

# Computer Instruments and Unit Generators

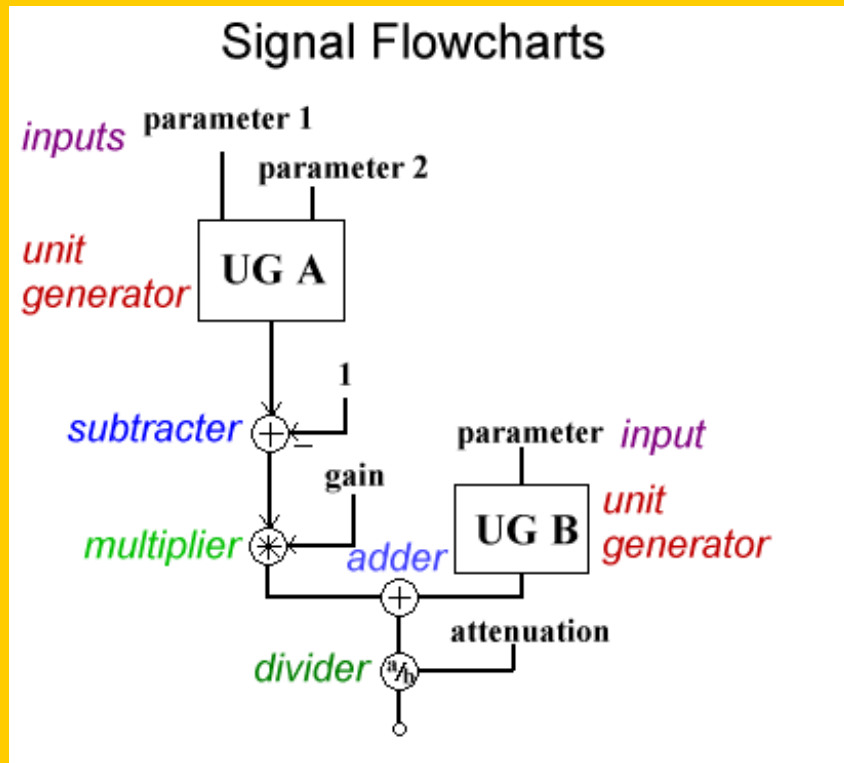
The term *instrument* refers to a processing algorithm which performs a music/sound event. Instruments are made of smaller units called *unit generators* or **UGs**.

Unit generators are based on algorithms designed to perform specific signal processing tasks, such as generating a wave or shaping the amplitude of a sound.

UGs are the building blocks provided by computer music languages to generate and process signals.

# Flowcharts

**Signal Flowcharts** are used to show graphically the interconnection of UGs and the flow of signal in an instrument.



These can be translated into the code that represent the instrument in a computer music language, such as csound.

## GEN subroutines and Function tables

**GEN** routines are an efficient way of handling functions which might be needed by some UGs in order to create or modify sounds.

They are used to generate pre-calculated (*function*) **tables** of *fixed size* containing the results of function evaluation for certain ranges and parameters.

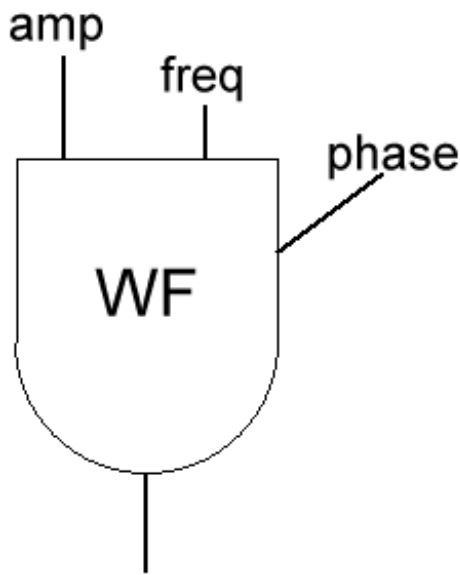
Instead of calculating the value of a function every time it is needed, certain UGs simply read from a pre-calculated table. An oscillator is a typical example of such UG.

**Csound** provides a full set of GEN routines for several types of functions.

# Oscillators

Oscillators are the basic signal generators in a computer sound synthesis system. They generate a periodic waveform.

