
};

//=========Alias-Suppressed Pulse, Square, and Impulse Trains============
// Alias-Suppressed Pulse, Square and Impulse Trains.
//
// `pulsetrainN`, `pulsetrain`, `squareN`, `square`, `imptrainN`, `imptrain`,
// `triangleN`, `triangle`
//
// All are zero-mean and meant to oscillate in the audio frequency range.
// Use simpler sample-rounded `lf_*` versions above for LFOs.
//
// #### Usage
//
// ```
// pulsetrainN(N,freq,duty) : _
// pulsetrain(freq, duty) : _ // = pulsetrainN(2)
//
// squareN(N,freq) : _
// square : _ // = squareN(2)
//
// imptrainN(N,freq) : _
// imptrain : _ // = imptrainN(2)
//
// triangleN(N,freq) : _
// triangle : _ // = triangleN(2)
// ```
//
```

```
// Where:
//
// * `N`: polynomial order, a constant numerical expression
// * `freq`: frequency in Hz
//===================================================================


//------------------`(os.)impulse`--------------------------------
// One-time impulse generated when the Faust process is started.
// `impulse` is a standard Faust function.
//
// #### Usage
//
// ```
// impulse : _
// ```
//--------------------------------------------------------
impulse = 1-1';


//------------------`(os.)pulsetrainN`--------------------------------
// Alias-suppressed pulse train oscillator.
//
// #### Usage
//
// ```
// pulsetrainN(N,freq,duty) : _
// ```
//
// Where:
//
// * `N`: order, as a constant numerical expression
// * `freq`: frequency in Hz
// * `duty`: duty cycle between 0 and 1
//--------------------------------------------------------
pulsetrainN(N,freq,duty) = diffdel(sawN(N,freqC),del) with {
 // non-interpolated-delay version: diffdel(x,del) = x - x@int(del+0.5);
 // linearly interpolated delay version (sounds good to me):
    diffdel(x,del) = x-x@int(del)*(1-ma.frac(del))-x@(int(del)+1)*ma.frac(del);
 // Third-order Lagrange interpolated-delay version (see filters.lib):
 // diffdel(x,del) = x - fdelay3(DELPWR2,max(1,min(DELPWR2-2,ddel)));
 DELPWR2 = 2048; // Needs to be a power of 2 when fdelay*() used above.
 delmax = DELPWR2-1; // arbitrary upper limit on diff delay (duty=0.5)
 SRmax = 96000.0; // assumed upper limit on sampling rate
 fmin = SRmax / float(2.0*delmax); // 23.4 Hz (audio freqs only)
 freqC = max(freq,fmin); // clip frequency at lower limit
 period = (float(ma.SR) / freqC); // actual period
 ddel = duty * period; // desired delay
 del = max(0,min(delmax,ddel));
};


//------------------`(os.)pulsetrain`--------------------------------
// Alias-suppressed pulse train oscillator. Based on `pulsetrainN(2)`.
// `pulsetrain` is a standard Faust function.
//
// #### Usage
//
// ```
// pulsetrain(freq,duty) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
// * `duty`: duty cycle between 0 and 1
//--------------------------------------------------------
```

```
pulsetrain = pulsetrainN(2);


//-----------------`(os.)squareN`-------------------------------
// Alias-suppressed square wave oscillator.
//
// #### Usage
//
// ```
// squareN(N,freq) : _
// ```
//
// Where:
//
// * `N`: order, as a constant numerical expression
// * `freq`: frequency in Hz
//---------------------------------------------------------
squareN(N,freq) = pulsetrainN(N,freq,0.5);


//-----------------`(os.)square`-------------------------------
// Alias-suppressed square wave oscillator. Based on `squareN(2)`.
// `square` is a standard Faust function.
//
// #### Usage
//
// ```
// square(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//---------------------------------------------------------
square = squareN(2);


//-----------------`(os.)imptrainN`-------------------------------
// Alias-suppressed impulse train generator.
//
// #### Usage
//
// ```
// imptrainN(N,freq) : _
// ```
//
// Where:
//
// * `N`: order, as a constant numerical expression
// * `freq`: frequency in Hz
//---------------------------------------------------------
imptrainN(N,freq) = impulse + 0.5*ma.diffn(sawN(N,freq));


//-----------------`(os.)imptrain`-------------------------------
// Alias-suppressed impulse train generator. Based on `imptrainN(2)`.
// `imptrain` is a standard Faust function.
//
// #### Usage
//
// ```
// imptrain(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//---------------------------------------------------------
```

```
imptrain = imptrainN(2); // default based on saw2


//-----------------`(os.)triangleN`-------------------------------
// Alias-suppressed triangle wave oscillator.
//
// #### Usage
//
// ```
// triangleN(N,freq) : _
// ```
//
// Where:
//
// * `N`: order, as a constant numerical expression
// * `freq`: frequency in Hz
//-------------------------------------------------------
triangleN(N,freq) = squareN(N,freq) : fi.pole(p) : *(gain) with {
  gain = 4.0*freq/ma.SR; // for aproximate unit peak amplitude
  p = 0.999;
};


//-----------------`(os.)triangle`-------------------------------
// Alias-suppressed triangle wave oscillator. Based on `triangleN(2)`.
// `triangle` is a standard Faust function.
//
// #### Usage
//
// ```
// triangle(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//-------------------------------------------------------
triangle = triangleN(2); // default based on saw2


//=============================Filter-Based Oscillators================================
// Filter-Based Oscillators.
//
// #### Usage
//
// ```
// osc[b|rq|rs|rc|s](freq), where freq = frequency in Hz.
// ```
//
// #### References
//
// * <http://lac.linuxaudio.org/2012/download/lac12-slides-jos.pdf>
// * <https://ccrma.stanford.edu/~jos/pdf/lac12-paper-jos.pdf>
//====================================================================================

//-------------------------`(os.)oscb`-------------------------------
// Sinusoidal oscillator based on the biquad.
//
// #### Usage
//
// ```
// oscb(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//------------------------------------------------------------
```

```
oscb(f) = impulse : fi.tf2(1,0,0,a1,1)
with {
  a1 = -2*cos(2*ma.PI*f/ma.SR);
};


//-------------------------`(os.)oscrq`---------------------------
// Sinusoidal (sine and cosine) oscillator based on 2D vector rotation,
//  = undamped "coupled-form" resonator
//  = lossless 2nd-order normalized ladder filter.
//
// #### Usage
//
// ```
// oscrq(freq) : _,_
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//
// #### Reference
//
// * <https://ccrma.stanford.edu/~jos/pasp/Normalized_Scattering_Junctions.html>
//------------------------------------------------------------
oscrq(f) = impulse : fi.nlf2(f,1); // sine and cosine outputs

//-------------------------`(os.)oscrs`---------------------------
// Sinusoidal (sine) oscillator based on 2D vector rotation,
//  = undamped "coupled-form" resonator
//  = lossless 2nd-order normalized ladder filter.
//
// #### Usage
//
// ```
// oscrs(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//
// #### Reference
//
// * <https://ccrma.stanford.edu/~jos/pasp/Normalized_Scattering_Junctions.html>
//------------------------------------------------------------
oscrs(f) = impulse : fi.nlf2(f,1) : _,!; // sine

//-------------------------`(os.)oscrc`---------------------------
// Sinusoidal (cosine) oscillator based on 2D vector rotation,
//  = undamped "coupled-form" resonator
//  = lossless 2nd-order normalized ladder filter.
//
// #### Usage
//
// ```
// oscrc(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//
// #### Reference
//
// * <https://ccrma.stanford.edu/~jos/pasp/Normalized_Scattering_Junctions.html>
//------------------------------------------------------------
oscrc(f) = impulse : fi.nlf2(f,1) : !,_; // cosine
```

```
oscrp(f,p) = oscrq(f) : *(cos(p)), *(sin(p)) : + ; // p=0 for sine, p=PI/2 for cosine, etc.

oscr = oscrs; // default = sine (starts without a pop)

//-------------------------`(os.)oscs`-------------------------------
// Sinusoidal oscillator based on the state variable filter
// = undamped "modified-coupled-form" resonator
// = "magic circle" algorithm used in graphics.
//
// #### Usage
//
// ```
// oscs(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//------------------------------------------------------------
oscs(f) = (*(-1) : sint(wn) : sintp(wn,impulse)) ~ _
with {
  wn = 2*ma.PI*f/ma.SR; // approximate
  // wn = 2*sin(PI*f/SR); // exact
  sint(x) = *(x) : + ~ _ ; // frequency-scaled integrator
  sintp(x,y) = *(x) : +(y): + ~ _; // same + state input
};

