

MANUAL PÚBLICO DE REFERENCIA DE CSOUND

TRADUCCIÓN AL ESPAÑOL DE LA VERSIÓN 4.0

por Barry Vercoe, Media Lab M.I.T.
y colaboradores

Versión canónica inglesa editada por
John ffitch, Richard Boulanger,
Jean Piché y David Boothe

Versión española traducida y revisada
por Servando Valero.

Nota sobre los Derechos de Autor

Copyright 1986, 1992 Instituto Tecnológico de Massachusetts. Todos los derechos reservados.

Desarrollado por **Barry L. Vercoe** en el Estudio de Música Experimental (Media Laboratory) del MIT, en Cambridge, Massachusetts, con el apoyo parcial de la Fundación para el Desarrollo de Sistemas y de la Beca # IRI-8704665 de la Fundación para las Ciencias Naturales.

Se concede por la presente permiso para usar, copiar o modificar estos programas y su documentación únicamente con propósitos educativos o de investigación y sin derechos, siempre y cuando este Copyright y esta nota de permiso aparezcan en todas las copias y su documentación. Para cualquier otro uso de este software, en su forma original o modificada, incluyendo, pero no limitado a, su distribución en todo o en parte, deberá obtenerse permiso previo del MIT. El MIT no se hace responsable del uso de este software para cualquier propósito. Se proporciona tal cual, sin garantía explícita o implícita. Extended Csound es desarrollado por Analog Devices, Norwood, MA.

*La Edición original hipertexto del Manual del Csound del MIT fue preparada para la World Wide Web por **Peter J. Nix** del Departamento de Música de la Universidad de Leeds y **Jean Piché** de la Facultad de Música de la Universidad de Montreal. La versión impresa, en la cual está basada esta traducción de **Servando Valero**, es la Edición Impresa en Inglés del Manual, preparada por **David M. Boothe**, conforme a la Edición original Hipertexto. Los editores reconocen plenamente los derechos de los autores de los programas y documentación originales, como se declara arriba y, asimismo, piden que esta nota aparezca dondequiera que se use este material.*

Colaboradores

Además del núcleo de código desarrollado por Barry Vercoe en el MIT, una gran parte del código de Csound fue modificado, desarrollado y ampliado por un grupo independiente de programadores, compositores y científicos. Los derechos de autor de este código pertenecen a los respectivos autores:

Mike Berry	Matt Ingalls
Eli Breder	Richard Karpen
Michael Casey	Victor Lazzarini
Michael Clark	Allan Lee
Perry Cook	David Macintyre
Sean Costello	Peter Neubäcker
Richard Dobson	Marc Resibois
Mark Dolson	Gabriel Maldonado
Rasmus Ekman	Hans Mikelson
Dan Ellis	Paris Smaragdis
Tom Erbe	Greg Sullivan
John ffitch	Robin Whittle
Bill Gardner	

La versión inglesa original de este manual fue compilada según el Manual canónico de Csound, mantenido por John ffitch, Richard Boulanger, Jean Piché y David Boothe.

Esta versión española está basada en la edición inglesa original y es traducida, revisada y mantenida por Servando Valero Castiñeira.

Índice

Nota sobre los Derechos de Autor

Colaboradores

Índice

1 Prefacio

- 1.1 Dónde conseguir la Versión Pública de Csound y su Manual
- 1.2 Cómo usar el Manual de Csound
- 1.3 La Lista de Correo de Csound

2 Sintaxis de la Orquesta

- 2.1 Directorios y Ficheros
- 2.2 Nomenclatura
- 2.3 Tipos de Sentencia de la Orquesta
- 2.4 Constantes y Variables
- 2.5 Expresiones

3 Sintaxis de la Orquesta: Sentencias de la Cabecera de la Orquesta

- 3.1 sr, kr, ksmps, nchnls
- 3.2 strset, pset
- 3.3 seed
- 3.4 ftgen
- 3.5 massign, ctrlinit

4 Sintaxis de la Orquesta: Sentencias de Bloques de Instrumentos

- 4.1 instr, endin

5 Sintaxis de la Orquesta: Inicialización de Variables

- 5.1 =, init, tival, divz

6 Control de los Instrumentos: Llamadas a Instrumentos

- 6.1 schedule, schedwhen
- 6.2 schedkwhen
- 6.3 turnon

7 Control de los Instrumentos: Sentencias de Control de la Duración

- 7.1 ihold, turnoff

8 Control de los Instrumentos: Control de Ejecuciones en Tiempo Real

- 8.1 active
- 8.2 cpuprc, maxalloc, prealloc

9 Control de los Instrumentos: Lectura de Tiempo

- 9.1 timek, times, timeinstk, timeinsts

10 Control de los Instrumentos: Control del Reloj

- 10.1 clockon, clockoff, readclock

11 Control de los Instrumentos: Detección y Control

- 11.1 pitch
- 11.2 pitchamdf
- 11.3 tempest
- 11.4 follow
- 11.5 trigger
- 11.6 peak
- 11.7 xyin, tempo

12 Control de los Instrumentos: Valores Condicionales

- 12.1 >, <, >=, <=, ==, !=, ?

13 Control de los Instrumentos: Macros

- 13.1 #define, \$NOMBRE, #undef
- 13.2 #include

14 Control de los Instrumentos: Control del Flujo del Programa

- 14.1 igoto, tigoto, kgoto, goto, if, timeout

15 Control de los Instrumentos: Reinicialización

- 15.1 reinit, rigoto, rireturn

16 Operaciones Matemáticas: Operaciones Aritméticas y Lógicas

- 16.1 -, +, &&, ||, *, /, ^, %

17 Operaciones Matemáticas: Funciones Matemáticas

- 17.1 int, frac, i, abs, exp, log, log10, sqrt
- 17.2 powoftwo, logbtwo

18 Operaciones Matemáticas: Funciones Trigonómicas

- 18.1 sin, cos, tan, sininv, cosinv, taninv, sinh, cosh, tanh

19 Operaciones Matemáticas: Funciones de Amplitud

19.1 dbamp, ampdb

20 Operaciones Matemáticas: Funciones Aleatorias

20.1 rnd, birnd

21 Operaciones Matemáticas: Opcodes Equivalentes a Funciones

21.1 sum

21.2 product

21.3 pow

21.4 taninv2

21.5 mac, maca

22 Convertidores de Altura: Funciones

22.1 octpch, pchoct, cpspch, octcps, cpsoct

23 Convertidores de Altura: Opcodes de Afinación

23.1 cps2pch, cpsxpch

24 Soporte MIDI: Convertidores

24.1 notnum, veloc, cpsmidi, cpsmidib, octmidi, octmidib, pchmidi, pchmidib, ampmidi, aftouch, pchbend, midictrl

24.2 cpstmid

25 Soporte MIDI: Entrada de Controladores

25.1 initc7, initc14, initc21

25.2 midic7, midic14, midic21, ctrl7, ctrl14, ctrl21

25.3 chanctrl

26 Soporte MIDI: Bancos de Mandos Deslizantes

26.1 slider8, slider16, slider32, slider64, slider8f, slider16f, slider32f, slider64f, s16b14, s32b14

27 Soporte MIDI: Entrada / Salida Genérica

27.1 midiin

27.2 midiout

28 Soporte MIDI: Activación / Desactivación de Nota

28.1 noteon, noteoff, noteondur, noteondur2

28.2 moscil, midion

28.3 midion2

29 Soporte MIDI: Salida de Mensajes MIDI

29.1 outic, outkc, outic14, outkc14, outipb, outkpb, outiat, outkat, outipc, outkpc, outipat, outkpat

29.2 nrpn

29.3 mdelay

30 Soporte MIDI: Mensajes en Tiempo Real

30.1 mclock, mrtmsg

31 Soporte MIDI: Extendedores de Eventos

31.1 xtratim, release

32 Generadores de Señal: Generadores Lineales y Exponenciales

32.1 line, expon, linseg, linsegr, expseg, expsegr, expsega

32.2 adsr, madsr, xadsr, mxadsr

33 Generadores de Señal: Acceso a Tablas

33.1 table, tablei, table3, oscil1, oscil1i, osciln

34 Generadores de Señal: Generadores de Fase

34.1 phasor

34.2 phasorbnk

35 Generadores de Señal: Osciladores Básicos

35.1 oscil, oscili, oscil3

35.2 poscil, poscil3

35.3 lfo

36 Generadores de Señal: Osciladores de Espectro Dinámico

36.1 buzz, gbuzz

36.2 vco

37 Generadores de Señal: Síntesis y Resíntesis Aditiva

37.1 adsyn

37.2 adsynt

37.3 hsboscil

38 Generadores de Señal: Síntesis FM

38.1 foscil, foscili

38.2 fmvoice

38.3 fmbell, fmrhode, fmwurlie, fmmetal, fmb3, fmpercfl

39 Generadores de Señal: Reproducción de Muestras

- 39.1 loscil, loscil3
- 39.2 lposcil, lposcil3

40 Generadores de Señal: Síntesis Granular

- 40.1 fof, fof2
- 40.2 fog
- 40.3 grain
- 40.4 granule
- 40.5 sndwarp, sndwarpst

41 Generadores de Señal: Modelado Físico Waveguide

- 41.1 pluck
- 41.2 wtpluck
- 41.3 repluck, wtpluck2
- 41.4 wgbow
- 41.5 wgflute
- 41.6 wgbrass
- 41.7 wgclar

42 Generadores de Señal: Modelos y Emulaciones

- 42.1 moog
- 42.2 shaker
- 42.3 marimba, vibes
- 42.4 mandol
- 42.5 gogobel
- 42.6 voice
- 42.7 lorenz
- 42.8 planet

43 Generadores de Señal: Resíntesis STFT (Vocoding)

- 43.1 pvoc
- 43.2 pvread, pvbufread, pvinterp, pvcross, tableseg, tablexseg, vpvoc
- 43.3 pvadd