Typical parameters are: **Frequency** (freq), **Amplitude** (amp), **Phase offset**, and **Waveform** (WF). An oscillator is represented in a signal flowchart by the symbol below:



# Wavetables

**Oscillators** generally depend on a stored wavetable, which is continuously sampled in order to generate the oscillator periodic output.

The wavetable is a *function table* which holds one or more cycles of a waveform. Wavetables can be generated by *GEN routines*.

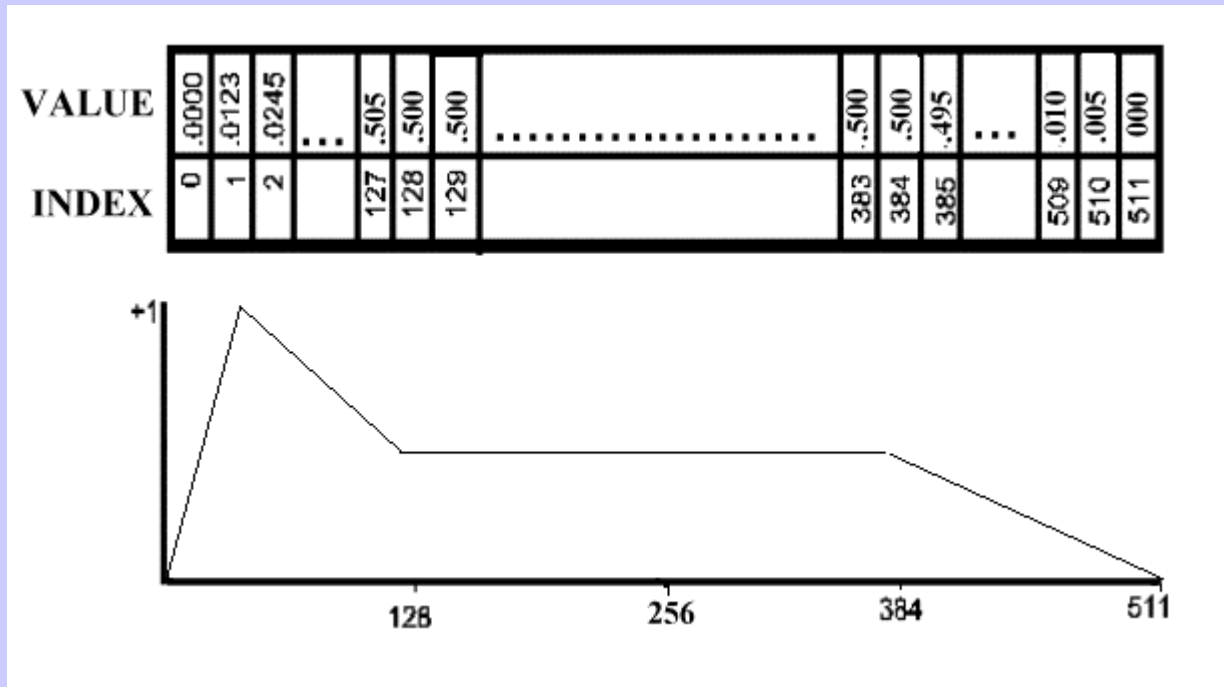
In **csound**, GENs can be used to fill a table, by, for instance:

- **creating waveshapes** based on mixed sinusoids of different frequencies, amplitudes and phases.
- **reading samples** off a soundfile.