//-----------------`(os.)quadosc`--------------------
// Quadrature (cosine and sine) oscillator based on QuadOsc by Martin Vicanek.
//
// #### Usage
//
// ```
// quadosc(freq) : _,_
// ```
//
// where
//
// * `freq`: frequency in Hz
//
// #### Reference
// * <https://vicanek.de/articles/QuadOsc.pdf>
//------------------------------------------------------------
// Authors:
// Dario Sanfilippo <sanfilippo.dario@gmail.com>
// and Oleg Nesterov (jos ed.)
quadosc(f) = tick ~ (_,_)
with {
    k1 = tan(f * ma.PI / ma.SR);
    k2 = 2 * k1 / (1 + k1 * k1);
    tick(u_0,v_0) = u_1,v_1
    with {
        tmp = u_0 - k1 * v_0;
        v_1 = v_0 + k2 * tmp;
        u_1 = tmp - k1 * v_1 : select2(1',1);
    };
};

//----------------------------`(os.)sidebands`--------------------------------------
// Adds harmonics to quad oscillator.
//
// #### Usage
//
// ```
//    cos(x),sin(x) : sidebands(vs) : _,_
// ```
//
```

```
// Where:
//
// * `vs` : list of amplitudes
//
// #### Example test program
//
// ```
//   cos(x),sin(x) : sidebands((10,20,30))
// ```
//
// outputs:
//
// ```
//   10*cos(x) + 20*cos(2*x) + 30*cos(3*x),
//   10*sin(x) + 20*sin(2*x) + 30*sin(3*x);
// ```
//
// The following:
//
// ```
//   process = os.quadosc(F) : sidebands((10,20,30))
// ```
//
// is (modulo floating point issues) the same as:
//
// ```
//   c = os.quadosc : _,!;
//   s = os.quadosc : !,_;
//   process =
//       10*c(F) + 20*c(2*F) + 30*c(F),
//       10*s(F) + 20*s(2*F) + 30*s(F);
// ```
//
// but much more efficient.
//
// #### Implementation Notes
//
// This is based on the trivial trigonometric identities:
//
// ```
//   cos((n + 1) x) = 2 cos(x) cos(n x) - cos((n - 1) x)
//   sin((n + 1) x) = 2 cos(x) sin(n x) - sin((n - 1) x)
// ```
//
// Note that the calculation of the cosine/sine parts do not depend
// on each other, so if you only need the sine part you can do:
//
// ```
//   process = os.quadosc(F) : sidebands(vs) : !,_;
// ```
//
// and the compiler will discard the half of the calculations.
//---------------------------------------------------------------------------
sidebands(vs, c0,s0)
    = c0*vn(0),s0*vn(0), 1,c0, 0,s0
    : seq(n, outputs(vs)-1, add(vn(n+1)))
    : _,_, !,!, !,!
with {
    // ba.take(n+1, vs)
    vn(n) = vs : route(outputs(vs),1, n+1,1);

    add(vn, co,so, cn_2,cn_1, sn_2,sn_1) =
        co+cn*vn, so+sn*vn, cn_1,cn, sn_1,sn
    with {
        cn = 2*c0*cn_1 - cn_2;
        sn = 2*c0*sn_1 - sn_2;
    };
};
```

```
//-----------------------------`(os.)sidebands_list`-------------------------------------
// Creates the list of complex harmonics from quad oscillator.
//
// Similar to `sidebands` but doesn't sum the harmonics, so it is more
// generic but less convenient for immediate usage.
//
// #### Usage
//
// ```
//    cos(x),sin(x) : sidebands_list(N) : si.bus(2*N)
// ```
//
// Where:
//
// * `N` : number of harmonics, compile time constant > 1
//
// #### Example test program
//
// ```
//    cos(x),sin(x) : sidebands_list(3)
// ```
//
// outputs:
//
// ```
//    cos(x),sin(x), cos(2*x),sin(2*x), cos(3*x),sin(3*x);
// ```
//
// The following:
//
// ```
//    process = os.quadosc(F) : sidebands_list(3)
// ```
//
// is (modulo floating point issues) the same as:
//
// ```
//    process = os.quadosc(F), os.quadosc(2*F), os.quadosc(3*F);
// ```
//
// but much more efficient.
//----------------------------------------------------------------------------
sidebands_list(N, c0,s0)
    = c0,s0, 1,c0, 0,s0
    : seq(n, N-1, si.bus(2*(n+1)), add)
    : si.bus(2*N), !,!, !,!
with {
    add(cn_2,cn_1, sn_2,sn_1) =
        cn,sn, cn_1,cn, sn_1,sn
    with {
        cn = 2*c0*cn_1 - cn_2;
        sn = 2*c0*sn_1 - sn_2;
    };
};

//================ Waveguide-Resonator-Based Oscillators ================
// Sinusoidal oscillator based on the waveguide resonator `wgr`.
//=======================================================================

//-----------------`(os.)oscwc`--------------------
// Sinusoidal oscillator based on the waveguide resonator `wgr`. Unit-amplitude
// cosine oscillator.
//
// #### Usage
//
// ```
// oscwc(freq) : _
```

```
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//
// #### Reference
//
// * <https://ccrma.stanford.edu/~jos/pasp/Digital_Waveguide_Oscillator.html>
//------------------------------------------------------------
oscwc(fr) = impulse : fi.wgr(fr,1) : _,!; // cosine (cheapest at 1 mpy/sample)

//-----------------`(os.)oscws`--------------------
// Sinusoidal oscillator based on the waveguide resonator `wgr`. Unit-amplitude
// sine oscillator.
//
// #### Usage
//
// ```
// oscws(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//
// #### Reference
//
// * <https://ccrma.stanford.edu/~jos/pasp/Digital_Waveguide_Oscillator.html>
//------------------------------------------------------------
oscws(fr) = impulse : fi.wgr(fr,1) : !,_; // sine (needs a 2nd scaling mpy)

//-----------------`(os.)oscq`--------------------
// Sinusoidal oscillator based on the waveguide resonator `wgr`.
// Unit-amplitude cosine and sine (quadrature) oscillator.
//
// #### Usage
//
// ```
// oscq(freq) : _,_
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//
// #### Reference
//
// * <https://ccrma.stanford.edu/~jos/pasp/Digital_Waveguide_Oscillator.html>
//------------------------------------------------------------
oscq(fr) = impulse : fi.wgr(fr,1);   // phase quadrature outputs

//-----------------`(os.)oscw`--------------------
// Sinusoidal oscillator based on the waveguide resonator `wgr`.
// Unit-amplitude cosine oscillator (default).
//
// #### Usage
//
// ```
// oscw(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//
// #### Reference
//
```

```
// * <https://ccrma.stanford.edu/~jos/pasp/Digital_Waveguide_Oscillator.html>
//------------------------------------------------------------
oscw = oscwc;

// end jos section
//#########################################################################################
/************************************************************************
FAUST library file, further contributions section

All contributions below should indicate both the contributor and terms
of license. If no such indication is found, "git blame" will say who
last edited each line, and that person can be emailed to inquire about
license disposition, if their license choice is not already indicated
elsewhere among the libraries. It is expected that all software will be
released under LGPL, STK-4.3, MIT, BSD, or a similar FOSS license.
************************************************************************/

//==================== Casio CZ Oscillators ==========================
// Oscillators that mimic some of the Casio CZ oscillators.
//
// There are two sets:
//
// * a set with an index parameter
//
// * a set with a res parameter
//
// The "index oscillators" outputs a sine wave at index=0 and gets brighter with a higher
     index.
// There are two versions of the "index oscillators":
//
// * with P appended to the name: is phase aligned with `fund:sin`
//
// * without P appended to the name: has the phase of the original CZ oscillators
//
// The "res oscillators" have a resonant frequency.
// "res" is the frequency of resonance as a factor of the fundamental pitch.
//
// For the `fund` waveform, use a low-frequency oscillator without anti-aliasing such as `os.
     lf_saw`.
//====================================================================

//----------`(os.)CZsaw`----------
// Oscillator that mimics the Casio CZ saw oscillator.
// `CZsaw` is a standard Faust function.
//
// #### Usage
//
// ```
// CZsaw(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 to 1. 0 = sine-wave, 1 = saw-wave
//------------------------------------------------------------
declare CZsaw author "Bart Brouns";
declare CZsaw licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZsaw(fund, index) = CZ.sawChooseP(fund, index, 0);