44 Generadores de Señal: Resíntesis LPC

- 44.1 lpread, lpreson, lpfreson
- 44.2 lpslot, lpinterp

45 Generadores de Señal: Generadores Aleatorios (Ruido)

- 45.1 rand, randh, randi
- 45.2 x-class noise generators

46 Control de Tablas de Función: Consulta a Tablas

- 46.1 ftlen, ftlptim, ftsr, nsamp
- 46.2 tableng

47 Control de Tablas de Función: Selección de Tablas

47.1 tablekt, tableikt

48 Control de Tablas de Función: Operaciones de Lectura y Escritura

48.1 tableiw, tablew, tablewkt

48.2 tablegpw, tablemix, tablecopy, tableigpw, tableimix, tableicopy

48.3 tablera, tablewa

49 Modificadores de Señal: Filtros Estándar

49.1 port, portk, tone, tonek, atone, atonek, reson, resonk, areson, aresonk

49.2 tonex, atonex, resonx

49.3 resonr, resonz

49.4 resony

49.5 lowres, lowresx

49.6 vlowres

49.7 lowpass2

49.8 biquad, rezy, moogvcf

49.9 svfilter

49.10 hilbert

49.11 butterhp, butterlp, butterbp, butterbr

49.12 filter2, zfilter2

50 Modificadores de Señal: Filtros Especializados

50.1 nlfilt

50.2 pareq

50.3 dcblock

51 Modificadores de Señal: Modificadores de Envoltente

51.1 linen, linenr, envlpx, envlpxr

52 Modificadores de Señal: Modificadores de Amplitud

52.1 rms, gain, balance

52.2 dam

53 Modificadores de Señal: Limitadores de Señal

53.1 limit, mirror, wrap

54 Modificadores de Señal: Líneas de Retardo

54.1 delayr, delayw, delay, delay1

54.2 deltap, deltapl, deltapn, deltap3

54.3 multitap

54.4 vdelay, vdelay3

55 Modificadores de Señal: Reverberación

- 55.1 comb, alpass, reverb
- 55.2 reverb2, nreverb
- 55.3 nestedap

56 Modificadores de Señal: Guías de Onda

- 56.1 wguide1, wguide2
- 56.2 streson

57 Modificadores de Señal: Efectos Especiales

- 57.1 harmon
- 57.2 flanger
- 57.3 distort1
- 57.4 phaser1, phaser2

58 Modificadores de Señal: Convolución y Transformación (Morphing)

- 58.1 convolve
- 58.2 cross2

59 Modificadores de Señal: Panoramización y Espacialización

- 59.1 pan
- 59.2 locsig, locsend
- 59.3 space, spsend, spdist
- 59.4 hrtfer

60 Modificadores de Señal: Operadores a Nivel de Muestra

- 60.1 samphold, downsamp, upsamp, interp, integ, diff
- 60.2 ntrpol
- 60.3 fold

61 El Sistema de Cableado Zak

- 61.1 zakinit
- 61.2 ziw, zkw, zaw, ziwm, zkwm, zawm
- 61.3 zir, zkr, zar, zarg
- 61.4 zkmod, zamod, zkcl, zacl

62 Operaciones Usando Tipos de Datos Espectrales

- 62.1 specaddm, specdiff, specscal, spechist, specfilt
- 62.2 specptrk
- 62.3 specsum, specdisp
- 62.4 spectrum

63 Entrada y Salida de Señal: Entrada de Señal

- 63.1 in, ins, inq, inh, ino, soundin, diskIn

64 Entrada y Salida de Señal: Salida de Señal

64.1 soundout, soundouts, out, outs1, outs2, outs, outq1, outq2, outq3, outq4, outq, outh, outo

65 Entrada y Salida de Señal: Entrada y Salida de Ficheros

65.1 dumpk, dumpk2, dumpk3, dumpk4, readk, readk2, readk3, readk4

65.2 fout, foutk, fouti, foutir, fiopen

65.3 fin, fink, fini

65.4 vincr, clear

66 Entrada y Salida de Señal: Consultas a Ficheros de Sonido

66.1 filelen, filesr, filenchnls, filepeak

66.2 print, display, dispfft

66.3 printk, printks

66.4 printk2

67 La Partitura Numérica Estándar

67.1 Preprocesado de Partituras Estándar

67.2 Símbolos de Siguiente-P y Previo-P

67.3 Ramping

67.4 Macros de la Partitura

67.5 Partitura con Múltiples Ficheros

67.6 Evaluación de Expresiones

67.7 Sentencia f (Sentencia de Tabla de Función)

67.8 Sentencia i (Sentencia de Instrumento o Nota)

67.9 Sentencia a (Sentencia de Avance)

67.10 Sentencia t (Sentencia de Tempo)

67.11 Sentencia b

67.12 Sentencia v

67.13 Sentencia s

67.14 Sentencia e

67.15 Sentencia r (Sentencia de Repetición)

67.16 Sentencia m (Sentencia de Marca)

67.17 Sentencia n

68 Rutinas GEN

68.1 GEN01

68.2 GEN02

68.3 GEN03

68.4 GEN04

68.5 GEN05, GEN07

68.6 GEN06

68.7 GEN08

68.8 GEN09, GEN10, GEN19

68.9 GEN11

68.10 GEN12

68.11 GEN13, GEN14

68.12 GEN15

68.13 GEN17

68.14 GEN20

68.15 GEN21

68.16 GEN23

68.17 GEN25, GEN27

68.18 GEN28

69 La Orden Csound

- 69.1 Orden de Precedencia
- 69.2 Indicadores Genéricos
- 69.3 Indicadores Específicos de PC Windows
- 69.4 Indicadores Específicos de Macintosh
- 69.5 Descripción

70 Formato de Fichero Unificado para Orquestas y Partituras

- 70.1 Descripción
- 70.2 Formato de Fichero de Datos Estructurados
- 70.3 Ficheros de Parámetros de la Línea de Comando

71 Preprocesado de Ficheros de Partitura

- 71.1 La opción Extract
- 71.2 Preprocesado Independiente con Scsort

72 Utilidades para Ficheros de Sonido

- 72.1 sndinfo
- 72.2 hetro
- 72.3 lpanal
- 72.4 pvanal
- 72.5 cvanal
- 72.6 pvlook

73 Cscore

- 73.1 Eventos, Listas y Operaciones
- 73.2 Escribiendo un Programa Principal
- 73.3 Ejemplos más Avanzados
- 73.4 Compilando un Programa de Cscore

74 Añadiendo tus propios Módulos a Csound

75 Apéndice A: Información Miscelánea

- 75.1 Conversión de Alturas
- 75.2 Valores de Intensidad Sonora (para un tono a 1000 Hz)
- 75.3 Valores de Formantes
- 75.4 Funciones de Ventana

Referencia Rápida de Csound

1 PREFACIO AL MANUAL DE CSOUND

por Barry Vercoe, MIT Media Lab

La realización de música por ordenador conlleva sintetizar señales de audio digitales, compuestas de muestras tomadas a intervalos discretos de tiempo, que representan formas de onda continuas. Existen varias maneras de hacer esto, proporcionando cada una de ellas una forma de control diferente. La síntesis directa genera formas de onda muestreando (o sampleando, como se suele decir en nuestra jerga, del inglés *sampling*) una función almacenada en memoria que representa un sólo ciclo de dicha onda. La síntesis aditiva genera todos los parciales de una forma de onda compleja, cada uno con su propio envolvente dinámico. La síntesis subtractiva toma un sonido complejo para luego filtrarlo. La síntesis no lineal usa modulación de frecuencia y modelado de onda (*waveshaping*) para dar características complejas a señales sencillas. Por otra parte, el muestreo y el almacenamiento de sonidos naturales permiten su uso posterior a voluntad.

Ya que una especificación detallada del sonido en cada instante resultaría tediosa, el control sobre éste se consigue en Csound de dos maneras: 1) mediante los instrumentos de una orquesta, y 2) mediante los eventos de la partitura. Pero en el contexto de Csound, una orquesta (*orchestra*) es en realidad un programa de ordenador capaz de generar sonido, mientras que por partitura (*score*) entendemos un fichero de datos con los que dicho programa puede trabajar. Que el ataque de una nota sea una constante fija en un instrumento, o por el contrario, una variable de cada nota en la partitura depende de la forma en que el usuario quiera controlarlo.

Los instrumentos de la orquesta de Csound son definidos con una sintaxis sencilla que invoca complejas rutinas de procesado de audio. La partitura que se pasa a la orquesta contiene información sobre la altura, la duración y otros elementos de control, codificados numéricamente en un formato estándar de partitura. Aunque muchos usuarios están satisfechos con dicho formato, a veces es conveniente usar lenguajes de alto nivel para procesar estas partituras.

Los programas en los que se basa Csound tienen una larga historia de desarrollo, empezando por Music 4, un programa escrito por Max Mathews a principios de los 60 en los laboratorios de la Bell Telephone. Este programa introdujo el concepto de almacenamiento en tablas, así como mucha de la terminología que desde entonces ha permitido a los investigadores en música por ordenador comunicarse entre ellos. Godfrey Winham, en Princeton, realizó valiosas aportaciones con su Music 4B; mi propio Music 360 (1968) le debe mucho a su trabajo. Con Music 11 (1973) tomé otra dirección: de mi intenso trabajo en años precedentes en el diseño del hardware de sintetizadores surgió la idea de separar el procesado de las señales de audio del de las de control. Esta división aún perdura en Csound. Gracias a que está escrito íntegramente en C, Csound puede ser instalado fácilmente en cualquier máquina basada en Unix o en C. En el MIT, Csound se ejecuta en estaciones VAX/DEC bajo Ultrix 4.2, en SUNs bajo OS 4.1, en SGIs bajo 5.0, en IBM PCs bajo DOS 6.2 y Windows 3.1 y en Macintosh bajo ThinkC 5.0.

Tomando este único lenguaje para definir el procesado de la señal digital y formatos de audio portables, como AIFF y WAV, los usuarios pueden cambiar de una máquina a otra fácilmente. La versión de 1991 añadió un codificador vocal de fase (*fase Vocoder*), FOF (del francés *Fonction d'onde formantique*, Funciones de Onda de Formantes) y los tipos de datos espectrales. En 1992 vieron la luz las unidades de conversión y de control MIDI, permitiendo a Csound recibir datos desde ficheros MIDI o desde un teclado externo. En 1994 se integraron los programas de análisis de sonido (**lpc**, **pvoc**) en el

módulo principal, permitiendo así que todo el procesado de Csound se hiciera desde un único fichero ejecutable. Además, Cscore podía ahora pasar directamente partituras a la orquesta para realizar ejecuciones iterativas. La versión de 1995 expandió las opciones MIDI e introdujo el generador *linseg* orientado a MIDI, filtros Butterworth, síntesis granular y, además, mejoró el rastreador de altura basado en los tipos espectrales. De especial importancia fue la introducción de herramientas para la generación de eventos en tiempo de ejecución (Cscore y MIDI), ya que permitían configurar mecanismos de percepción y respuesta en campos como los de la composición y la experimentación interactiva. Parecía por fin que el software de síntesis en tiempo real prometía algo verdaderamente importante.

1.1 Dónde Conseguir Csound y su Manual

La versión de dominio público de Csound y la edición hipertexto en inglés de este manual están disponibles en:

`ftp://ftp.maths.bath.ac.uk/pub/dream`

o

`ftp://ftp.musique.umontreal.ca/pub/mirrors/dream`

La edición impresa de este manual está disponible, junto con la edición inglesa original, en

<http://www.geocities.com/~csoundmanual>

1.2 Cómo usar el Manual de Csound

El manual de Csound está estructurado como un manual de referencia (no como un tutorial), ya que esta es la forma en la que el usuario lo encontrará, a la larga, más útil cuando diseñe instrumentos. Trabajar con Csound puede resultar una experiencia exigente al principio. Por tanto, es altamente recomendable examinar los tutoriales que se incluyen en el suplemento a este manual. Una vez que se aclaren los conceptos básicos, gracias al tutorial para principiantes, el lector puede aventurarse en el resto del texto localizando la información en las entradas del índice.

1.3 La Lista de Correo de Csound

Existe un grupo de correo donde se discute sobre Csound. Es mantenido por James Andrews de la Universidad de Exeter en Gran Bretaña.