# Tables and envelopes

**Envelope shapes** are another use for function tables.



**GEN 7** in csound can be used for such tables.

## Table Lookup

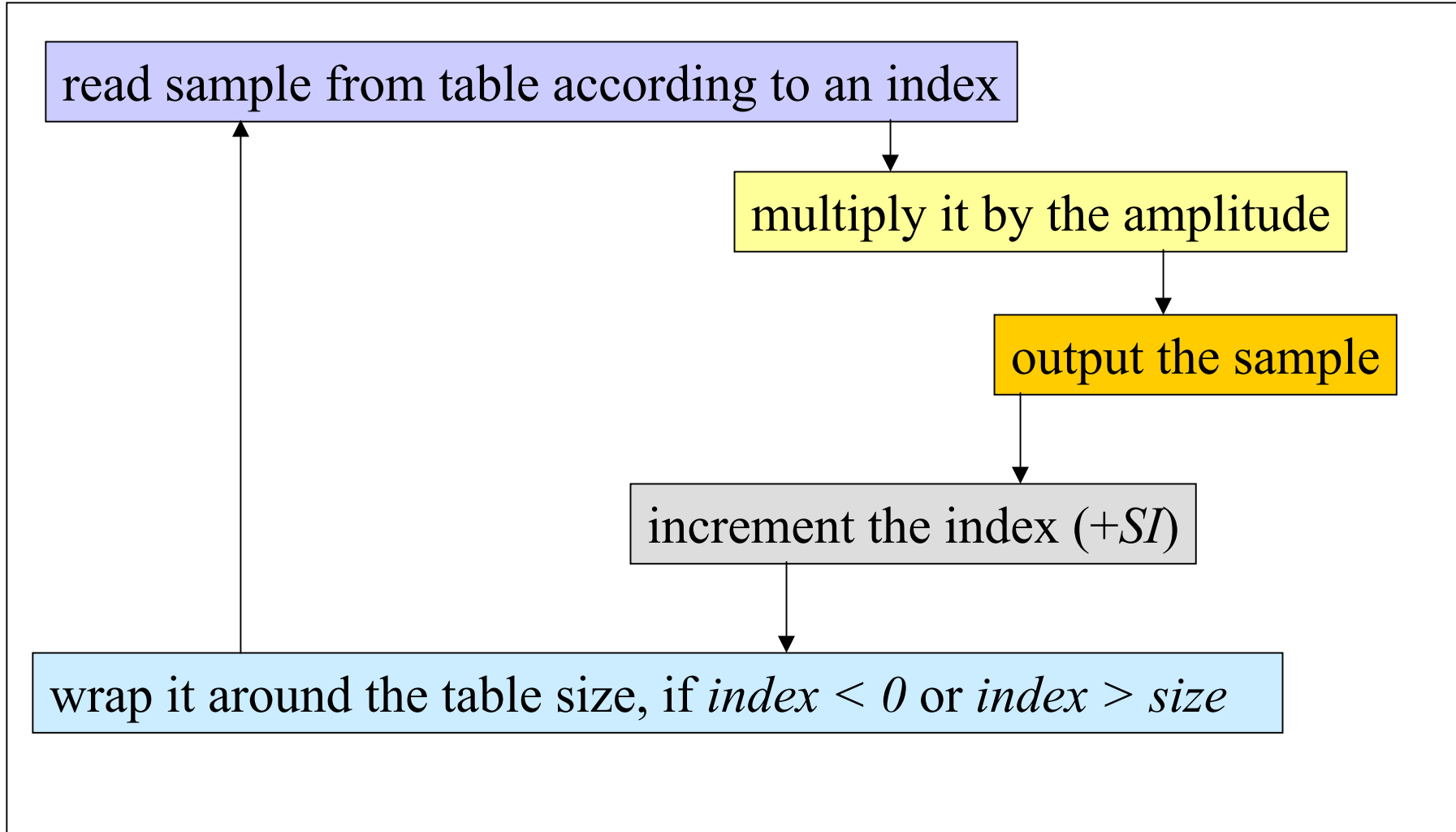
An oscillator continuously looks up values in the function table according to the *fundamental frequency* (rate), the *sampling rate* (SR) and the **table size** (its number of elements).

It reads a position in the table, then skips to another one according to a *sampling increment*  $SI$ , defined by

$$SI = \text{tablesize} \times (\text{fundamental} / SR)$$

The values on the table are usually normalised (i.e. between 0 and  $\pm 1$ ). The **amplitude** scales them to the desired output range.

## How an oscillator works



## Precision

When the index is *non-integral* (not a whole number), the oscillator has to decide which table position to read.

**Truncation:** the decimal part is discarded (pos 3.2 => pos 3). This is fast, but errors are generated, which will degrade the signal-to-noise ratio (SNR). For a 512-size table, the SNR is 42dB

**Interpolation:** adjacent values are interpolated (3.2 => 80% of pos 3 and 20% of pos 4). Slower, but more precise and the SNR is better. For 512 points, the SNR is 96dB.

For any type of lookup, the *larger* the table the *better* the SNR.