//----------`(os.)CZsawP`----------
// Oscillator that mimics the Casio CZ saw oscillator,
// with it's phase aligned to `fund:sin`.
```

```
// `CZsawP` is a standard Faust function.
//
// #### Usage
//
// ```
// CZsawP(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 to 1. 0 = sine-wave, 1 = saw-wave
//------------------------------------------------------------
declare CZsawP author "Bart Brouns";
declare CZsawP licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZsawP(fund, index) = CZ.sawChooseP(fund, index, 1);

//----------`(os.)CZsquare`----------
// Oscillator that mimics the Casio CZ square oscillator
// `CZsquare` is a standard Faust function.
//
// #### Usage
//
// ```
// CZsquare(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 to 1. 0 = sine-wave, 1 = square-wave
//------------------------------------------------------------
declare CZsquare author "Bart Brouns";
declare CZsquare licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZsquare(fund, index) = CZ.squareChooseP(fund, index, 0);

//----------`(os.)CZsquareP`----------
// Oscillator that mimics the Casio CZ square oscillator,
// with it's phase aligned to `fund:sin`.
// `CZsquareP` is a standard Faust function.
//
// #### Usage
//
// ```
// CZsquareP(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 to 1. 0 = sine-wave, 1 = square-wave
//------------------------------------------------------------
declare CZsquareP author "Bart Brouns";
declare CZsquareP licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns
```

```

CZsquareP(fund, index) = CZ.squareChooseP(fund, index, 1);

//----------`(os.)CZpulse`----------
// Oscillator that mimics the Casio CZ pulse oscillator.
// `CZpulse` is a standard Faust function.
//
// #### Usage
//
// ```
// CZpulse(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 gives a sine-wave, 1 is closer to a pulse
//------------------------------------------------------------
declare CZpulse author "Bart Brouns";
declare CZpulse licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZpulse(fund, index) = CZ.pulseChooseP(fund, index, 0);

//----------`(os.)CZpulseP`----------
// Oscillator that mimics the Casio CZ pulse oscillator,
// with it's phase aligned to `fund:sin`.
// `CZpulseP` is a standard Faust function.
//
// #### Usage
//
// ```
// CZpulseP(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 gives a sine-wave, 1 is closer to a pulse
//------------------------------------------------------------
declare CZpulseP author "Bart Brouns";
declare CZpulseP licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZpulseP(fund, index) = CZ.pulseChooseP(fund, index, 1);

//----------`(os.)CZsinePulse`----------
// Oscillator that mimics the Casio CZ sine/pulse oscillator.
// `CZsinePulse` is a standard Faust function.
//
// #### Usage
//
// ```
// CZsinePulse(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 gives a sine-wave, 1 is a sine minus a
     pulse
//------------------------------------------------------------
```

```
declare CZsinePulse author "Bart Brouns";
declare CZsinePulse licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZsinePulse(fund, index) = CZ.sinePulseChooseP(fund, index, 0);

//----------`(os.)CZsinePulseP`----------
// Oscillator that mimics the Casio CZ sine/pulse oscillator,
// with it's phase aligned to `fund:sin`.
// `CZsinePulseP` is a standard Faust function.
//
// #### Usage
//
// ```
// CZsinePulseP(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 gives a sine-wave, 1 is a sine minus a
     pulse
//------------------------------------------------------------
declare CZsinePulseP author "Bart Brouns";
declare CZsinePulseP licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZsinePulseP(fund, index) = CZ.sinePulseChooseP(fund, index, 1);

//----------`(os.)CZhalfSine`----------
// Oscillator that mimics the Casio CZ half sine oscillator.
// `CZhalfSine` is a standard Faust function.
//
// #### Usage
//
// ```
// CZhalfSine(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 gives a sine-wave, 1 is somewhere between a
     saw and a square
//------------------------------------------------------------
declare CZhalfSine author "Bart Brouns";
declare CZhalfSine licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZhalfSine(fund, index) = CZ.halfSineChooseP(fund, index, 0);

//----------`(os.)CZhalfSineP`----------
// Oscillator that mimics the Casio CZ half sine oscillator,
// with it's phase aligned to `fund:sin`.
// `CZhalfSineP` is a standard Faust function.
//
// #### Usage
//
// ```
```

```
// CZhalfSineP(fund,index) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `index`: the brightness of the oscillator, 0 gives a sine-wave, 1 is somewhere between a
      saw and a square
//------------------------------------------------------------
declare CZhalfSineP author "Bart Brouns";
declare CZhalfSineP licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZhalfSineP(fund, index) = CZ.halfSineChooseP(fund, index, 1);

//----------`(os.)CZresSaw`----------
// Oscillator that mimics the Casio CZ resonant sawtooth oscillator.
// `CZresSaw` is a standard Faust function.
//
// #### Usage
//
// ```
// CZresSaw(fund,res) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `res`: the frequency of resonance as a factor of the fundamental pitch.
//------------------------------------------------------------
declare CZresSaw author "Bart Brouns";
declare CZresSaw licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZresSaw(fund,res) = CZ.resSaw(fund,res);

//----------`(os.)CZresTriangle`----------
// Oscillator that mimics the Casio CZ resonant triangle oscillator.
// `CZresTriangle` is a standard Faust function.
//
// #### Usage
//
// ```
// CZresTriangle(fund,res) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `res`: the frequency of resonance as a factor of the fundamental pitch.
//------------------------------------------------------------
declare CZresTriangle author "Bart Brouns";
declare CZresTriangle licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZresTriangle(fund,res) = CZ.resTriangle(fund,res);

//----------`(os.)CZresTrap`----------
// Oscillator that mimics the Casio CZ resonant trapeze oscillator
```

```
// `CZresTrap` is a standard Faust function.
//
// #### Usage
//
// ```
// CZresTrap(fund,res) : _
// ```
//
// Where:
//
// * `fund`: a saw-tooth waveform between 0 and 1 that the oscillator slaves to
// * `res`: the frequency of resonance as a factor of the fundamental pitch.
//------------------------------------------------------------
declare CZresTrap author "Bart Brouns";
declare CZresTrap licence "STK-4.3";

// CZ oscillators by Mike Moser-Booth:
// <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
// Ported from pd to Faust by Bart Brouns

CZresTrap(fund, res) = CZ.resTrap(fund, res);

CZ = environment {

    saw(fund, index) = sawChooseP(fund, index, 0);
    sawP(fund, index) = sawChooseP(fund, index, 1);
    sawChooseP(fund, index, p) =
      (((FUND(fund,align,p)*((.5-INDEX)/INDEX)),(-1*FUND(fund,align,p)+1)*((.5-INDEX)/(1-INDEX))):min+FUND(fund,align,p))*2*ma.PI:cos
    with {
      INDEX = (.5-(index*.5)):max(0.01):min(0.5);
      align = si.interpolate(index, 0.75, 0.5);
    };

    square(fund, index) = squareChooseP(fund, index, 0);
    squareP(fund, index) = squareChooseP(fund, index, 1);
    squareChooseP(fund, index, p) = (FUND(fund,align,p)>=0.5), (ma.decimal((FUND(fund,align,p)*2)+1)<:_-min(_,(-1*_+1)*((INDEX)/(1-INDEX)))) :+ *ma.PI:cos
    with {
      INDEX = (index:pow(0.25)):max(0):min(1);
      align = si.interpolate(INDEX, -0.25, 0);
    };

    pulse(fund, index) = pulseChooseP(fund, index, 0);
    pulseP(fund, index) = pulseChooseP(fund, index, 1);
    pulseChooseP(fund, index, p) = ((FUND(fund,align,p)-min(FUND(fund,align,p),((-1*FUND(fund,align,p)+1)*(INDEX/(1-INDEX)))))*2*ma.PI):cos
    with {
      INDEX = index:min(0.99):max(0);
      align = si.interpolate(index, -0.25, 0.0);
    };

    sinePulse(fund, index) = sinePulseChooseP(fund, index, 0);
    sinePulseP(fund, index) = sinePulseChooseP(fund, index, 1);
    sinePulseChooseP(fund, index, p) = (min(FUND(fund,align,p)*((0.5-INDEX)/INDEX),(-1*FUND(fund,align,p)+1)*((.5-INDEX)/(1-INDEX)))+FUND(fund,align,p))*4*ma.PI:cos
    with {
      INDEX = ((index*-0.49)+0.5);
      align = si.interpolate(index, -0.125, -0.25);
    };

    halfSine(fund, index) = halfSineChooseP(fund, index, 0);
    halfSineP(fund, index) = halfSineChooseP(fund, index, 1);
    halfSineChooseP(fund, index, p) = (select2(FUND(fund,align,p)<.5, .5*(FUND(fund,align,p)-.5)/INDEX+.5, FUND(fund,align,p)):min(1))*2*ma.PI:cos
    with {
      INDEX = (.5-(index*0.5)):min(.5):max(.01);
      align = si.interpolate(index:min(0.975), -0.25, -0.5);
```