Para incorporar tu nombre a la lista envía un mensaje a:

```
csound-request@maths.ex.ac.uk
```

con una única línea en el cuerpo del mensaje en la forma:

```
subscribe name@host
```

Los mensajes enviados a

```
csound@maths.ex.ac.uk
```

van a todos los miembros suscritos a la lista.

1.3.1 REPORTE DE ERRORES

Si se sospechara de un error en el código, por favor comuníquese a la lista.

2 SINTAXIS DE LA ORQUESTA

Una *sentencia* u orden en una orquesta tiene siempre el siguiente formato:

```
etiqueta: resultado opcode argumento1, argumento2, ... ;comentarios
```

La etiqueta es opcional e indica que la orden principal que le sigue puede ser el objetivo potencial de una orden goto (ver **Sentencias de Control de Programa**). Una etiqueta no tiene de por sí ningún efecto sobre la orden que le sigue.

Los comentarios son opcionales y tienen el sólo propósito de permitir al usuario documentar su orquesta. Los comentarios empiezan siempre por un punto y coma (;) y se extienden hasta el final de la línea.

El resto (resultado, opcode y argumentos) forman la orden principal. También son opcionales, es decir, una línea puede tener sólo una etiqueta, un comentario o estar en blanco. Si está presente, la orden principal debe estar completa en una única línea. El **opcode**, o código de operación, determina la tarea a realizar. Normalmente lleva algunos valores de entrada (llamados argumentos, de los que puede haber hasta 800 aproximadamente). Y, normalmente, también presenta una variable, en el campo del resultado, a la que enviar, a una determinada frecuencia, los valores de salida. Hay cuatro posibles frecuencias o tipos (NT: la palabra inglesa *rate* significa aquí velocidad o frecuencia, pero será traducida muchas veces como "tipo" a lo largo de todo el manual por razones de claridad y concisión). Estos 4 tipos se procesan:

- (1) una vez sólo, cuando la orquesta es configurada (es, en realidad, una asignación permanente)
- (2) una vez al principio de cada nota (es decir, en su inicialización: *tipo i-*)
- (3) una vez en cada bucle de control de la ejecución (frecuencia de control o *tipo k-*)
- (4) una vez cada muestra de sonido de cada bucle de control (frecuencia de audio o *tipo a-*)

2.1 Directorios y Ficheros

Muchos generadores y la orden Csound misma especifican nombres de fichero en los que escribir o de los que leer. Opcionalmente, pueden ser descripciones (rutas) completas, en los que el directorio en el que se encuentra el fichero se especifica con detalle en la forma:

Unidad:\directorio\subdirectorio\...\fichero

Si no se especifica el camino completo, los ficheros se buscarán en varios directorios por orden, dependiendo de qué tipo de archivo sea y de cómo estén configuradas ciertas variables de entorno. Esto último es opcional, pero puede servir para particionar y organizar eficientemente los directorios para que los ficheros fuente puedan ser compartidos en vez de duplicados en directorios distintos. Las variables de entorno pueden definir los directorios para los archivos de sonido (SFDIR), los archivos de muestras (SSDIR), los archivos de análisis de sonido (SADIR), y los ficheros para incluir en nuestras orquestas y partituras (INCDIR).

El orden de búsqueda es el siguiente:

1. - Los archivos de sonido que se generan son colocados en SFDIR (si existe), si no, en el directorio actual.
2. - Los archivos de sonido para lectura se buscan en el directorio actual, luego en SSDIR y después en SFDIR.
3. - Los archivos de análisis para lectura se buscan en el directorio actual y luego en SADIR.
4. - Los ficheros de código para incluir (con la orden `#include`) en las orquestas y partituras se buscan en el directorio actual, luego en el mismo directorio en que esté la orquesta o la partitura (según cual de las dos lo requiera) y por último en INCDIR.

Empezando con la versión 3.54 de Csound, el fichero "csound.txt" contiene los mensajes (en formato binario) que Csound usa para proporcionar información al usuario durante la ejecución. Esto permite que los mensajes estén en cualquier idioma, aunque el que se escoge por defecto es el inglés. El fichero debe ser colocado en el mismo directorio que el ejecutable de Csound. De otra manera, dicho fichero podría ser colocado en SFDIR, SSDIR o SADIR. Los usuarios de Unix pueden alojarlo también en "usr/local/lib/". La variable de entorno CSSTRNGS puede ser usada para definir el directorio en el que reside la base de datos. El contenido de dicha variable puede ser sustituido mediante la opción `-j` de la línea de comandos. (Nuevo en la versión 3.55)

2.2 Nomenclatura

A lo largo de todo este documento, los opcodes son indicados en **negrita** y los mnemónicos de sus argumentos y sus resultados, cuando son mencionados en el texto, se representan en *itálica*. Los nombres de los argumentos son generalmente mnemónicos en inglés (*arg* para *argument*, un argumento genérico; *amp* para *amplitude*, amplitud; *psh* para *phase*, fase, por ejemplo), mientras que el resultado se indica con la letra *r*. Ambos vienen precedidos de un calificativo *i*, *k*, *a* o *x* (por ejemplo *kamp*, *iphs*, *ar*). El prefijo *i*- indica un escalar válido en el momento de la inicialización; los prefijos *k*-o *a*- indican respectivamente un valor de control (un escalar) o uno de audio (un vector, es decir, un array de valores), modificado y referenciado continuamente a lo largo de la ejecución (es decir, en cada período de control mientras el instrumento esté activo).

Los argumentos se usan en las frecuencias prefijadas por los tipos. Los resultados son calculados de la misma manera y quedan disponibles para su uso posterior como argumentos de entrada en cualquier parte. Con muy pocas excepciones, la frecuencia de los argumentos de entrada no puede exceder la del resultado. La validez de estos argumentos viene definida de la siguiente manera:

- 1- argumentos con el prefijo *i*- serán válidos en el instante de inicialización de una nota.
- 2- argumentos con el prefijo *k*- serán accesibles tanto como valores de control como de inicialización.
- 3- argumentos con el prefijo *a*- deben ser vectores (arrays de elementos).
- 4- argumentos con el prefijo *x*- pueden ser vectores o escalares (el compilador distinguirá).

Todos los argumentos, a no ser que se especifique de otra manera, pueden ser expresiones cuyos resultados sigan las especificaciones anteriores. La mayoría de los opcodes (como por ejemplo **linen** y **oscil**) pueden ser usados en más de una forma, determinada por el prefijo del símbolo del resultado.

En la orquesta de Csound, las sentencias pueden dividirse en 12 categorías, consistiendo cada una de ellas de 65 subcategorías. Cada una forma un capítulo separado de este Manual. Las categorías (y el número correspondiente de su capítulo) son los siguientes:

Sintaxis de la Orquesta

- 3: Sentencias de Cabecera de la Orquesta
- 4: Sentencias de Bloques de Instrumentos
- 5: Inicialización de Variables

Control de los Instrumentos

- 6: Llamadas a Instrumentos
- 7: Control de la Duración
- 8: Control de la Ejecución en Tiempo Real
- 9: Lectura de Tiempo
- 10: Control del Reloj
- 11: Detección y Control
- 12: Valores Condicionales
- 13: Macros
- 14: Control del Flujo del Programa
- 15: Reinicialización

Operaciones Matemáticas

- 16: Operaciones Aritméticas y Lógicas
- 17: Funciones Matemáticas
- 18: Funciones Trigonométricas
- 19: Funciones de Amplitud
- 20: Funciones Aleatorias
- 21: Opcodes Equivalentes a Funciones

Convertidores de Altura

- 22: Funciones
- 23: Opcodes de Afinación

Soporte MIDI

- 24: Convertidores
- 25: Entrada de Controladores
- 26: Bancos de Mandos Deslizantes
- 27: E/S Genérica
- 28: Activación/Desactivación de Nota
- 29: Salida de Mensajes MIDI
- 30: Mensajes en Tiempo Real
- 31: Extendedores de Eventos MIDI

Generadores de Señal

- 32: Generadores Lineales y Exponenciales
- 33: Acceso a Tablas
- 34: Generadores de Fase
- 35: Osciladores Básicos
- 36: Osciladores de Espectro Dinámico
- 37: Síntesis y Resíntesis Aditiva
- 38: Síntesis FM
- 39: Reproducción de Muestras
- 40: Síntesis Granular
- 41: Modelado Físico por Guía de Ondas
- 42: Modelos y Emulaciones
- 43: Resíntesis STFT (Vocoding)
- 44: Resíntesis LPC
- 45: Generadores Aleatorios (Ruido)

Control de Tablas de Función

- 46: Consultas a Tablas
- 47: Selección de Tablas
- 48: Operaciones de Lectura/Escritura

Modificadores de Señal

- 49: Filtros Estándar
- 50: Filtros Especializados
- 51: Modificadores de Envoltorio
- 52 Modificadores de Amplitud
- 53: Limitadores de Señal
- 54: Líneas de Retardo
- 55: Reverberación
- 56: Guías de Onda
- 57: Efectos Especiales
- 58: Convolución y Transformación (Morphing)
- 59: Panoramización y Espacialización
- 60: Operadores a Nivel de Muestra

El Sistema de Cableado Zak

- 61: El Sistema de Cableado Zak

Operaciones que Usan Tipos Espectrales de Datos

- 62: Operaciones que Usan Tipos Espectrales de Datos

Entrada y Salida de Señales

63: Entrada

64: Salida

65: E/S de Ficheros

66: Consultas a Ficheros de Sonido

67: Impresión y Representación en Pantalla

2.3 Tipos de Sentencias en la Orquesta

Una orquesta en **Csound** se compone de una serie de *sentencias de cabecera* que configuran varios parámetros globales, seguidas por un número de *bloques de instrumentos* que representan diferentes tipos de instrumentos. Un bloque de instrumento, a su vez, se compone de *sentencias ordinarias* que definen valores, controlan el flujo lógico de la señal, o llaman a los diferentes tipos de subrutinas de procesado de señal que calculan la salida de audio.

Las *sentencias de cabecera de la orquesta* operan sólo una vez, precisamente en el momento de la configuración de la orquesta. Normalmente, constan de la asignación de un valor determinado a ciertos *símbolos globales reservados*, por ejemplo `sr=20000`. Todas las sentencias de cabecera pertenecen a un pseudo instrumento 0, que se inicializa en el instante 0 de la partitura y antes que ningún otro instrumento. Una *orden ordinaria* también puede formar parte de la cabecera, por ejemplo `gifreq=cpspch(8.09)`, considerando que es una operación sólo de inicialización.

Una *orden ordinaria* se ejecuta durante la inicialización, durante la ejecución o en ambas. Las operaciones que producen un resultado se ejecutan a la frecuencia especificada por dicho resultado (es decir, en el instante de la inicialización de una nota para los resultados del *tipo i-* y durante la ejecución para los resultados de *tipo k-* y *tipo a-*), con la única excepción del opcode **init**. La mayoría de los **generadores** y **modificadores**, sin embargo, producen señales que dependen no sólo del valor instantáneo de sus argumentos sino también de su estado interno reservado. Por tanto, estas unidades que operan en tiempo de ejecución tienen un componente de inicialización implícito que configura dicho estado. El instante de ejecución de una operación que no produce resultado alguno queda implícitamente manifiesto en el opcode mismo.