## Oscillators in csound

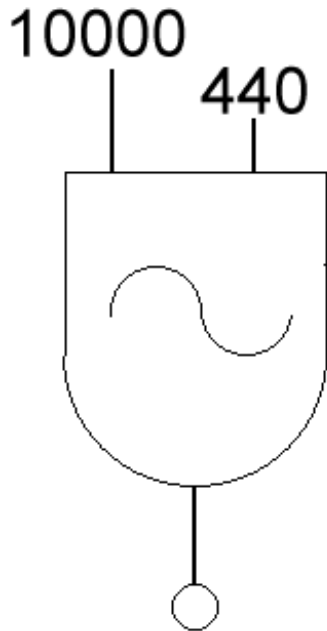
The basic **oscillator UGs** are defined by the *opcodes* **oscil** (truncating) and **oscili** (interpolating), eg.:

```
aout oscil amp,afreq,itable[,iphase]
```

the line above defines a truncating oscillator:

- *Output* is stored in the variable *aout*
- *Parameters* are defined by the variables *amp*, *afreq* and *itable*
- *Amplitude* is *amp*, *frequency* is *freq*.
- The *number of the function table* used by this *oscil* is *itable*.
- The *phase offset* parameter *iphase* is *optional*.
- Parameters are *separated by commas*. The opcode is separated from the variables by *spaces*. This applies to *all opcodes*!

## A very simple example



```
sr=44100
kr=100
ksmps=441
nchnls=1

instr 1

asig oscil 10000,440,1
      out asig

endin
```

```
f1 0 1024 10 1 ; sine wave
i1 0 2 ; active from 0, for 2 secs
```

An oscillator, using a sine wave, with a **fixed** amplitude of 10000 and **fixed** frequency of 440 (Hz)

## Defining a wavetable

**Wavetables** (and other tables) are generated using **GEN** routines. These are defined by score **F-statements**

A line starting with **f** defines a gen function

```
f1 0 1024 10 1 ; defines a sine wave
```

This is interpreted as follows:

- *function table* number 1,
- *created at time* 0 (start of synthesis)
- *size* 1024 (number of points of the table)
- *using* GEN 10 (generates harmonic functions)
- *harmonic 1 amplitude* 1 (and only one harmonic defined)

## More on Harmonics

The previous example generates a pure sinewave (1 harmonic only). In order to create waves with different shapes, more harmonics are added together:

A **sawtooth** wave contains any number of harmonics with *amplitudes* equal to  $1/\text{harmonic\_number}$  (1, 1/2, 1/3, 1/4 etc):

```
f1 0 1024 10 1 .5 .3 .25 ; wave w/ 4 harmonics
```

A **square** wave contains any number of harmonics, but *no even harmonics*. Amplitudes are equal to  $1/\text{harmonic\_number}$  :

```
f1 0 1024 10 1 0 .3 0 .2 ; wave w/ 3 odd harmonics
```

**For more, see the definition of GEN10 in the csound manual !**

## Controlling instrument parameters from the score

Only **p1**, **p2** and **p3** have fixed meaning. Other **p-fields** are *instrument-assigned*. For instance, you might want to use **p4** to control *amplitude* and **p5** to control *frequency*.

```
;instr start dur amp freq  
i1      0      5  10000 220
```

Now you need to use **p4** and **p5** in your instrument:

```
instr 1  
asnd  oscil p4, p5, 1  
      out  asnd  
endin
```

# Envelopes

An important UG is the **envelope** shaper, which can create shapes that can sculpt amplitudes (and other parameters). The **linear envelope** generator **linen** is the simplest one.