```
    };

    FUND =
      case {
        (fund,align,0) => fund;
        (fund,align,1) => (fund+align) : ma.frac; // align phase with fund
      };
    resSaw(fund,res) = (((-1*(1-fund))*((cos((ma.decimal((max(1,res)*fund)+1))*2*ma.PI)*-.5)
        +.5))*2)+1;
    resTriangle(fund,res) = select2(fund<.5, 2-(fund*2), fund*2)*INDEX*2-1
    with {
      INDEX = ((fund*(res:max(1)))+1:ma.decimal)*2*ma.PI:cos*.5+.5;
    };
    resTrap(fund, res) = (((1-fund)*2):min(1)*sin(ma.decimal(fund*(res:max(1)))*2*ma.PI));
};

//==============================PolyBLEP-Based Oscillators=================================

//----------`(os.)polyblep`----------
// PolyBLEP residual function, used for smoothing steps in the audio signal.
//
// #### Usage
//
// ```
// polyblep(Q,phase) : _
// ```
//
// Where:
//
// * `Q`: smoothing factor between 0 and 0.5. Determines how far from the ends of the phase
     interval the quadratic function is used.
// * `phase`: normalised phase (between 0 and 1)
//---------------------------------------------------------------
declare polyblep author "Jacek Wieczorek";

polyblep(Q, phase) = (0, L(phase / Q), R((phase - 1) / Q)) : select3(sel)
with {
    sel = (phase < Q) + 2*(phase > 1 - Q);
    L(x) = 2*x - x*x - 1; // Used near the left end of the interval
    R(x) = 2*x + x*x + 1; // Used near the right end of the interval
};

//----------`(os.)polyblep_saw`----------
// Sawtooth oscillator with suppressed aliasing (using `polyblep`).
//
// #### Usage
//
// ```
// polyblep_saw(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//---------------------------------------------------------------
declare polyblep_saw author "Jacek Wieczorek";

polyblep_saw(freq) = naive - polyblep(Q , phase)
with {
    phase = phasor(1, freq);
    naive = 2 * phase - 1;
    Q = freq / ma.SR;
};

//----------`(os.)polyblep_square`----------
// Square wave oscillator with suppressed aliasing (using `polyblep`).
//
// #### Usage
```

```
//
// ```
// polyblep_square(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//-----------------------------------------------------------
declare polyblep_square author "Jacek Wieczorek";

polyblep_square(freq) = naive - polyblep(Q, phase) + polyblep(Q, ma.modulo(phase + 0.5, 1))
with {
    phase = phasor(1, freq);
    naive = 2 * (phase * 2 : int) - 1;
    Q = freq / ma.SR;
};

//----------`(os.)polyblep_triangle`----------
// Triangle wave oscillator with suppressed aliasing (using `polyblep`).
//
// #### Usage
//
// ```
// polyblep_triangle(freq) : _
// ```
//
// Where:
//
// * `freq`: frequency in Hz
//-----------------------------------------------------------
declare polyblep_triangle author "Jacek Wieczorek";

polyblep_triangle(freq) = polyblep_square(freq) : fi.pole(0.999) : *(4 * freq / ma.SR);

// end further contributions section
//###########################################################################################
```

Listing 4: `maths.lib`

```
//################################### maths.lib ##########################################
//  Mathematic library for Faust. Its official prefix is `ma`.
//
// #### References
// * <https://github.com/grame-cncm/faustlibraries/blob/master/maths.lib>
//###########################################################################################
// Some functions are implemented as Faust foreign functions of `math.h` functions
// that are not part of Faust's primitives. Defines also various constants and several
// utilities.
//###########################################################################################

// ## History
// * 06/13/2016 [RM]  normalizing and integrating to new libraries
// * 07/08/2015 [YO]  documentation comments
// * 20/06/2014 [SL]  added FTZ function
// * 22/06/2013 [YO]  added float|double|quad variants of some foreign functions
// * 28/06/2005 [YO]  postfixed functions with 'f' to force float version instead of double
// * 28/06/2005 [YO]  removed 'modf' because it requires a pointer as argument