Los *argumentos* son los valores que se envían a una determinada operación. La mayoría de ellos aceptan expresiones compuestas de **constantes**, **variables**, **símbolos reservados**, **convertidores de valor**, **operaciones aritméticas** y **valores condicionales**.

2.4 Constantes y Variables

Las **constantes** son números en coma flotante, como por ejemplo 1, 3.14159 ó -73.45. Se puede disponer de ellas en cualquier momento y nunca cambian de valor.

Las **variables** son elementos que contienen valores. Se puede disponer de ellas en cualquier momento y pueden ser actualizadas a cualquiera de las cuatro frecuencias de actualización disponibles (en la configuración de la orquesta, en la inicialización de cada nota (*tipo i-*), en el bucle de control (*tipo k-*) o en cada muestra del audio digital producido (*tipo a-*)). Las variables de tipo i- o k- son escalares (es decir, sólo pueden tomar un valor en un instante determinado) y se usan principalmente para almacenar y acceder a datos de control, esto es, datos que cambian bien en la inicialización de cada nota (en el caso de las variables de tipo i-), bien en el bucle de control de la ejecución (en el caso de las variables de tipo k-). Las variables de tipo i- o k- son útiles, por tanto, para almacenar los parámetros de las notas, tales como alturas, duraciones, movimientos lentos de frecuencia, vibratos, etc. Las variables de tipo a-, en cambio, son *arrays*, es decir, vectores de información (esto es, una tabla de valores). Aunque se actualizan en la misma pasada del bucle de control de la ejecución en la que lo hacen las variables de tipo k-, estos vectores representan una resolución temporal mayor, al dividir el período de control en subperíodos de muestra (ver **ksmps**). Las variables de tipo a- se usan para almacenar y acceder a los datos que cambian a la frecuencia de muestreo de la señal de audio (por ejemplo las señales de salida de los osciladores, los filtros, etc.)

Otra distinción es la que se hace entre variables locales y globales. Las **variables locales** son privadas para un instrumento en particular y no pueden ser ni modificadas ni leídas desde ningún otro instrumento. Sus valores se preservan y pueden llevar información de pasada en pasada (por ejemplo, de la inicialización al bucle de control de ejecución) dentro de un mismo instrumento. Las variables locales empiezan con las letras **p**, **i**, **k** o **a**, según su tipo. Puede aparecer un mismo nombre de variable local en dos bloques de instrumento distintos sin conflicto alguno.

Las variables **globales** son accesibles por todos los instrumentos de la orquesta. Los nombres son los mismos que reciben las locales pero precedidos de la letra **g**, o pueden también ser símbolos reservados especiales. Las variables globales se usan para pasar valores generales, para que puedan comunicarse entre sí los distintos instrumentos, o para mandar sonido de un instrumento a otro (por ejemplo, en la mezcla previa a la reverberación).

Comprendidas estas distinciones, he aquí las 8 formas de variables locales y globales:

tipo	actualización	locales	globales
símbolos reservados	al principio (permanentes)	-----	símbolo r
parámetros de la partitura	tipo i-	pnúmero	-----
símbolos v	tipo i-	vnúmero	gvnúmero
variables tipo i-	tipo i-	inombre	ginombre
controladores midi	cualquier momento	cnúmero	-----
señales de control	tipo k-	knombre	gknombre
señales de audio	tipo a-	anombre	ganombre
tipos de datos espectrales	tipo k-	wnombre	-----

donde "símbolo r" es un *símbolo reservado* especial (como por ejemplo **sr**, **kr**), "número" es un entero positivo referido a un **campo p** (ver más adelante) o a una secuencia de números, y "nombre" es una cadena de caracteres y/o dígitos con un significado local o global. Como es lógico, los parámetros de

la partitura son variables locales de tipo `i-`, cuyos valores son copiados desde la orden de la partitura que los invoca, justo antes de la pasada de inicialización de un instrumento, mientras que los controladores MIDI son variables que pueden ser actualizadas asíncronamente desde un fichero MIDI o cualquier dispositivo MIDI externo.

2.5 Expresiones

Las expresiones pueden ser compuestas y tener cualquier grado de complejidad. Cada parte de una expresión se evalúa de acuerdo a su propia frecuencia. Por ejemplo, si los términos dentro de una subexpresión cambian todos a la frecuencia de control o más lentamente, dicha subexpresión será evaluada sólo a frecuencia de control; el resultado puede ser usado entonces en un cálculo a frecuencia de audio. Por ejemplo, en la expresión:

```
k1 + abs(int(p5) + frac(p5) * 100/12 + sqrt(k1))
```

la subexpresión 100/12 será evaluada en la inicialización de la orquesta; p5 será evaluado en la inicialización de la nota y el resto de la expresión será evaluado cada bucle de control k-. Dicha expresión puede usarse como argumento de un generador elemental, o como parte de una **sentencia de asignación** por ejemplo.

3 SINTAXIS DE LA ORQUESTA: SENTENCIAS DE CABECERA DE LA ORQUESTA

3.1 sr, kr, ksmpls, nchnls

```
sr          =      iarg
kr          =      iarg
ksmpls     =      iarg
nchnls     =      iarg
```

3.1.1 DESCRIPCIÓN

Estas sentencias son asignaciones de valores globales, realizadas al principio de la orquesta, antes de que sean definidos los bloques de instrumento. Su función es configurar ciertas *variables de símbolos reservados* que son requeridas para la ejecución. Una vez configurados, estos símbolos reservados pueden ser usados como expresiones en cualquier lugar de la orquesta.

sr = (opcional) - establece la frecuencia de muestreo en *n1* muestras por segundo y por canal. El valor por defecto es 10000.

kr = (opcional) - establece la frecuencia de control en *n2* muestras por segundo. El valor por defecto es 1000.

ksmpls = (opcional) - establece en *n3* el número de muestras en cada período de control. **Este valor debe ser igual a sr/kr.** El valor por defecto es 10.

nchnls = (opcional) - establece en *n4* el número de canales de la salida de audio. (1 = mono, 2 = estéreo, 4 = cuadrafónico). El valor por defecto es 1 (mono)

Además, cualquier **variable global** puede ser inicializada por una *asignación de inicialización* en cualquier parte antes de la primera orden **instr**.

Todas las asignaciones anteriores son ejecutadas como instrumento 0, al principio de la ejecución real.

3.1.2 EJEMPLO

```
sr          =      10000
kr          =      500
ksmpls     =      20
gil        =      sr/2.
ga         init  0
itranspose =      octpch(.01)
           strset 10, "asound.wav"
```

La última orden de este ejemplo permitirá que el valor 10 sustituya al fichero *asound.wav* en cualquier parte de la orquesta donde éste sea llamado o referenciado.

3.2 strset, pset

```
strset      iarg, "cadena de caracteres"  
pset       con1, con2, con3,...
```

3.2.1 DESCRIPCIÓN

Permite que ciertos parámetros globales sean inicializados en el instante de carga de la orquesta, en lugar de en la inicialización del instrumento o en tiempo de ejecución.

3.2.2 INICIALIZACIÓN

iarg – valor numérico que se asocia a una cadena alfanumérica

con1, *con2*, etc. – valores preestablecidos para un instrumento MIDI

strset permite que una cadena, como por ejemplo el nombre de un fichero, sea asociada a un valor numérico. Su uso es opcional.

pset - esta unidad define e inicializa arrays numericos en el instante de carga de la orquesta. Pueden ser usados en la cabecera de la orquesta (es decir, como instrumento 0) o dentro de un instrumento. En este último caso, no forma parte de la inicialización o ejecución de éste y sólo se permite una sentencia por instrumento. Estos valores están disponibles como tipos i- por defecto. Cuando un instrumento es disparado desde MIDI sólo toma los campos p1 y p2 del evento MIDI, y p3, p4, etc. recibirán los valores reales preseleccionados.

3.2.3 EJEMPLO

El ejemplo siguiente ilustra el uso de **pset** dentro de un instrumento.

```
kcps = i2/3 + cpsoct(k2 + octpch(p5))  
instr 1  
  pset 0,0,3,4,5,6 ; pfield substitutes  
a1    oscil 10000, 440, p6
```

3.3 seed

`seed ival`

3.3.1 DESCRIPCIÓN

seed proporciona el valor germinal del generador de números pseudo-aleatorios que usan todos los generadores de ruido **rand**, **randi**, **randh**, y los de **clase x**.

rnd(x) y **birnd(x)** no se ven afectados por **seed**.

3.3.2 INICIALIZACIÓN

ival – valor germinal que será usado por el generador aleatorio

3.3.3 EJECUCIÓN

El uso de **seed** proporcionará resultados predecibles desde una orquesta usando generadores aleatorios, cuando sea necesario en ejecuciones múltiples.

3.4 ftgen

```
gir ftgen ifn, itime, isize, igen, iarga[, iargb...iargz]
```

3.4.1 DESCRIPCIÓN

Genera una tabla de función en la partitura desde dentro de la orquesta.

3.4.2 INICIALIZACIÓN

gir - un número de tabla mayor que 100, proporcionado desde fuera o asignado automáticamente. Si se usan dentro de un instrumento puede ser una variable local *ir*.

ifn - el número de la tabla solicitada. Si *ifno* es 0, el número se asigna automáticamente y el valor se coloca en *iafno*. Cualquier otro valor se tomará como número de la tabla.

itime - o se ignora o corresponde al campo p2 de una **sentencia f** de la partitura.

isize - tamaño de la tabla. Corresponde al campo p3 de una **sentencia f** de la partitura.

igen - rutina GEN de la tabla de función. Corresponde al campo p4 de una **sentencia f** de la partitura.

iarga-iargz - argumentos de la tabla de función. Corresponde a los campos de p5 en adelante de una **sentencia f** de la partitura.

3.4.3 EJECUCIÓN

Es equivalente a la generación de una tabla en la partitura con la **sentencia f**.

3.4.4 AUTOR

Barry Vercoe
M.I.T., Cambridge, Mass
1997

3.5 massign, ctrlinit

```
massign    ichnl, insnum
ctrlinit   ichnl, ictrlno1, ival1[, ictrlno2, ival2 [, ictrlno3,
            ival3[,..ival32]]
```

3.5.1 DESCRIPCIÓN

Inicia los controladores MIDI para una orquesta de CSound.

3.5.2 INICIALIZACIÓN

ichnl – número de canal MIDI.

insnum – número del instrumento de Csound en la orquesta.

ictrlno1, *ictrlno2*, etc. – números de los controladores MIDI.

ival1, *ival2*, etc. – valor inicial del controlador MIDI correspondiente.

3.5.3 EJECUCIÓN

massign asigna un número de canal MIDI a un instrumento de Csound.

ctrlinit - configura los valores iniciales para una serie de controladores MIDI.