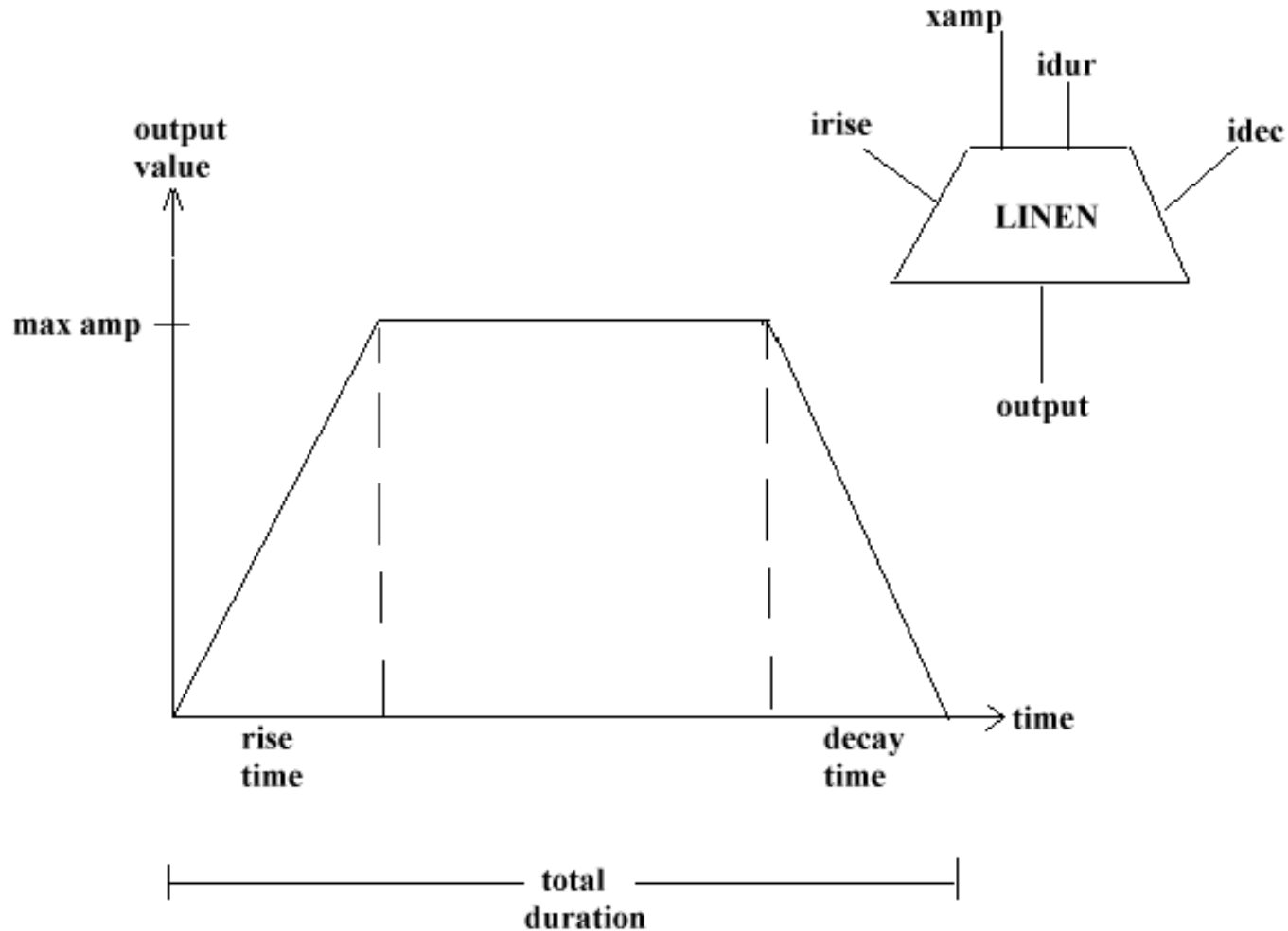
```
kenv linen kamp,irise,idur,idec
```

It creates a **3-stage** envelope:

- *max amp*: kamp
- *rise time (attack)*: irise
- *total duration*: idur
- *decay time*: idec

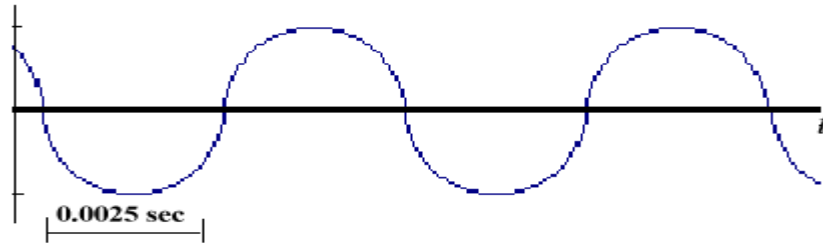
The output variable `kenv` in the above example receives the envelope output.