/************************************************************************
************************************************************************
FAUST library file
Copyright (C) 2003-2016 GRAME, Centre National de Creation Musicale
---------------------------------------------------------------------
```

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation; either version 2.1 of the
License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with the GNU C Library; if not, write to the Free
Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
02111-1307 USA.

EXCEPTION TO THE LGPL LICENSE : As a special exception, you may create a
larger FAUST program which directly or indirectly imports this library
file and still distribute the compiled code generated by the FAUST
compiler, or a modified version of this compiled code, under your own
copyright and license. This EXCEPTION TO THE LGPL LICENSE explicitly
grants you the right to freely choose the license for the resulting
compiled code. In particular the resulting compiled code has no obligation
to be LGPL or GPL. For example you are free to choose a commercial or
closed source license or any other license if you decide so.
************************************************************************
************************************************************************/

// This library contains platform specific constants
pl = library("platform.lib");
ma = library("maths.lib"); // for compatible copy/paste out of this file

declare name "Faust Math Library";
declare version "2.8.0";
declare author "GRAME";
declare copyright "GRAME";
declare license "LGPL with exception";

//=============================Functions Reference========================================
//========================================================================================


//---------------------------------`(ma.)SR`---------------------------------------
// Current sampling rate given at init time. Constant during program execution.
//
// #### Usage
//
// ```
// SR : _
// ```
//-----------------------------------------------------------------------------
SR = pl.SR;

//---------------------------------`(ma.)T`---------------------------------------
// Current sample duration in seconds computed from the sampling rate given at init time.
     Constant during program execution.
//
// #### Usage
//
// ```
// T : _
// ```
//-----------------------------------------------------------------------------
T = 1.0 / SR;

//---------------------------------`(ma.)BS`---------------------------------------
// Current block-size. Can change during the execution at each block.
//
// #### Usage
```

````
//
// ```
// BS : _
// ```
//----------------------------------------------------------------------------
BS = pl.BS;


//---------------------------------`(ma.)PI`---------------------------------------
// Constant PI in double precision.
//
// #### Usage
//
// ```
// PI : _
// ```
//----------------------------------------------------------------------------
PI = 3.14159265358979323846;


//---------------------------------`(ma.)deg2rad`----------------------------------
// Convert degrees to radians.
//
// #### Usage
//
// ```
// 45. : deg2rad
// ```
//----------------------------------------------------------------------------
deg2rad = _ * PI / 180.;


//---------------------------------`(ma.)rad2deg`----------------------------------
// Convert radians to degrees.
//
// #### Usage
//
// ```
// ma.PI : rad2deg
// ```
//----------------------------------------------------------------------------
rad2deg = _ * 180. / PI;


//---------------------------------`(ma.)E`---------------------------------------
// Constant e in double precision.
//
// #### Usage
//
// ```
// E : _
// ```
//----------------------------------------------------------------------------
E = 2.71828182845904523536;


//---------------------------------`(ma.)EPSILON`---------------------------------------
// Constant EPSILON available in simple/double/quad precision,
// as defined in the [floating-point standard](https://en.wikipedia.org/wiki/IEEE_754)
// and [machine epsilon](https://en.wikipedia.org/wiki/Machine_epsilon),
// that is smallest positive number such that `1.0 + EPSILON != 1.0`.
//
// #### Usage
//
// ```
// EPSILON : _
// ```
//----------------------------------------------------------------------------
````

```
singleprecision EPSILON = 1.192092896e-07;
doubleprecision EPSILON = 2.2204460492503131e-016;
quadprecision EPSILON = 1.084202172485504434007452e-019;
fixedpointprecision EPSILON = 2.2204460492503131e-016;


//--------------------------------`(ma.)MIN`--------------------------------------
// Constant MIN available in simple/double/quad precision (minimal positive value).
//
// #### Usage
//
// ```
// MIN : _
// ```
//-----------------------------------------------------------------------------
singleprecision MIN = 1.175494351e-38;
doubleprecision MIN = 2.2250738585072014e-308;
quadprecision MIN = 2.2250738585072014e-308;
fixedpointprecision MIN = 2.2250738585072014e-308;


//--------------------------------`(ma.)MAX`------------------------------
// Constant MAX available in simple/double/quad precision (maximal positive value).
//
// #### Usage
//
// ```
// MAX : _
// ```
//-----------------------------------------------------------------------------
singleprecision MAX = 3.402823466e+38;
doubleprecision MAX = 1.7976931348623158e+308;
quadprecision MAX = 1.7976931348623158e+308;
fixedpointprecision MAX = 1.7976931348623158e+308;

// Obsolete, kept for compatibility reasons
INFINITY = MAX;

//--------------------------------`(ma.)FTZ`--------------------------------------
// Flush to zero: force samples under the "maximum subnormal number"
// to be zero. Usually not needed in C++ because the architecture
// file take care of this, but can be useful in JavaScript for instance.
//
// #### Usage
//
// ```
// _ : FTZ : _
// ```
//
// #### Reference
//
// <http://docs.oracle.com/cd/E19957-01/806-3568/ncg_math.html>
//-----------------------------------------------------------------------------
FTZ(x) = x * (abs(x) > MIN);


//--------------------------------`(ma.)copysign`--------------------------------------
// Changes the sign of x (first input) to that of y (second input).
//
// #### Usage
//
// ```
// _,_ : copysign : _
// ```
//-----------------------------------------------------------------------------
copysign = ffunction(float copysignf|copysign|copysignl (float, float), <math.h>,"");
```

```
//--------------------------------`(ma.)neg`--------------------------------------
// Invert the sign (-x) of a signal.
//
// #### Usage
//
// ```
// _ : neg : _
// ```
//-----------------------------------------------------------------------------
neg(x) = -x;


//--------------------------------`(ma.)not`---------------------------------------
// Bitwise `not` implemented with [xor](https://faustdoc.grame.fr/manual/syntax/#xor-primitive) as `not(x) = x xor -1;`.
// So working regardless of the size of the integer, assuming negative numbers in two's complement.
//
// #### Usage
//
// ```
// _ : not : _
// ```
//-----------------------------------------------------------------------------
not(x) = x xor -1;


//-------`(ma.)sub(x,y)`------------------
// Subtract `x` and `y`.
//
// #### Usage
//
// ```
// _,_ : sub : _
// ```
//-----------------------------
sub(x,y) = y-x;


//--------------------------------`(ma.)inv`--------------------------------------
// Compute the inverse (1/x) of the input signal.
//
// #### Usage
//
// ```
// _ : inv : _
// ```
//-----------------------------------------------------------------------------
inv(x) = 1/x;


//--------------------------------`(ma.)cbrt`-------------------------------------
// Computes the cube root of of the input signal.
//
// #### Usage
//
// ```
// _ : cbrt : _
// ```
//-----------------------------------------------------------------------------
cbrt = ffunction(float cbrtf|cbrt|cbrtl (float), <math.h>,"");


//--------------------------------`(ma.)hypot`------------------------------------
// Computes the euclidian distance of the two input signals
// sqrt(x*x+y*y) without undue overflow or underflow.
//
// #### Usage
```

```
//
// ```
// _,_ : hypot : _
// ```
//-----------------------------------------------------------------------------
hypot = ffunction(float hypotf|hypot|hypotl (float, float), <math.h>,"");


//--------------------------------`(ma.)ldexp`-------------------------------------
// Takes two input signals: x and n, and multiplies x by 2 to the power n.
//
// #### Usage
//
// ```
// _,_ : ldexp : _
// ```
//-----------------------------------------------------------------------------
ldexp = ffunction(float ldexpf|ldexp|ldexpl (float, int), <math.h>,"");


//--------------------------------`(ma.)scalb`-------------------------------------
// Takes two input signals: x and n, and multiplies x by 2 to the power n.
//
// #### Usage
//
// ```
// _,_ : scalb : _
// ```
//-----------------------------------------------------------------------------
scalb = ffunction(float scalbnf|scalbn|scalbnl (float, int), <math.h>,"");


//--------------------------------`(ma.)log1p`---------------------------------
// Computes log(1 + x) without undue loss of accuracy when x is nearly zero.
//
// #### Usage
//
// ```
// _ : log1p : _
// ```
//-----------------------------------------------------------------------------
log1p = ffunction(float log1pf|log1p|log1pl (float), <math.h>,"");


//--------------------------------`(ma.)logb`--------------------------------------
// Return exponent of the input signal as a floating-point number.
//
// #### Usage
//
// ```
// _ : logb : _
// ```
//-----------------------------------------------------------------------------
logb = ffunction(float logbf|logb|logbl (float), <math.h>,"");


//--------------------------------`(ma.)ilogb`-------------------------------------
// Return exponent of the input signal as an integer number.
//
// #### Usage
//
// ```
// _ : ilogb : _
// ```
//-----------------------------------------------------------------------------
ilogb = ffunction(int ilogbf|ilogb|ilogbl (float), <math.h>,"");
```

```
//---------------------------------‘(ma.)log2‘-------------------------------------
// Returns the base 2 logarithm of x.
//
// #### Usage
//
// ‘‘‘
// _ : log2 : _
// ‘‘‘
//-----------------------------------------------------------------------------
log2(x) = log(x)/log(2.0);


//---------------------------------‘(ma.)expm1‘-------------------------------------
// Return exponent of the input signal minus 1 with better precision.
//
// #### Usage
//
// ‘‘‘
// _ : expm1 : _
// ‘‘‘
//-----------------------------------------------------------------------------
expm1 = ffunction(float expm1f|expm1|expm1l (float), <math.h>,"");


//---------------------------------‘(ma.)acosh‘-------------------------------------
// Computes the principle value of the inverse hyperbolic cosine