3.5.4 AUTOR

Barry Vercoe
M.I.T., Cambridge, Mass
1997

4 SINTAXIS DE LA ORQUESTA: SENTENCIAS DE BLOQUES DE INSTRUMENTOS

4.1 instr, endin

```
instr i, j, ...  
.   
. < cuerpo  
. del  
. instrumento  
.   
endin
```

4.1.1 DESCRIPCIÓN

Estas sentencias delimitan el cuerpo de un instrumento. Siempre deben aparecer en parejas.

instr - empieza el bloque de definición de los instrumentos *i, j, ...*

i, j, ... deben ser números enteros, nunca expresiones. Son legales todos los enteros positivos, en cualquier orden, pero es conveniente evitar números muy altos.

endin - termina el bloque de instrumento actual.

Nota:

Puede haber cualquier número de bloques de instrumento en una orquesta.

Los instrumentos pueden ser definidos en cualquier orden (pero serán siempre inicializados y ejecutados en orden ascendente según su número).

Los bloques de instrumento no pueden ser anidarse (es decir, un bloque no puede contener a otro).

5 SINTAXIS DE LA ORQUESTA: INICIALIZACIÓN DE VARIABLES

5.1 =, init, tival, divz

```
ir    =    iarg
kr    =    karg
ar    =    xarg
kr    init iarg
ar    init iarg
ir    tival
ir    divz ia, ib, isubst (aún no implementados)
kr    divz ka, kb, ksubst
ar    divz xa, xb, ksubst
      pset con1, con2, con3,...
```

5.1.1 DESCRIPCIÓN

= (asignación simple) asigna al resultado el valor de la expresión *iarg* (*karg*, *xarg*). Esto proporciona un medio de almacenar en una variable o constante un determinado resultado para su uso posterior.

init - inicializa el resultado, es decir asigna a una variable k- o a- el resultado de la expresión *iarg*. Observa que **init** proporciona el único medio de que una orden de inicialización almacene su resultado en una variable de ejecución (tipo k- o a-); la orden no tiene ningún efecto en tiempo de ejecución.

tival - asigna a la variable i- el valor del estado del indicador interno de "nota ligada" del instrumento. Por convenio, se toma el valor 1 si la nota ha sido ligada a la nota precedente (ver **Sentencia i**); si no hay ligadura se asigna el valor 0 (ver también **tigoto**)

divz - asigna al resultado el valor de a/b , siempre que "b" sea distinta a 0; cuando "b" es 0, asigna el valor de *subst*.

6 CONTROL DE INSTRUMENTOS: LLAMADAS A INSTRUMENTOS

6.1 schedule, schedwhen

```
schedule    inst, iwhen, idur [, p4, p5,...]  
schedwhen  ktrigger, kinst, kwhen, kdur [, p4, p5,...]
```

6.1.1 DESCRIPCIÓN

Añade un nuevo evento a la partitura.

6.1.2 INICIALIZACIÓN

inst – número de instrumento del nuevo evento.

iwhen – tiempo en el que ocurrirá el nuevo evento.

idur – duración del nuevo evento.

6.1.3 EJECUCIÓN

ktrigger – valor de disparo del nuevo evento.

schedule añade un nuevo evento a la partitura. Los argumentos, incluidas las opciones, son las mismas que en la partitura. El instante *iwhen* (p2) es el momento en que ocurren los eventos.

Si la duración es 0 o negativa el nuevo evento es de tipo MIDI y hereda el sub-evento de desactivación de la instrucción **schedule**.

En el caso de **schedwhen**, el evento es programado sólo cuando el valor -k de *ktrigger* es inicialmente distinto de 0.

6.1.4 EJEMPLO

```
;; repite las notas y las separa 1 seg

instr 1
  schedule 2, 1, 0.5, p4, p5
a1  shaker p4, 60, 0.999, 0, 100, 0
    out    a1
endin

instr 2
a1  marimba p4, cpspch(p5), p6, p7, 2, 6.0, 0.05, 1, 0.1
    out    a1
endin

instr 3
kr  table   kr, 1
    schedwhen kr, 1, 0.25, 1, p4, p5
endin
```

6.1.5 AUTOR

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Noviembre, 1998 (Nuevo en la versión 3.491)
Basado en el trabajo de Gabriel Maldonado.

6.2 schedkwhen

schedkwhen *ktrigger*, *kmintim*, *kmaxnum*, *kinsnum*, *kwhen*, *kdur*[, *kp4*, *kp5*, ...]

6.2.1 DESCRIPCIÓN

Añade un nuevo evento de partitura generado por un disparador a frecuencia de control.

6.2.2 EJECUCIÓN

ktrigger – dispara un nuevo evento de partitura. Si *trigger* = 0, no se generará ningún nuevo evento.

kmintim – tiempo mínimo, en segundos, entre eventos generados. Si *kmintim* <= 0, no existe límite de tiempo. Si *kinsum* es negativo (para desactivar un instrumento), este test es pasado de largo.

kmaxnum – máximo número de apariciones simultáneas permitidas del *kinsum* de un instrumento. Si el número de apariciones de *kinsum* es >= *kmaxnum*, no se producirá ningún nuevo evento. Si *kmaxnum* es <= 0, éste no será usado como límite en la generación de eventos. Si *kinsum* es negativo (para desactivar un instrumento), este test será omitido.

kinsnum – número de instrumento. Equivalente a p1

kwhen – instante de comienzo del nuevo evento. Equivalente a p2 en una **sentencia i** de la partitura. Se mide desde el instante del evento disparador. *kwhen* debe ser >= 0. Si *kwhen* > 0, el instrumento no será inicializado hasta el instante preciso en que deba ser ejecutado.

kdur – duración del evento. Equivalente a p3 en una **sentencia i** de la partitura. Si *kdur* = 0, el instrumento sólo hará una pasada de inicialización, sin su posterior ejecución. Si *kdur* es negativo, se inicia una nota tenida (ver **ihold** y **sentencia i**).

kp4, *kp5*, etc. – equivalente a p4, p5, etc., en una **sentencia i** de la partitura.

Nota: Mientras se espera que los eventos sean disparados por **schedkwhen**, la ejecución debe seguir adelante, ya que Csound podría terminar si no esperara más eventos de partitura. Para garantizar una ejecución continua puede usarse una **sentencia f0** en la partitura.

6.2.3 AUTOR

Rasmus Ekman
EMS
Stockholm, Sweden
Nuevo en la Versión 3.59

6.3 turnon

`turnon insnum[,itime]`

6.3.1 DESCRIPCIÓN

Activa un instrumento por un tiempo indefinido.

6.3.2 INICIALIZACIÓN

insnum – número de instrumento a ser activado.

itime –retardo, en segundos, después del cual el instrumento *insnum* será activado. El valor por defecto es 0.

6.3.3 EJECUCIÓN

turnon activa el instrumento *isnum* después de un retardo de *itime* segundos, o inmediatamente si *itime* no viene especificado. El Instrumento estará activo hasta que sea explícitamente desactivado (ver **turnoff**).

7 CONTROL DE INSTRUMENTOS: SENTENCIAS DE CONTROL DE LA DURACIÓN

7.1 ihold, turnoff

ihold
turnoff

7.1.1 DESCRIPCIÓN

Estas sentencias permiten a la nota actual modificar su propia duración:

ihold - esta sentencia de tipo i- hace que una nota de duración finita se convierta en una nota tenida. Si sucede así, tiene el mismo efecto que un p3 negativo (ver **Sentencia i**), excepto por el hecho de que, de esta manera, p3 sigue siendo positivo y la nota pasa a ser mantenida indefinidamente. Esta puede ser más tarde desactivada explícitamente con **turnoff**, o con una nota posterior del mismo instrumento que la pise (es decir, que la nota mantenida quede ligada con esta nueva nota). Sólo es efectiva en la inicialización, no en una pasada de reinicialización de **reinit**.

turnoff - esta sentencia permite a un instrumento autodesactivarse en tiempo de ejecución. Tanto si es de duración finita o tenida, la nota que está siendo ejecutada en ese momento por el instrumento es inmediatamente eliminada de la lista de notas activas. Ninguna otra nota se verá afectada.

7.1.2 EJEMPLO

En el siguiente ejemplo se fuerza la conclusión de una nota cuando una señal de control sobrepasa un cierto umbral (en este caso la frecuencia Nyquist)

```
k1    expon      440, p3/10,880    ; empieza gliss y continúa
      if k1 < sr/2 kgoto contin    ; hasta que se detecte el umbral Nyquist
      turnoff                                ; entonces finaliza
contin:
a1    oscil      a1, k1, 1
```

8 CONTROL DE INSTRUMENTOS: CONTROL DE LA EJECUCIÓN EN TIEMPO REAL

8.1 active

`ir active inst`

8.1.1 DESCRIPCIÓN

Devuelve el número de activaciones de un instrumento.

8.1.2 INICIALIZACIÓN

inst – número del instrumento a ser interrogado.

8.1.3 EJECUCIÓN

active devuelve en el instante en que es invocado el número de activaciones de un instrumento *inst*.

8.1.4 AUTOR

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Julio, 1999
Nuevo en la versión 3.57

8.2 cpuprc, maxalloc, prealloc

cpuprc	<i>instrnum</i> , <i>ipercent</i>
maxalloc	<i>instrnum</i> , <i>icount</i>
prealloc	<i>instrnum</i> , <i>icount</i>

8.2.1 DESCRIPCIÓN

Controla la asignación de los recursos de la CPU en base a los instrumentos para optimizar la salida en tiempo real.

8.2.2 INICIALIZACIÓN

instrnum - número de instrumento

ipercent - porcentaje del tiempo de procesado asignado a la CPU . Puede ser expresado también como un valor fraccionario.

icount – número de asignaciones de instrumento.

8.2.3 EJECUCIÓN

cpuprc define el porcentaje de tiempo de procesado de la cpu para un instrumento, con tal de evitar el desborde del buffer en ejecuciones en tiempo real, habilitando algo así como un umbral de polifonía. El usuario debe ajustar el valor *ipercent* para cada instrumento que será activado en tiempo real. Asumiendo que el tiempo total de procesado teórico de la CPU es del 100%, dicho porcentaje sólo podrá definirse empíricamente, porque son demasiados los factores que contribuyen a limitar la polifonía en tiempo real en distintos ordenadores.

Por ejemplo, si *ipercent* es puesto al 5% para el instrumento 1, el máximo número de voces que le podrán ser asignadas en tiempo real será de 20 ($5\% * 20 = 100\%$). Si el usuario intenta ejecutar una nota más mientras que las 20 anteriores están aún sonando, Csound prohibirá la asignación de esa nota y mostrará un mensaje como este:

```
can't allocate last note because it exceeds 100% of cpu time
```

Con tal de evitar desbordes en el buffer de audio, se sugiere ajustar el máximo número de voces un poco por debajo de lo que permite la potencia real del ordenador. A veces un instrumento puede requerir más tiempo de procesado de lo normal. Si, por ejemplo, el instrumento contiene un oscilador que lee una tabla que no cabe en la memoria caché, será un poco más lento de lo normal. Además, cualquier programa que corra al mismo tiempo en multitarea, puede restar potencia de procesado en algun grado.