# Linen: flowchart symbol and parameter details

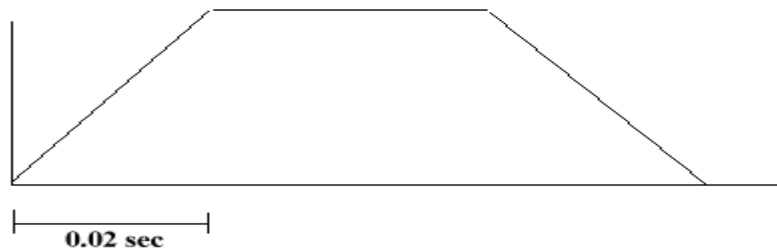


# Control Rates

If we compare the rate of change of a waveform with that of an envelope, we'll see that the envelope changes more slowly



1/2 cycle of a 500Hz wave lasts for 0.0025 sec (2.5 ms)



A medium/slow attack takes 0.02 sec (20 ms).

A slower rate is used for **control signals** such as envelopes. This is called the **control rate (kr)**, or **k-rate**.

## Csound update rates

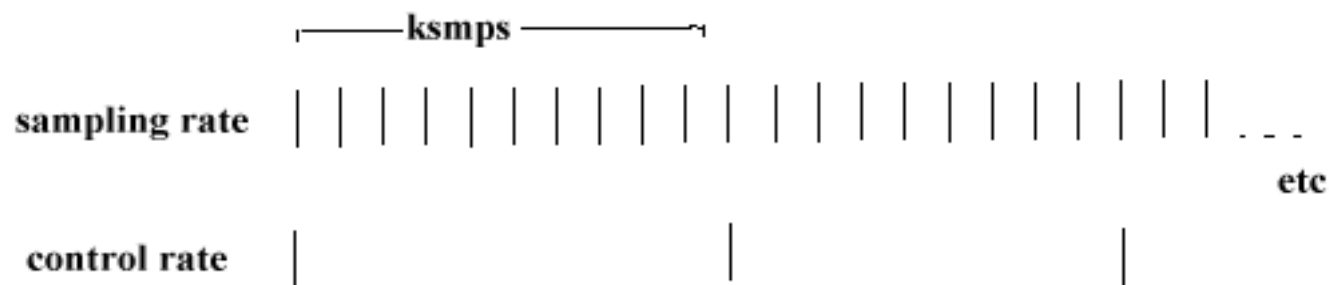
In fact, csound has **3 update rates**:

**i-time: initialisation**, variables updated every new activation time.

**k-rate: control**, variables updated every  $1/kr$  seconds

**a-rate: audio**, variables updated every  $1/sr$  seconds

Csound audio is generated in blocks of *ksmps* samples at the sampling rate, every control (*k-*)period:



## Orchestra variable types

Variables types reflect this update scheme:

***i-time variables*** always start with an **i** and are updated at the *initialisation* time only. Examples: **idur**, **irise**, **ianything**.

***k-rate variables*** always start with a **k** and are updated *every control period*, used for control signals. They hold a *single sample*. Examples: **kamp**, **kfreq**, **kanything**.

***a-rate variables*** always start with an **a** and are updated *every sampling period*. They hold a whole *block of samples*. Examples: **aout**, **aleft**, **asig**, **aanything**.

The sampling rate is the number of *samples* generated per second, whereas the control rate is the number of *ksmps*-sample *blocks* per second.

## Assignment and arithmetic operators

Any of +, -, \* or / can be used with variables. Brackets ( ) can be used to group operations.

Also the *assignment* operator (=) can be used to set variables to any value.

Examples:

```
igain = 0.5
```

```
amix = aoscil1*igain + aoscil2*igain
```

```
a1 oscil k1, (k1/1000)+100, 1
```

## UG versions

Many UGs have two versions, one for audio signal output and another for control signal generation, eg.:

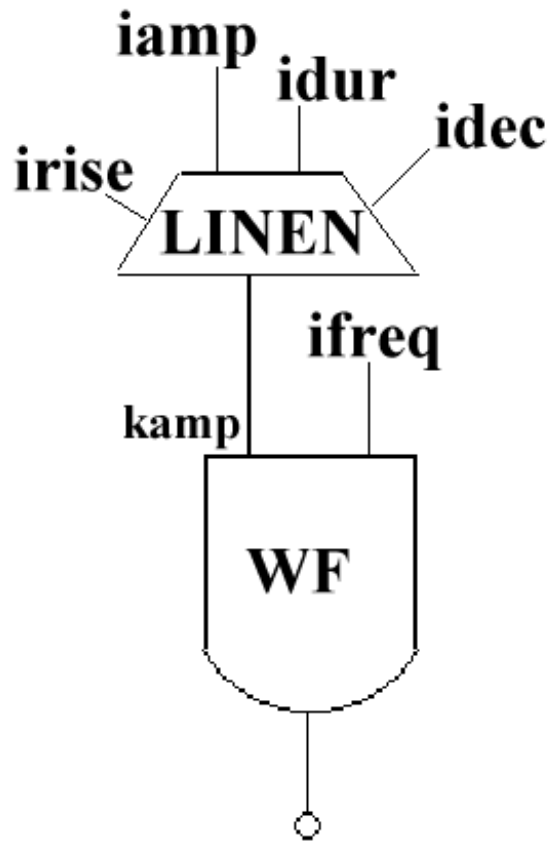
```
kr  linen  kamp, irise, idur, idec  
ar  linen  xamp, irise, idur, idec
```

The output variable determines which version is used. The first linen generates a **k-rate signal** and the second an **a-rate** signal.

The 'x' in `xamp` means the parameter can be an *a*-, *k*- or *i*-type variable or a constant.

*K*-rate parameters can generally also receive *i*-time variables or constants. *I*-time parameters can only take that type of variable (or a constant).

# Using linen as a signal generator

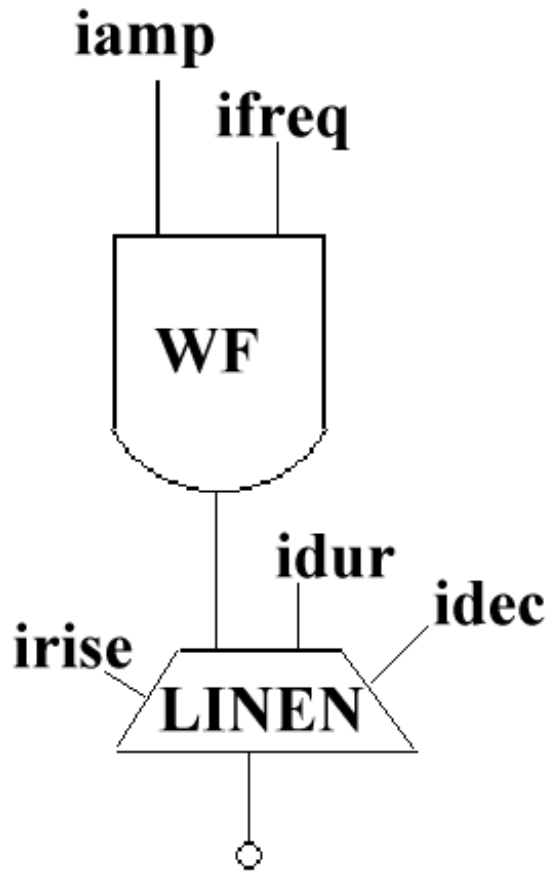


```
instr 1  
  
idur = p3  
iamp = p4  
ifreq = p5  
irise = .02  
idec = .02
```

```
kamp linen    iamp,irise,idur,idec  
aout oscil   kamp,ifreq,1  
      out     aout
```

```
endin
```

# LINEN as a signal processor



```
instr 1

idur = p3
iamp = p4
ifreq = p5
irise = .02
idec = .02

aosc oscil iamp,ifreq,1
aout linen aosc,irise,idur,idec
out aout

endin
```