// of the input signal.
//
// #### Usage
//
// ‘‘‘
// _ : acosh : _
// ‘‘‘
//-----------------------------------------------------------------------------
acosh = ffunction(float acoshf|acosh|acoshl (float), <math.h>, "");


//--------------------------------‘(ma.)asinh‘-----------------------------------
// Computes the inverse hyperbolic sine of the input signal.
//
// #### Usage
//
// ‘‘‘
// _ : asinh : _
// ‘‘‘
//-----------------------------------------------------------------------------
asinh = ffunction(float asinhf|asinh|asinhl (float), <math.h>, "");


//--------------------------------‘(ma.)atanh‘-----------------------------------
// Computes the inverse hyperbolic tangent of the input signal.
//
// #### Usage
//
// ‘‘‘
// _ : atanh : _
// ‘‘‘
//-----------------------------------------------------------------------------
atanh = ffunction(float atanhf|atanh|atanhl (float), <math.h>, "");


//---------------------------------‘(ma.)sinh‘---------------------------------------
// Computes the hyperbolic sine of the input signal.
//
// #### Usage
//
// ‘‘‘
// _ : sinh : _
```

````
// ```
//-----------------------------------------------------------------------------
sinh = ffunction(float sinhf|sinh|sinhl (float), <math.h>, "");


//---------------------------------`(ma.)cosh`--------------------------------------
// Computes the hyperbolic cosine of the input signal.
//
// #### Usage
//
// ```
// _ : cosh : _
// ```
//-----------------------------------------------------------------------------
cosh = ffunction(float coshf|cosh|coshl (float), <math.h>, "");


//---------------------------------`(ma.)tanh`--------------------------------------
// Computes the hyperbolic tangent of the input signal.
//
// #### Usage
//
// ```
// _ : tanh : _
// ```
//-----------------------------------------------------------------------------
tanh = ffunction(float tanhf|tanh|tanhl (float), <math.h>,"");


//---------------------------------`(ma.)erf`---------------------------------------
// Computes the error function of the input signal.
//
// #### Usage
//
// ```
// _ : erf : _
// ```
//-----------------------------------------------------------------------------
erf = ffunction(float erff|erf|erfl(float), <math.h>,"");


//---------------------------------`(ma.)erfc`---------------------------------------
// Computes the complementary error function of the input signal.
//
// #### Usage
//
// ```
// _ : erfc : _
// ```
//-----------------------------------------------------------------------------
erfc = ffunction(float erfcf|erfc|erfcl(float), <math.h>,"");


//---------------------------------`(ma.)gamma`-------------------------------------
// Computes the gamma function of the input signal.
//
// #### Usage
//
// ```
// _ : gamma : _
// ```
//-----------------------------------------------------------------------------
gamma = ffunction(float tgammaf|tgamma|tgammal(float), <math.h>,"");


//---------------------------------`(ma.)lgamma`------------------------------------
// Calculates the natural logorithm of the absolute value of
// the gamma function of the input signal.
````

```
//
// #### Usage
//
// ```
// _ : lgamma : _
// ```
//------------------------------------------------------------------------------
lgamma = ffunction(float lgammaf|lgamma|lgammal(float), <math.h>,"");


//----------------------------------`(ma.)J0`---------------------------------------
// Computes the Bessel function of the first kind of order 0
// of the input signal.
//
// #### Usage
//
// ```
// _ : J0 : _
// ```
//------------------------------------------------------------------------------
J0 = ffunction(float j0(float), <math.h>,"");


//----------------------------------`(ma.)J1`---------------------------------------
// Computes the Bessel function of the first kind of order 1
// of the input signal.
//
// #### Usage
//
// ```
// _ : J1 : _
// ```
//------------------------------------------------------------------------------
J1 = ffunction(float j1(float), <math.h>,"");


//----------------------------------`(ma.)Jn`---------------------------------------
// Computes the Bessel function of the first kind of order n
// (first input signal) of the second input signal.
//
// #### Usage
//
// ```
// _,_ : Jn : _
// ```
//------------------------------------------------------------------------------
Jn = ffunction(float jn(int, float), <math.h>,"");


//----------------------------------`(ma.)Y0`---------------------------------------
// Computes the linearly independent Bessel function of the second kind
// of order 0 of the input signal.
//
// #### Usage
//
// ```
// _ : Y0 : _
// ```
//------------------------------------------------------------------------------
Y0 = ffunction(float y0(float), <math.h>,"");


//----------------------------------`(ma.)Y1`---------------------------------------
// Computes the linearly independent Bessel function of the second kind
// of order 1 of the input signal.
//
// #### Usage
//
```

```
// ```
// _ : Y0 : _
// ```
//-----------------------------------------------------------------------------
Y1 = ffunction(float y1(float), <math.h>,"");


//----------------------------------`(ma.)Yn`---------------------------------------
// Computes the linearly independent Bessel function of the second kind
// of order n (first input signal) of the second input signal.
//
// #### Usage
//
// ```
// _,_ : Yn : _
// ```
//-----------------------------------------------------------------------------
Yn = ffunction(float yn(int, float), <math.h>,"");


//----------------------------`(ma.)fabs`, `(ma.)fmax`, `(ma.)fmin
      `---------------------------
// Just for compatibility...
//
// ```
// fabs = abs
// fmax = max
// fmin = min
// ```
//-----------------------------------------------------------------------------
fabs = abs;
fmax = max;
fmin = min;

//-------------------------------`(ma.)np2`--------------------------------------
// Gives the next power of 2 of x.
//
// #### Usage
//
// ```
// np2(n) : _
// ```
//
// Where:
//
// * `n`: an integer
//-----------------------------------------------------------------------------
np2 = -(1) <: >>(1)|_ <: >>(2)|_ <: >>(4)|_ <: >>(8)|_ <: >>(16)|_ : +(1);


//-----------------------------`(ma.)frac`---------------------------------------
// Gives the fractional part of n.
//
// #### Usage
//
// ```
// frac(n) : _
// ```
//
// Where:
//
// * `n`: a decimal number
//------------------------------------------------------------------------------
frac(n) = n - floor(n);
decimal = frac;

// NOTE: decimal does the same thing as frac but using floor instead. JOS uses frac a lot
// in filters.lib so we decided to keep that one... decimal is declared though for
```

```
// backward compatibility.
// decimal(n) = n - floor(n);


//-----------------------------`(ma.)modulo`--------------------------------------
// Modulus operation using the `(x%y+y)%y` formula to ensures the result is always non-negative, even if `x` is negative.
//
// #### Usage
//
// ```
// modulo(x,y) : _
// ```
//
// Where:
//
// * `x`: the numerator
// * `y`: the denominator
//-----------------------------------------------------------------------------
modulo(x,y) = (x % y + y) % y;


//---------------`(ma.)isnan`----------------
// Return non-zero if x is a NaN.
//
// #### Usage
//
// ```
// isnan(x)
// _ : isnan : _
// ```
//
// Where:
//
// * `x`: signal to analyse
//------------------------------------------
isnan = ffunction(int isnanf|isnan|isnanl (float),<math.h>,"");


//---------------`(ma.)isinf`----------------
// Return non-zero if x is a positive or negative infinity.
//
// #### Usage
//
// ```
// isinf(x)
// _ : isinf : _
// ```
//
// Where:
//
// * `x`: signal to analyse
//------------------------------------------
isinf = ffunction(int isinff|isinf|isinfl (float),<math.h>,"");

nextafter = ffunction(float nextafter(float, float),<math.h>,"");


//--------------------------`(ma.)chebychev`------------------------------
// Chebychev transformation of order N.
//
// #### Usage
//
// ```
// _ : chebychev(N) : _
// ```
//
// Where:
```

```
//
// * `N`: the order of the polynomial, a constant numerical expression
//
// #### Semantics
//
// ```
// T[0](x) = 1,
// T[1](x) = x,
// T[n](x) = 2x*T[n-1](x) - T[n-2](x)
// ```
//
// #### Reference
//
// <http://en.wikipedia.org/wiki/Chebyshev_polynomial>
//------------------------------------------------------------------------
chebychev(0,x) = 1;
chebychev(1,x) = x;
chebychev(n,x) = 2*x*chebychev(n-1, x) - chebychev(n-2, x);


//-----------------------`(ma.)chebychevpoly`------------------------------
// Linear combination of the first Chebyshev polynomials.
//
// #### Usage
//
// ```
// _ : chebychevpoly((c0,c1,...,cn)) : _
// ```
//
// Where:
//
// * `cn`: the different Chebychevs polynomials such that:
//  chebychevpoly((c0,c1,...,cn)) = Sum of chebychev(i)*ci
//
// #### Reference
//
// <http://www.csounds.com/manual/html/chebyshevpoly.html>
//------------------------------------------------------------------------
chebychevpoly(lcoef) = _ <: L(0,lcoef) :> _
    with {
        L(n,(c,cs)) = chebychev(n)*c, L(n+1,cs);
        L(n,c)     = chebychev(n)*c;
    };


//-----------------`(ma.)diffn`----------------------------
// Negated first-order difference.
//
// #### Usage
//
// ```
// _ : diffn : _
// ```
//--------------------------------------------------------
diffn(x) = x' - x; // negated first-order difference


//-----------------`(ma.)signum`----------------------------
// The signum function signum(x) is defined as
// -1 for x<0, 0 for x==0, and 1 for x>0.
//
// #### Usage
//
// ```
// _ : signum : _
// ```
//--------------------------------------------------------
signum(x) = (x>0)-(x<0);
```

```
//-----------------‘(ma.)nextpow2‘---------------------------
// The nextpow2(x) returns the lowest integer m such that
// 2^m >= x.
//
// #### Usage
//
// ‘‘‘
// 2^nextpow2(n) : _
// ‘‘‘
// Useful for allocating delay lines, e.g.,
// ‘‘‘
// delay(2^nextpow2(maxDelayNeeded), currentDelay);
// ‘‘‘
//-----------------------------------------------------
nextpow2(x) = ceil(log(x)/log(2.0));


//-------------------‘(ma.)zc‘---------------------------------------------
// Indicator function for zero-crossing: it returns 1 if a zero-crossing
// occurs, 0 otherwise.
//
// #### Usage
//
// ‘‘‘
// _ : zc : _
// ‘‘‘
//-----------------------------------------------------------------------
zc(x) = x * x’ < 0;


//-------------------‘(ma.)primes‘---------------------------------------------
// Return the n-th prime using a waveform primitive. Note that primes(0) is 2,
// primes(1) is 3, and so on. The waveform is length 2048, so the largest
// precomputed prime is primes(2047) which is 17863.
//
// #### Usage
//
// ‘‘‘
// _ : primes : _
// ‘‘‘
//-----------------------------------------------------------------------
primes(x) = rdtable(waveform{
2,3,5,7,11,13,17,19,23,29,
31,37,41,43,47,53,59,61,67,71,
73,79,83,89,97,101,103,107,109,113,