Al principio, se les da a todos los instrumentos un valor por defecto *ipercent* = 0.0% (es decir, infinita velocidad de procesado de la CPU). Este valor es adecuado para sesiones diferidas. **maxalloc** limita el número de asignaciones de un intrsumento. **prealloc** crea espacio para los instrumentos pero no los ejecuta.

Todas las apariciones de **cpuprc**, **maxalloc** y **prealloc** deben ser definidas en la sección de cabecera, no el cuerpo del instrumento.

8.2.4 EJEMPLO

```
sr = 44100
kr = 441
ksmps = 100
nchnls =2
cpuprc 1, 2.5      ; da al instr 1 un 2.5% de tiempo de procesado,
                   ; es decir, un máximo de 40 voces (2.5% * 40 = 100%)
cpuprc 2, 33.333  ; da al instr 2 un 33.333% de tiempo de procesado,
                   ; es decir, un máximo de 3 voces (33.333%*3 = 100%)
```

```
instr 1
...body...
endin
```

```
instr 2
....body...
endin
```

8.2.5 AUTOR

Gabriel Maldonado
Italia
Julio, 1999
Nuevo en la versión 3.57

9 CONTROL DE INSTRUMENTOS: LECTURA DE TIEMPO

9.1 timek, times, instimek, instimes

```
ir    timek
kr    timek
ir    times
kr    times
kr    instimek
kr    instimes
```

9.1.1 DESCRIPCIÓN

Opcodes que leen tiempo absoluto desde el principio de la ejecución o desde el principio de un instrumento (usando dos formatos distintos).

9.1.2 EJECUCIÓN

timek se usa para tiempos expresados en ciclos de control. Así con:

```
sr    =    44100
kr    =    6300
ksmps =    7
```

después de un segundo, **timek** devolvería 3150. El valor devuelto será siempre un número entero.

times proporciona tiempo en segundos. Devolverá 0.5 después de transcurrido medio segundo.

times y **timek** son opcodes similares que operan sólo al principio de la activación de un instrumento. Ambos producen una variable de inicialización (-i) en la salida, cuyo nombre empieza por "i" o "gi",.

timek y **times** producen ambos una variable de control como salida. No hay parámetros de entrada.

instimek y **instimes** son similares a **timek** y **times**, excepto en que devuelven el tiempo transcurrido desde el principio de la activación del instrumento.

9.1.3 AUTOR

Robin Whittle
Australia
Mayo 1997

10 CONTROL DE INSTRUMENTOS: CONTROL DEL RELOJ

10.1 clockon, clockoff, readclock

	clockon	inum
	clockoff	inum
<i>ir</i>	readclock	inum

10.1.1 DESCRIPCIÓN

Pone en marcha (**clockon**) o para (**clockoff**) uno de varios relojes internos y lee (**readclock**) el valor de un reloj.

10.1.2 INICIALIZACIÓN

inum - número de reloj. Hay 32 relojes numerados del 0 al 31. Todos los demás valores serán tomados como el reloj 32.

ir – valor, en la inicialización, del reloj especificado por *inum*.

10.1.3 EJECUCIÓN

Entre los opcodes **clockon** y **clockoff** se usa el reloj de la CPU para acumular en el reloj el tiempo transcurrido. La precisión depende de la máquina, pero es del orden de los milisegundos en sistemas basados en UNIX o Windows. **readclock** lee el valor actual de un reloj en el instante de la inicialización.

Nota: no hay manera de poner a 0 un reloj.

10.1.4 AUTOR

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Julio, 1999
Nuevo en la versión 3.56

11 CONTROL DE INSTRUMENTOS: DETECCIÓN Y CONTROL

11.1 pitch

```
koct, kamp pitch      asig, iupdte, ilo, ihi, idbthresh[, ifrqs, iconf, \\  
istrt, iocts, iq, inptls, irolloff, iskip]
```

11.1.1 DESCRIPCIÓN

Usando la misma técnica que **spectrum** y **specptrk**, **pitch** rastrea la altura de una señal, en formato "octava-punto-nota", y la amplitud en dB.

11.1.2 INICIALIZACIÓN

iupdte –longitud del período, en segundos, en los que se actualiza la salida.

ilo, ihi –rango en el que se detecta la altura, expresado en forma "octava-punto-nota".

idbthresh –amplitud, expresada en dB, necesaria para que una altura dada sea detectada. Una vez detectada, el proceso continua hasta que su amplitud baja 6 dB.

ifrqs –número de divisiones de una octava. El valor por defecto es 12 y está limitado a 120.

iconf –número de conformaciones necesarias para un salto de octava. El valor por defecto es 10.

istrt –altura inicial del rastreador. El valor por defecto es $(ilo + ihi)/2$.

iocts –número de divisiones de octava en el espectro. El valor por defecto es 6.

iq –Q de los filtros de análisis. El valor por defecto es 10.

inptls – número de armónicos, usados en el emparejamiento. El tiempo de ejecución crece con el número de armónicos. El valor por defecto es 4.

irolloff – caída de amplitud para la serie de filtros, expresada como una fracción por octava. El número de parciales y el coeficiente de caída pueden afectar al rastreo de altura, así que será necesario experimentar un poco. Los valores deben ser positivos. Se sugiere usar 4 o 5 armónicos, con un coeficiente de caída de 0.6, ó 10 ó 12 armónicos con una caída de 0.75 para timbres complejos con una fundamental débil. El valor por defecto es 0.6.

iskip – si no es 0, se omite la inicialización.

11.1.3 EJECUCIÓN

pitch analiza la señal de entrada, *asig*, para proporcionar un par altura-amplitud de salida, para la frecuencia más fuerte de la señal. El valor se actualiza cada *iupdte* segundos.

1.11.4 AUTOR

John ffitch

Universidad de Bath, Codemist Ltd.

Bath, UK

Abril, 1999

Nuevo en la versión 3.54

11.2 pitchamdf

```
kcps,          pitchamdf  asig, imincps, imaxcps [, icps[, imedi[, idowns[,//  
krms          iexcps]]]]
```

11.2.1 DESCRIPCIÓN

Rastrea la frecuencia de una señal basada en el método AMDF (Función de Diferencia de Magnitudes Medias). La salida serán las alturas y amplitudes rastreadas. El método es bastante rápido y debería ser posible ejecutarlo en tiempo real. Esta técnica da normalmente sus mejores resultados usada con señales monofónicas.

11.2.2 INICIALIZACIÓN

imincps – frecuencia mínima estimada (en Hz) presente en la señal.

imaxcps – frecuencia máxima estimada (en Hz) presente en la señal.

icps – frecuencia inicial (en Hz) estimada de la señal. Si es 0, $icps = (imincps + imaxcps) / 2$. El valor por defecto es 0.

imedi – tamaño del filtro medio aplicado a la salida *kcps*. El tamaño del filtro será $imedi * 2 + 1$. Si es 0, no será aplicado el filtro. El valor por defecto es 1.

idowns – factor de subsampleo para *asig*. Debe ser un entero. Un factor de $idowns > 1$ dará lugar a una ejecución más rápida, pero también puede ocasionar una detección de frecuencia algo peor. El rango útil es de 1 a 4. El valor por defecto es 1.

iexcps – frecuencia a la que será ejecutado el análisis de la altura, expresada en Hz. Si es 0, *iexcps* toma el valor de *imincps*, lo que normalmente produce buenos resultados. De todas formas, la experimentación con otros valores puede conducir a mejores resultados. El valor por defecto es 0.

11.2.3 EJECUCIÓN

kcps – salida del rastreo de altura.

krms – salida del rastreo de amplitudes.

pitchamdf normalmente trabaja mejor con señales monofónicas, y es bastante fiable si se proporcionan valores iniciales apropiados. Ajustando *imincps* y *imaxcps* para definir un margen de rastreo de alturas tan estrecho como sea posible, se conseguirán mejores resultados en la detección y la ejecución.

Debido a que el procedimiento sólo puede empezar a detectar alturas después de un retraso inicial, ajustar *icps* a un valor cercano a la altura real de la señal evita la aparición de valores falsos al comienzo.

El filtro medio evita que *kcps* salte. Experimenta para determinar el valor óptimo de *imedi* para una señal dada.

De todas formas, los valores iniciales pueden normalmente dejarse a los valores por defecto. El filtrado pasa bajos de *asig* antes de pasarla al opcode **pitchamdf** puede mejorar la ejecución, especialmente con formas de onda complejas.

11.2.4 EJEMPLO

```
ginput      ftgen      1, 0, 0, -1, "input.wav", 0, 4, 0 ; señal de entrada
giwave     ftgen      2, 0, 1024, 10, 1, 1, 1, 1 ; onda sintética
instr 1
asig       loscil     1, 1, ginput, 1 ; consigue la señal
; con la frec. original
asig       tone      asig, 1000 ; filtro pasa bajos
kcps, krms pitchamdf asig, 150, 500, 200 ; extrae la altura
; y el envolvente
asigl      oscil     krms, kcps, iwave ; resintetiza
; con alguna otra onda
          out       asigl
endin
```

11.2.5 AUTOR

Peter Neubäcker
Munich, Alemania
Agosto, 1999
Nuevo en la Versión 3.59

11.3 tempest

```
ktemp      tempest      kin, iprd, imindur, imemdur, ihp, ithresh, ihtim, \  
ixfdbak, istartempo, ifn[, idisprd, itweek]
```

11.3.1 DESCRIPCIÓN

Estima el *tempo* según los patrones de pulso (partes fuertes) de una señal de control.

11.3.2 INICIALIZACIÓN

iprd - período entre análisis (en segundos). Normalmente 0.02 segundos más o menos.

imindur - duración mínima (en segundos), que sirve como unidad de tiempo. Normalmente .2 segundos más o menos.

imemdur - duración (en segundos) del buffer *kin* de memoria a corto plazo que será escaneado para buscar patrones periódicos. Normalmente 3 segundos más o menos.

ihp - punto de potencia media (en Hz) de un filtro pasa-bajos usado para suavizar la entrada *kin* antes de cualquier otro proceso. Esto tenderá a suprimir la parte inestable de la señal. Normalmente 2 Hz.

ithresh- umbral de sonoridad por el que se corta la señal *kin* filtrada, antes de que sea colocada en el buffer de memoria a corto plazo como datos relevantes de tiempo. Normalmente, es el nivel de ruido de los datos de entrada.

ihtim - tiempo medio (en segundos) de un filtro máscara interno que enmascara los datos *kin* nuevos en presencia de datos recientes más fuertes. Normalmente 0.005 segundos más o menos.

ixfdbak - proporción del valor anticipado de esta unidad a ser mezclado con el *kin* entrante antes de cualquier procesado. Normalmente 3 más o menos.

istartempo - tiempo inicial (en pulsos por minuto). Normalmente 60.

ifn - número de la tabla de función almacenada (dibujada de izquierda a derecha) por la cual se atenuarán los datos de la memoria a corto plazo a lo largo del tiempo.

idisprd (opcional) - si no es 0, muestra los buffers a corto plazo posterior y siguiente cada *idisprd* segundos (normalmente un múltiplo de *iprd*). El valor por defecto es 0 (no muestra nada).

itweek (opcional) - ajuste de la unidad para que sea estable cuando analice eventos controlados por su propia salida. El valor por defecto es 1 (sin cambios).