## Other control signal generators

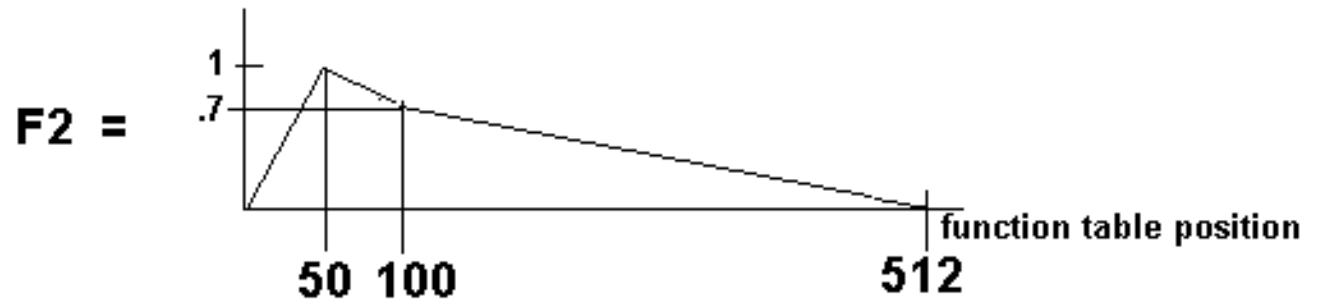
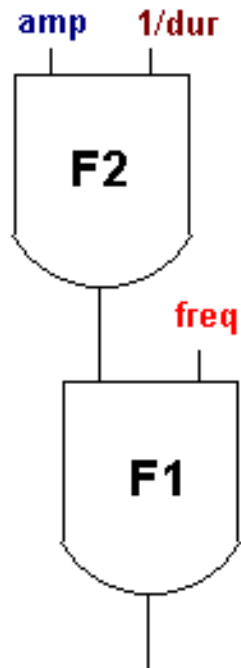
A series of different UGs can be used, as **linen**, to generate linear (or exponential) control signals, such as:

- **line** and **linseg**: generate linear segments of lines between two or more points:

```
k1 line 100,p3,1000 ;a line from 100 to 1000 in p3 secs
```

- **expon** and **expseg**: *exponential* line segments.
- **adsr**: *4-stage* attack-decay-sus-release envelope.
- **envlpx**: an *advanced version* of `linen`.

# Table-lookup Oscillators as Envelope Generators



**F1 = any waveform**

Function  $F2$  above can be generated using **GEN 7**:

```
f2 0 513 7 0 50 1 50 0.7 412 0
```

**K-rate** versions of `oscil/oscili` are used for this purpose.

## Table-lookup with `oscil1` and `oscil1i`

The use of oscillators to *look up* tables is so common that specialised oscillator UGs are present to do this:

```
kr oscil1 idel, kamp, idur, ifn  
kr oscil1i idel, kamp, idur, ifn
```

Instead of frequency, these oscillators have an *idur* parameter, which is the total duration of the lookup operation.

They also have an *idel* parameter that defines an initial **delay** in the lookup operation, if necessary.

These oscillators run just *once* through the table in *idur* secs, after a delay of *idel* secs (which can be 0).

## Some Envelope-table GENs

**GEN 5** and **7** create tables of linear and exponential line segments, respectively. The segments are defined by **values** separated by **table positions**, as in (*starting from p-field 5*):

**0 256 1 256 0**

(Read: *a line from 0 to 1 in 256 points and back to 0 in 256 points*)

For GEN 5, 0 values are not allowed (it is an exponential)

**GEN 25** and **27** are the same as 5 and 7, except that the segments are defined by *breakpoints*:

**0 256 1 512 0**

(*a line from 0 reaching 1 at point 256 and back to 0 at 512*)

## Table size and additional guard-point

Tables sizes are specified as either *powers-of-2* (2,4,...,512,1024 etc), or *powers-of-two plus one* (3,5,...,513,1025 etc).

The *actual* size is always power-of-two plus one. The last point is called a *guard point*. It will contain:

- a **copy** of the **first position** value, if size is a *power-of-two*.
- an extension of the contour of the table (called an *extended guard point*), if size is a *power-of-two plus one*.

So we should use:

- *Power-of-two plus one* tables for single-scan uses (e.g. envelopes).
- *Power-of-two* tables for wrap-around uses (e.g. oscillators).

## Summary: Key Concepts

- *Instruments* are built with UGs. In **csound** these are defined by *opcodes*.
- *Oscillators* are basic UGs for periodic signal generation.
- *Oscillators* read from tables, defined with GEN subroutines.
- *Envelope generators* create control signals for parameter shaping. There are several types of envelope UGs.
- Csound has *three types of variables*: i-, k- and a-rate types that are used to hold single values, control and audio signals, respectively.
- *Arithmetic and assignment* operators can be used freely.
- Many UGs have *control* and *audio rate versions*.
- *Oscillators* can also generate *envelope* control signals using single-scan envelope tables.