127,131,137,139,149,151,157,163,167,173,
179,181,191,193,197,199,211,223,227,229,
233,239,241,251,257,263,269,271,277,281,
283,293,307,311,313,317,331,337,347,349,
353,359,367,373,379,383,389,397,401,409,
419,421,431,433,439,443,449,457,461,463,
467,479,487,491,499,503,509,521,523,541,
547,557,563,569,571,577,587,593,599,601,
607,613,617,619,631,641,643,647,653,659,
661,673,677,683,691,701,709,719,727,733,
739,743,751,757,761,769,773,787,797,809,
811,821,823,827,829,839,853,857,859,863,
877,881,883,887,907,911,919,929,937,941,
947,953,967,971,977,983,991,997,1009,1013,
1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,
1087,1091,1093,1097,1103,1109,1117,1123,1129,1151,
1153,1163,1171,1181,1187,1193,1201,1213,1217,1223,
1229,1231,1237,1249,1259,1277,1279,1283,1289,1291,
1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,
1381,1399,1409,1423,1427,1429,1433,1439,1447,1451,
1453,1459,1471,1481,1483,1487,1489,1493,1499,1511,
```

```
1523,1531,1543,1549,1553,1559,1567,1571,1579,1583,
1597,1601,1607,1609,1613,1619,1621,1627,1637,1657,
1663,1667,1669,1693,1697,1699,1709,1721,1723,1733,
1741,1747,1753,1759,1777,1783,1787,1789,1801,1811,
1823,1831,1847,1861,1867,1871,1873,1877,1879,1889,
1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,
1993,1997,1999,2003,2011,2017,2027,2029,2039,2053,
2063,2069,2081,2083,2087,2089,2099,2111,2113,2129,
2131,2137,2141,2143,2153,2161,2179,2203,2207,2213,
2221,2237,2239,2243,2251,2267,2269,2273,2281,2287,
2293,2297,2309,2311,2333,2339,2341,2347,2351,2357,
2371,2377,2381,2383,2389,2393,2399,2411,2417,2423,
2437,2441,2447,2459,2467,2473,2477,2503,2521,2531,
2539,2543,2549,2551,2557,2579,2591,2593,2609,2617,
2621,2633,2647,2657,2659,2663,2671,2677,2683,2687,
2689,2693,2699,2707,2711,2713,2719,2729,2731,2741,
2749,2753,2767,2777,2789,2791,2797,2801,2803,2819,
2833,2837,2843,2851,2857,2861,2879,2887,2897,2903,
2909,2917,2927,2939,2953,2957,2963,2969,2971,2999,
3001,3011,3019,3023,3037,3041,3049,3061,3067,3079,
3083,3089,3109,3119,3121,3137,3163,3167,3169,3181,
3187,3191,3203,3209,3217,3221,3229,3251,3253,3257,
3259,3271,3299,3301,3307,3313,3319,3323,3329,3331,
3343,3347,3359,3361,3371,3373,3389,3391,3407,3413,
3433,3449,3457,3461,3463,3467,3469,3491,3499,3511,
3517,3527,3529,3533,3539,3541,3547,3557,3559,3571,
3581,3583,3593,3607,3613,3617,3623,3631,3637,3643,
3659,3671,3673,3677,3691,3697,3701,3709,3719,3727,
3733,3739,3761,3767,3769,3779,3793,3797,3803,3821,
3823,3833,3847,3851,3853,3863,3877,3881,3889,3907,
3911,3917,3919,3923,3929,3931,3943,3947,3967,3989,
4001,4003,4007,4013,4019,4021,4027,4049,4051,4057,
4073,4079,4091,4093,4099,4111,4127,4129,4133,4139,
4153,4157,4159,4177,4201,4211,4217,4219,4229,4231,
4241,4243,4253,4259,4261,4271,4273,4283,4289,4297,
4327,4337,4339,4349,4357,4363,4373,4391,4397,4409,
4421,4423,4441,4447,4451,4457,4463,4481,4483,4493,
4507,4513,4517,4519,4523,4547,4549,4561,4567,4583,
4591,4597,4603,4621,4637,4639,4643,4649,4651,4657,
4663,4673,4679,4691,4703,4721,4723,4729,4733,4751,
4759,4783,4787,4789,4793,4799,4801,4813,4817,4831,
4861,4871,4877,4889,4903,4909,4919,4931,4933,4937,
4943,4951,4957,4967,4969,4973,4987,4993,4999,5003,
5009,5011,5021,5023,5039,5051,5059,5077,5081,5087,
5099,5101,5107,5113,5119,5147,5153,5167,5171,5179,
5189,5197,5209,5227,5231,5233,5237,5261,5273,5279,
5281,5297,5303,5309,5323,5333,5347,5351,5381,5387,
5393,5399,5407,5413,5417,5419,5431,5437,5441,5443,
5449,5471,5477,5479,5483,5501,5503,5507,5519,5521,
5527,5531,5557,5563,5569,5573,5581,5591,5623,5639,
5641,5647,5651,5653,5657,5659,5669,5683,5689,5693,
5701,5711,5717,5737,5741,5743,5749,5779,5783,5791,
5801,5807,5813,5821,5827,5839,5843,5849,5851,5857,
5861,5867,5869,5879,5881,5897,5903,5923,5927,5939,
5953,5981,5987,6007,6011,6029,6037,6043,6047,6053,
6067,6073,6079,6089,6091,6101,6113,6121,6131,6133,
6143,6151,6163,6173,6197,6199,6203,6211,6217,6221,
6229,6247,6257,6263,6269,6271,6277,6287,6299,6301,
6311,6317,6323,6329,6337,6343,6353,6359,6361,6367,
6373,6379,6389,6397,6421,6427,6449,6451,6469,6473,
6481,6491,6521,6529,6547,6551,6553,6563,6569,6571,
6577,6581,6599,6607,6619,6637,6653,6659,6661,6673,
6679,6689,6691,6701,6703,6709,6719,6733,6737,6761,
6763,6779,6781,6791,6793,6803,6823,6827,6829,6833,
6841,6857,6863,6869,6871,6883,6899,6907,6911,6917,
6947,6949,6959,6961,6967,6971,6977,6983,6991,6997,
7001,7013,7019,7027,7039,7043,7057,7069,7079,7103,
7109,7121,7127,7129,7151,7159,7177,7187,7193,7207,
```

```
7211,7213,7219,7229,7237,7243,7247,7253,7283,7297,
7307,7309,7321,7331,7333,7349,7351,7369,7393,7411,
7417,7433,7451,7457,7459,7477,7481,7487,7489,7499,
7507,7517,7523,7529,7537,7541,7547,7549,7559,7561,
7573,7577,7583,7589,7591,7603,7607,7621,7639,7643,
7649,7669,7673,7681,7687,7691,7699,7703,7717,7723,
7727,7741,7753,7757,7759,7789,7793,7817,7823,7829,
7841,7853,7867,7873,7877,7879,7883,7901,7907,7919,
7927,7933,7937,7949,7951,7963,7993,8009,8011,8017,
8039,8053,8059,8069,8081,8087,8089,8093,8101,8111,
8117,8123,8147,8161,8167,8171,8179,8191,8209,8219,
8221,8231,8233,8237,8243,8263,8269,8273,8287,8291,
8293,8297,8311,8317,8329,8353,8363,8369,8377,8387,
8389,8419,8423,8429,8431,8443,8447,8461,8467,8501,
8513,8521,8527,8537,8539,8543,8563,8573,8581,8597,
8599,8609,8623,8627,8629,8641,8647,8663,8669,8677,
8681,8689,8693,8699,8707,8713,8719,8731,8737,8741,
8747,8753,8761,8779,8783,8803,8807,8819,8821,8831,
8837,8839,8849,8861,8863,8867,8887,8893,8923,8929,
8933,8941,8951,8963,8969,8971,8999,9001,9007,9011,
9013,9029,9041,9043,9049,9059,9067,9091,9103,9109,
9127,9133,9137,9151,9157,9161,9173,9181,9187,9199,
9203,9209,9221,9227,9239,9241,9257,9277,9281,9283,
9293,9311,9319,9323,9337,9341,9343,9349,9371,9377,
9391,9397,9403,9413,9419,9421,9431,9433,9437,9439,
9461,9463,9467,9473,9479,9491,9497,9511,9521,9533,
9539,9547,9551,9587,9601,9613,9619,9623,9629,9631,
9643,9649,9661,9677,9679,9689,9697,9719,9721,9733,
9739,9743,9749,9767,9769,9781,9787,9791,9803,9811,
9817,9829,9833,9839,9851,9857,9859,9871,9883,9887,
9901,9907,9923,9929,9931,9941,9949,9967,9973,10007,
10009,10037,10039,10061,10067,10069,10079,10091,10093,10099,
10103,10111,10133,10139,10141,10151,10159,10163,10169,10177,
10181,10193,10211,10223,10243,10247,10253,10259,10267,10271,
10273,10289,10301,10303,10313,10321,10331,10333,10337,10343,
10357,10369,10391,10399,10427,10429,10433,10453,10457,10459,
10463,10477,10487,10499,10501,10513,10529,10531,10559,10567,
10589,10597,10601,10607,10613,10627,10631,10639,10651,10657,
10663,10667,10687,10691,10709,10711,10723,10729,10733,10739,
10753,10771,10781,10789,10799,10831,10837,10847,10853,10859,
10861,10867,10883,10889,10891,10903,10909,10937,10939,10949,
10957,10973,10979,10987,10993,11003,11027,11047,11057,11059,
11069,11071,11083,11087,11093,11113,11117,11119,11131,11149,
11159,11161,11171,11173,11177,11197,11213,11239,11243,11251,
11257,11261,11273,11279,11287,11299,11311,11317,11321,11329,
11351,11353,11369,11383,11393,11399,11411,11423,11437,11443,
11447,11467,11471,11483,11489,11491,11497,11503,11519,11527,
11549,11551,11579,11587,11593,11597,11617,11621,11633,11657,
11677,11681,11689,11699,11701,11717,11719,11731,11743,11777,
11779,11783,11789,11801,11807,11813,11821,11827,11831,11833,
11839,11863,11867,11887,11897,11903,11909,11923,11927,11933,
11939,11941,11953,11959,11969,11971,11981,11987,12007,12011,
12037,12041,12043,12049,12071,12073,12097,12101,12107,12109,
12113,12119,12143,12149,12157,12161,12163,12197,12203,12211,
12227,12239,12241,12251,12253,12263,12269,12277,12281,12289,
12301,12323,12329,12343,12347,12373,12377,12379,12391,12401,
12409,12413,12421,12433,12437,12451,12457,12473,12479,12487,
12491,12497,12503,12511,12517,12527,12539,12541,12547,12553,
12569,12577,12583,12589,12601,12611,12613,12619,12637,12641,
12647,12653,12659,12671,12689,12697,12703,12713,12721,12739,
12743,12757,12763,12781,12791,12799,12809,12821,12823,12829,
12841,12853,12889,12893,12899,12907,12911,12917,12919,12923,
12941,12953,12959,12967,12973,12979,12983,13001,13003,13007,
13009,13033,13037,13043,13049,13063,13093,13099,13103,13109,
13121,13127,13147,13151,13159,13163,13171,13177,13183,13187,
13217,13219,13229,13241,13249,13259,13267,13291,13297,13309,
13313,13327,13331,13337,13339,13367,13381,13397,13399,13411,
13417,13421,13441,13451,13457,13463,13469,13477,13487,13499,
```

```
13513,13523,13537,13553,13567,13577,13591,13597,13613,13619,
13627,13633,13649,13669,13679,13681,13687,13691,13693,13697,
13709,13711,13721,13723,13729,13751,13757,13759,13763,13781,
13789,13799,13807,13829,13831,13841,13859,13873,13877,13879,
13883,13901,13903,13907,13913,13921,13931,13933,13963,13967,
13997,13999,14009,14011,14029,14033,14051,14057,14071,14081,
14083,14087,14107,14143,14149,14153,14159,14173,14177,14197,
14207,14221,14243,14249,14251,14281,14293,14303,14321,14323,
14327,14341,14347,14369,14387,14389,14401,14407,14411,14419,
14423,14431,14437,14447,14449,14461,14479,14489,14503,14519,
14533,14537,14543,14549,14551,14557,14561,14563,14591,14593,
14621,14627,14629,14633,14639,14653,14657,14669,14683,14699,
14713,14717,14723,14731,14737,14741,14747,14753,14759,14767,
14771,14779,14783,14797,14813,14821,14827,14831,14843,14851,
14867,14869,14879,14887,14891,14897,14923,14929,14939,14947,
14951,14957,14969,14983,15013,15017,15031,15053,15061,15073,
15077,15083,15091,15101,15107,15121,15131,15137,15139,15149,
15161,15173,15187,15193,15199,15217,15227,15233,15241,15259,
15263,15269,15271,15277,15287,15289,15299,15307,15313,15319,
15329,15331,15349,15359,15361,15373,15377,15383,15391,15401,
15413,15427,15439,15443,15451,15461,15467,15473,15493,15497,
15511,15527,15541,15551,15559,15569,15581,15583,15601,15607,
15619,15629,15641,15643,15647,15649,15661,15667,15671,15679,
15683,15727,15731,15733,15737,15739,15749,15761,15767,15773,
15787,15791,15797,15803,15809,15817,15823,15859,15877,15881,
15887,15889,15901,15907,15913,15919,15923,15937,15959,15971,
15973,15991,16001,16007,16033,16057,16061,16063,16067,16069,
16073,16087,16091,16097,16103,16111,16127,16139,16141,16183,
16187,16189,16193,16217,16223,16229,16231,16249,16253,16267,
16273,16301,16319,16333,16339,16349,16361,16363,16369,16381,
16411,16417,16421,16427,16433,16447,16451,16453,16477,16481,
16487,16493,16519,16529,16547,16553,16561,16567,16573,16603,
16607,16619,16631,16633,16649,16651,16657,16661,16673,16691,
16693,16699,16703,16729,16741,16747,16759,16763,16787,16811,
16823,16829,16831,16843,16871,16879,16883,16889,16901,16903,
16921,16927,16931,16937,16943,16963,16979,16981,16987,16993,
17011,17021,17027,17029,17033,17041,17047,17053,17077,17093,
17099,17107,17117,17123,17137,17159,17167,17183,17189,17191,
17203,17207,17209,17231,17239,17257,17291,17293,17299,17317,
17321,17327,17333,17341,17351,17359,17377,17383,17387,17389,
17393,17401,17417,17419,17431,17443,17449,17467,17471,17477,
17483,17489,17491,17497,17509,17519,17539,17551,17569,17573,
17579,17581,17597,17599,17609,17623,17627,17657,17659,17669,
17681,17683,17707,17713,17729,17737,17747,17749,17761,17783,
17789,17791,17807,17827,17837,17839,17851,17863
}, x);
```