11.4 follow

```
ar follow asig, idt
```

11.4.1 DESCRIPCIÓN

Este opcode es un "extractor" del envolvente de una señal de audio.

11.4.2 INICIALIZACIÓN

idt - período, en segundos, en el que se proporciona la amplitud media de la señal *asig*. Si la frecuencia de *asig* es baja, *idt* debe ser entonces un valor grande (más de la mitad del período de *asig*)

11.4.3 EJECUCIÓN

asig - señal de la cual se extrae el envolvente.

11.4.4 EJEMPLO

```
k1 line 0, p3, 30000 ; k1 es un envolvente simple
a1 oscil k1, 1000, 1 ; genera una señal simple usando k1
ak1 follow a1, .02 ; ak1 es ahora como k1
a2 oscil ak1, 1000, 1 ; genera una señal simple con ak1
out a2 ; ambas a1 y a2 son iguales.
```

Para evitar los ruiditos producidos por las discontinuidades del rastreo de envolventes complejos, se puede usar un filtro pasa-bajos que suavice el envolvente calculado.

11.4.5 AUTOR

Paris Smaragdis
MIT, Cambridge
1995

11.5 trigger

kout **trigger** ksig, kthreshold, kmode

11.5.1 DESCRIPCIÓN

Informa cuando una señal k- ha cruzado un umbral determinado.

11.5.2 EJECUCIÓN

ksig - señal de entrada.

kthreshold - umbral de **trigger**.

kmode - puede ser 0 , 1 ó 2.

Normalmente **trigger** devuelve ceros; sólo cada vez que la señal *ksig* cruza el umbral *kthreshold*, devuelve el valor 1.

Hay 3 modos de usar **trigger**:

- *kmode* = 0 - (abajo-arriba) **trigger** devuelve un 1 cuando el valor actual de *ksig* es mayor que el umbral *kthreshold*, mientras que el antiguo valor de *ksig* era igual o menor que dicho umbral.
- *kmode* = 1 - (arriba-abajo) **trigger** devuelve un 1 cuando el valor actual de *ksig* es menor que el umbral *kthreshold*, mientras que el antiguo valor de *ksig* era igual o mayor que dicho umbral.
- *kmode* = 2 - (ambos) **trigger** devuelve un 1 en cualquiera de los casos anteriores.

11.5.3 AUTOR

Gabriel Maldonado

Italia

Nuevo en la versión 3.49

11.6 peak

<i>kr</i>	peak	<i>ksig</i>
<i>kr</i>	peak	<i>asig</i>

11.6.1 DESCRIPCIÓN

Estos opcodes mantienen la variable de control de salida al nivel de pico absoluto recibido hasta ese momento.

11.6.2 EJECUCIÓN

kr – salida igual al valor absoluto mayor recibido hasta ese instante. Es, de hecho, también una entrada del opcode, ya que se lee *kr* para comprobar si se recibe algún valor mayor que él y, en dicho caso, hacer que *kr* tome el valor mayor.

ksig –señal de control de entrada.

asig –señal de audio de entrada.

11.6.3 NOMBRES DESECHADOS

Antes de la versión 3.63, la versión a frecuencia de control de **peak** se llamaba **peakk**. Ahora se puede usar **peak** tanto con entradas de control como de audio.

11.6.4 AUTOR

Robin Whittle
Australia
Mayo 1997

11.7 xyin, tempo

kx, ky	xyin	iprd, ixmin, ixmax, iymin, iymax[, ixinit, iyinit]
	tempo	ktempo, istartempo

11.7.1 DESCRIPCIÓN

Detecta la posición del cursor en una ventana de salida y aplica control de tempo a una partitura aún no interpretada, respectivamente. Cuando se llama a **xyin** la posición del ratón dentro de la pantalla de salida es usada para responder a dicha llamada. Este mecanismo simple implica que sólo puede ser usada con precisión una llamada a **xyin** a la vez. La posición del ratón se comunica en la ventana de salida.

11.7.2 INICIALIZACIÓN

iprd- período del proceso de detección del cursor (en segundos). Normalmente .1 segundos.

xmin, xmax, ymin, ymax - valores límite para las coordenadas x-y del cursor en la ventana de entrada.

ixinit, iyinit (opcional) - coordenadas x-y iniciales comunicadas. El valor por defecto es 0 para ambos. Si estos valores no están dentro del rango definido arriba son forzados a entrar en él.

istartempo - tiempo inicial (en pulsos por minuto). Normalmente 60.

11.7.3 EJECUCIÓN

xyin detecta la posición x-y del cursor en una ventana de entrada cada *iprd* segundos. Los valores de salida se repiten (no se interpolan) a frecuencia de control (-k), y se mantienen constantes hasta que un nuevo cambio es registrado en la ventana. Puede haber cualquier número de ventanas de entrada. Esta unidad es útil para obtener control de la ejecución en tiempo real, pero deben evitarse movimientos continuos si *iprd* es un valor inusualmente pequeño.

tempo permite controlar desde dentro de la orquesta la velocidad de ejecución de los eventos de la partitura de Csound. Opera sólo en presencia de un indicador **-t** de Csound. Cuando dicho indicador está operativo, los eventos de la partitura serán ejecutados, conforme a sus parámetros p2 y p3, inicialmente según el tiempo indicado en la línea de comando. Cuando se activa una sentencia **tempo** en un instrumento (*ktempo* 0), el tempo operativo se ajustará según el valor de *ktempo*, en pulsos por minuto. Puede haber cualquier número de sentencias **tempo** en una orquesta, pero se recomienda evitar activaciones simultáneas.

11.7.4 EJEMPLO

kx,ky	xyin	.05, 30, 0, 120, 0, 75	; detecta la posición del cursor
	tempo	kx, 75	; y controla el tiempo de la ejecución.

12 CONTROL DE INSTRUMENTOS: VALORES CONDICIONALES

12.1 >, <, >=, <=, ==, !=, ?

```
(a > b ? v1 : v2)
(a < b ? v1 : v2)
(a >= b ? v1 : v2)
(a <= b ? v1 : v2)
(a == b ? v1 : v2)
(a != b ? v1 : v2)
```

donde "a", "b", "v1" y "v2" pueden ser expresiones, a condición de que "a" y "b" no sean de tipo a-.

12.1.1 DESCRIPCIÓN

En las expresiones condicionales precedentes, "a" y "b" se comparan primero. Si la relación es verdadera ("a" mayor que "b", "a" menor que "b", "a" mayor o igual que "b", "a" menor o igual que "b", "a" igual que "b", "a" distinto de "b"), entonces la expresión condicional toma el valor de "v1"; si, por contra, la relación es falsa, la expresión toma el valor de "v2". (Por conveniencia, un único signo "=" funciona como "==".)

NB: Si "v1" o "v2" son expresiones, serán evaluadas antes que la condición. En términos de precedencia, todos los operadores condicionales (es decir, los operadores relacionales como >, <, ==, etc.) son posteriores a los operadores lógicos y aritméticos (como +, -, *, etc.)

12.1.2 EJEMPLO

```
(k1 < p5/2 + p6 ? k1 : p7)
```

la expresión $p5/2 + p6$ tiene prioridad. Para valores menores que dicha expresión se devolverá el valor de $k1$, para valores mayores el valor de $p7$.

13 CONTROL DE INSTRUMENTOS: MACROS

13.1 #define, \$NOMBRE, #undef

```
#define NOMBRE # texto a reemplazar #  
#define NOMBRE(a' b' c') # texto a reemplazar #  
$NOMBRE.  
#undef NOMBRE
```

13.1.1 DESCRIPCIÓN

Las macros son sustituciones textuales que toman lugar en el momento en que la orquesta está siendo leída. El sistema de macros en Csound es muy simple y usa dos caracteres especiales para indicar su presencia, los signos # y \$. Esto permite una más fácil escritura de las partituras, y proporciona una alternativa elemental a los sistemas de generación de partituras complejos. El sistema de macros es similar pero completamente independiente del sistema de macros del lenguaje de la partitura.

#define NOMBRE –define una macro simple. El nombre de la macro debe empezar con una letra y puede consistir en cualquier combinación de letras y números. Se distinguen mayúsculas y minúsculas. El nombre de las variables debe ser fijo. Se puede obtener más flexibilidad usando una macro con argumentos, tal y como se describe abajo.

#define NOMBRE (a' b' c') - define una macro con argumentos. Esto puede ser usado en situaciones más complejas. El nombre de una macro debe empezar con una letra y consistir en cualquier combinación de letras y números. Dentro del texto a reemplazar, los argumentos pueden ser sustituidos por \$A. De hecho, la implementación define los argumentos como macros simples. Puede haber hasta 5 argumentos, y los nombres pueden consistir en cualquier combinación de letras. Recuerda que se distingue entre mayúsculas y minúsculas.

\$NOMBRE. - llama a una macro previamente definida. Para usar una macro, su nombre es usado tras el carácter \$. Este nombre termina con el último carácter que no sea ni una letra ni un número. Si es preciso que el nombre no termine con un espacio, un punto, que será ignorado, puede ser usado para terminar dicho nombre. Por ejemplo, la cadena \$NOMBRE. será reemplazada por el texto definido en la macro. Por supuesto, este texto puede llamar a otras macros.

#undef NOMBRE – borra el nombre de una macro. Si una macro no va a ser usada en adelante puede ser borrada con **#undef NOMBRE**.

13.1.2 INICIALIZACIÓN

replacement text # - El texto que reemplazará cada aparición del nombre de la macro es cualquier cadena de caracteres (excepto #) y puede extenderse en más de una línea. Este texto está colocado entre los signos #, que aseguran que no se tomarán accidentalmente otros caracteres fuera de estos límites.

13.1.3 EJECUCIÓN

Se necesita algún cuidado con las macros textuales, ya que a veces juegan malas pasadas. Ellas no distinguen significado alguno, así que los espacios son significativos, por lo cual, en la definición, el texto que reemplazará a la macro va entre # #, a diferencia del lenguaje C. Usadas con cuidado, las macros son un concepto potente, pero no se puede abusar de ellas.

13.1.4 EJEMPLOS

13.1.4.1 Macro Simple

```
#define REVERB #ga = ga+a1
        out a1#

instr 1
a1    oscil
$REVERB.
endin

instr 2
a1    repluck
$REVERB.
endin
```

Esto dará lugar a:

```
instr 1
a1    oscil
ga = ga+a1
out a1
endin

instr 2
a1    repluck
ga = ga+a1
out a1
endin
```

13.1.4.2 Macro Con Argumentos

```
#define REVERB(A) #ga = ga+$A.
        out $A.#

instr 1
a1    oscil
$REVERB(a1)
endin

instr 2
a2    repluck
$REVERB(a2)
endin
```

Esto dará lugar:

```
instr 1  
a1 oscil  
ga = ga+a1  
out a1  
endin
```

```
instr 2  
a2 repluck  
ga = ga+a2  
out a2  
endin
```

13.1.5 AUTOR

John ffitch

Universidad de Bath/Codemist Ltd.