Listing 5: `platform.lib`

```
//################################### platform.lib ##########################################
// A library to handle platform specific code in Faust. Its official prefix is `pl`.
//
// #### References
// * <https://github.com/grame-cncm/faustlibraries/blob/master/platform.lib>
//########################################################################################
// It can be reimplemented to globally change the SR and the tablesize definitions


/************************************************************************
************************************************************************
FAUST library file
Copyright (C) 2020 GRAME, Centre National de Creation Musicale
----------------------------------------------------------------------
This program is free software; you can redistribute it and/or modify
```

```
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation; either version 2.1 of the
License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with the GNU C Library; if not, write to the Free
Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
02111-1307 USA.

EXCEPTION TO THE LGPL LICENSE : As a special exception, you may create a
larger FAUST program which directly or indirectly imports this library
file and still distribute the compiled code generated by the FAUST
compiler, or a modified version of this compiled code, under your own
copyright and license. This EXCEPTION TO THE LGPL LICENSE explicitly
grants you the right to freely choose the license for the resulting
compiled code. In particular the resulting compiled code has no obligation
to be LGPL or GPL. For example you are free to choose a commercial or
closed source license or any other license if you decide so.
************************************************************************
************************************************************************/

declare name "Generic Platform Library";
declare version "1.3.0";

//---------------------------------`(pl.)SR`-----------------------------------
// Current sampling rate (between 1 and 192000Hz). Constant during
// program execution. Setting this value to a constant will allow the
// compiler to optimize the code by computing constant expressions at
// compile time, and can be valuable for performance, especially on
// embedded systems.
//-----------------------------------------------------------------------------
SR = min(192000.0, max(1.0, fconstant(int fSamplingFreq, <math.h>)));

//---------------------------------`(pl.)BS`---------------------------------------
// Current block-size (between 1 and 16384 frames). Can change during the execution.
//-----------------------------------------------------------------------------
BS = min(16384.0, max(1.0, fvariable(int count, <math.h>)));

//---------------------------------`(pl.)tablesize`----------------------------
// Oscillator table size. This value is used to define the size of the
// table used by the oscillators. It is usually a power of 2 and can be lowered
// to save memory. The default value is 65536.
//-----------------------------------------------------------------------------
tablesize = 1 << 16;
```