Bath, UK

Abril, 1998 (Nuevo en la versión 3.48)

13.2 #include

```
#include "filename"
```

13.2.1 DESCRIPCIÓN:

A veces es conveniente tener la orquesta dividida en varios ficheros, por ejemplo con cada instrumento en un fichero separado. La nueva opción **#include**, que es una parte del sistema de macros, soporta esta forma de configurar la orquesta. Para usarla debe haber una línea en la forma:

```
#include "nombrefichero"
```

donde el carácter `:` puede ser reemplazado por cualquier otro que sirva. Para la mayoría de los casos, las comillas serán lo más conveniente. El nombre del fichero puede incluir la ruta completa.

Hay actualmente un límite de 20 en los ficheros y macros incluidos. Otro posible uso de **#include** puede ser definir una serie de macros que sean significativas en el estilo de cada compositor.

Una forma extrema de usar las macros sería tener cada instrumento definido como una macro, con el número de instrumento como parámetro. Así, una orquesta entera puede ser construida sólo con sentencias **#include**, seguida con llamadas a las macros.

```
#include "clarinete"  
#include "flauta"  
#include "fagot"  
$CLARINETE(1)  
$FLAUTA(2)  
$FAGOT(3)
```

Debemos enfatizar que estos cambios son sólo a nivel textual y no representan nada significativo para el compilador.

13.2.2 AUTOR

John ffitch

Universidad de Bath/Codemist Ltd.

Bath, Reino Unido

Abril, 1998 (Nuevo en la versión 3.48)

14 CONTROL DE INSTRUMENTOS: CONTROL DEL FLUJO DEL PROGRAMA

14.1 igoto, tigo, kgoto, goto, if, timout

```
igoto      label
tigo      label
kgoto      label
goto       label
if         ia R ib      igoto      label
if         ka R kb      kgoto      label
if         ia R ib      goto        label
timout     istrtr, idur, label
```

donde *etiqueta* se encuentra siempre dentro del mismo bloque de instrumento y no es una expresión, y donde *R* es uno de los operadores relacionales (*,* *<*, *=*, *<=*, *==*, *!=*) (y = por comodidad, ver también **Valores Condicionales**)

14.1.1 DESCRIPCIÓN

Estas sentencias se usan para controlar el orden en el que se ejecutarán las demás sentencias dentro de un mismo instrumento. Se pueden controlar por separado las pasadas de inicialización (**i-**) y de ejecución (**k-**) de la siguiente manera:

igoto - durante pasadas de inicialización (i-) solamente, transfiere incondicionalmente el control de la ejecución a la sentencia señalada por la *etiqueta*.

tigo - similar a **igoto**, pero efectivo solamente durante pasadas de inicialización en las cuales una nueva nota está siendo ligada a una nota mantenida previamente (ver **Sentencia i**); no tiene ningún efecto cuando no existe ligadura. Permite a un instrumento saltarse la inicialización de sus unidades dependiendo de si existe ligadura o no (ver también **tival**, **delay**).

kgoto - durante pasadas de ejecución (k-) solamente, transfiere incondicionalmente el control de la ejecución a la sentencia señalada por la *etiqueta*.

goto - (combinación de **igoto** y **kgoto**) Transfiere el control a *etiqueta* en cada pasada.

if...igoto - bifurcación condicional de la ejecución en la inicialización (i-), dependiendo de que la expresión lógica "ia R ib" sea cierta o no. Se bifurcará la ejecución sólo si la condición es cierta.

if...kgoto - bifurcación condicional en tiempo de ejecución (k-), dependiendo de que la expresión lógica "ka R kb" sea cierta o no. Se bifurcará la ejecución sólo si la condición es cierta.

if...goto - combinación de las anteriores. Se comprueba la condición en cada pasada.

timeout - bifurcación condicional en tiempo de ejecución (k-), dependiendo del tiempo de la nota transcurrido. *istrt* e *idur* especifican tiempos en segundos. La bifurcación de la ejecución a *etiqueta* tendrá lugar en el instante señalado por *istrt* y se mantendrá durante *idur* segundos. Observa que **timeout** puede ser reinicializado para activaciones múltiples de una sola nota (ver el ejemplo de **reinit**)

14.1.2 EJEMPLO

```
if k3 p5 + 10 kgoto próximo
```

15 CONTROL DE INSTRUMENTOS: REINICIALIZACIÓN

15.1 reinit, rigoto, rireturn

```
reinit    label
rigoto    label
rreturn
```

15.1.1 DESCRIPCIÓN

Estas sentencias permiten a un instrumento reiniciarse durante la ejecución.

reinit - siempre que se encuentra esta sentencia durante una pasada en tiempo de ejecución, ésta es temporalmente suspendida mientras se realiza una pasada especial de inicialización, empezando en *etiqueta* y continuando hasta que se encuentra una sentencia **rireturn** o **endin**. Entonces la ejecución se reanuda en el punto en que se dejó.

rigoto - similar a **igoto**, pero efectivo solamente durante una pasada de reinicialización de **reinit** (es decir, no tiene ningún efecto en las pasadas de inicialización estándar). Esta sentencia es útil para saltarse unidades que no deben ser reiniciadas.

rireturn - termina una pasada de reinicialización de **reinit** (es decir, no tiene ningún efecto en las pasadas de inicialización estándar). Esta sentencia, junto con **endin**, permite que se reanude la ejecución normal.

15.1.2 EJEMPLO

Las siguientes sentencias generan una señal exponencial de control cuyos valores se mueven desde 440 a 880, exactamente 10 veces a lo largo de la duración indicada por p3:

```
reset:    timeout    0, p3 /10, contin    ; después de p3/10 segundos,
          reinit     reset              ; reinicializa tanto timeout
contin:   expon      440, p3/10,880      ; como expon
          rireturn   ; entonces reanuda la ejecución
```

16 MATEMÁTICAS OPERACIONES:

OPERACIONES ARITMÉTICAS Y LÓGICAS

16.1 -, +, &&, ||, *, /, ^, %

-	a	(sin restricción de frecuencia)
+	a	(sin restricción de frecuencia)
a	&& b	(Y lógico; no válido a frecuencia de audio)
a	b	(O lógico; no válido a frecuencia de audio)
a	+ b	(sin restricción de frecuencia)
a	- b	(sin restricción de frecuencia)
a	* b	(sin restricción de frecuencia)
a	/ b	(sin restricción de frecuencia)
a	^ b	(b no puede ser de tipo audio)
a	% b	(sin restricción de frecuencia)

donde los argumentos a y b pueden ser a su vez expresiones.

16.1.1 DESCRIPCIÓN

Los operadores aritméticos llevan a cabo operaciones de cambio de signo (negación), preservación de signo, "Y" y "O" lógicos, suma, resta, multiplicación y división. Observa que un valor o una expresión puede estar en medio de dos de tales operadores, los cuales lo tomarán como su argumento a la izquierda o a la derecha, como por ejemplo en:

$$a + b * c.$$

En dichos casos se aplican las siguientes reglas:

- * y / preceden en importancia a + y -. Así la expresión anterior se toma como:

$$a + (b * c)$$

donde el signo * afecta primero a "b" y a "c" y luego el signo + se aplica a "a" y a "b * c".

- + y - preceden en importancia a &&, que a su vez precede a ||:

$$a \&\& b - c \ || \ d$$

se toma como $(a \&\& (b-c)) \ || \ d$

- Cuando ambos operadores tienen la misma prioridad, las operaciones se realizan de izquierda a derecha:

$$a - b - c$$

se toma como $(a - b) - c$.

Los paréntesis pueden ser usados para forzar la precedencia de diversos grupos.

El operador ^ calcula "a" elevado a "b", pero "b" no puede ser una señal de audio. La precedencia de este operador puede no funcionar correctamente aún. Lee la sección 5.2. Nuevo en la versión 3.493

El operador % devuelve el valor absoluto de "b". Es lo mismo que el módulo de los números enteros. Nuevo en la versión 3.50

17 OPERACIONES MATEMÁTICAS: FUNCIONES MATEMÁTICAS

17.1 int, frac, i, abs, exp, log, log10, sqrt

int	(x)	(solamente argumentos i- o k-)
frac	(x)	(solamente argumentos i- o k-)
i	(x)	(solamente argumentos k-)
abs	(x)	(sin restricción de tipo)
exp	(x)	(sin restricción de tipo)
log	(x)	(sin restricción de tipo)
log10	(x)	(sin restricción de tipo)
sqrt	(x)	(sin restricción de tipo)

donde los argumentos entre paréntesis pueden ser expresiones.

17.1.1 DESCRIPCIÓN

Los convertidores de valor realizan traducciones aritméticas de unidades de una clase a unidades de otra. El resultado puede ser a su vez un término de una expresión.

int(x) devuelve la parte entera de x.

frac(x) devuelve la parte fraccionaria de x.

i(x) devuelve el equivalente en tipo i- de un valor x de tipo k-.

abs(x) devuelve el valor absoluto de x.

exp(x) devuelve el valor de e elevado a x.

log(x) devuelve el logaritmo natural de x (sólo para valores positivos de x)

log10(x) devuelve el logaritmo en base 10 de x (sólo para valores positivos de x)

sqrt(x) devuelve la raíz cuadrada de x (sólo para valores positivos de x)

Observa que los valores de x están restringidos para **log**, **log10**, **sqrt**.

17.2 powoftwo, logbtwo

```
powoftwo (x) (sólo argumentos de tipo i- o k-)  
logbtwo  (x) (sólo argumentos de tipo i- o k-)
```

17.2.1 DESCRIPCIÓN

Efectua operaciones de potenciación de dos.

17.2.2 EJECUCIÓN

powoftwo() devuelve 2 elevado a x y permite números tanto positivos como negativos como argumentos. El rango de valores permitidos en **powoftwo()** es de -5 a +5, proporcionando una precisión más fina que la centésima en un rango de 10 octavas. Si se requiere un rango mayor de valores, deberá usarse el opcodoe **pow**.

logbtwo() devuelve el logaritmo en base 2 de x. El rango de valores permitidos como argumentos es de .25 a 4 (esto es, una respuesta de -2 a +2 octavas). Esta función es la inversa de **powoftwo()**.

Estas funciones son rápidas, porque leen valores almacenados en tablas. También son muy útiles para trabajar con coeficientes de afinación. Trabajan a frecuencia de inicialización o control.

17.2.3 AUTORES

Gabriel Maldonado
Italia
Junio, 1998

John ffitch
Universidad de Bath, Codemist, Ltd.
Bath, Reino Unido
Julio, 1999
Nuevo en la versión 3.57

18 OPERACIONES MATEMÁTICAS: FUNCIONES TRIGONOMÉTRICAS

18.1 sin, cos, tan, sininv, cosinv, taninv, sinh, cosh, tanh

sin	(x)	(sin restricción de tipo)
cos	(x)	(sin restricción de tipo)
tan	(x)	(sin restricción de tipo)
sininv	(x)	(sin restricción de tipo)
cosinv	(x)	(sin restricción de tipo)
taninv	(x)	(sin restricción de tipo)
sinh	(x)	(sin restricción de tipo)
cosh	(x)	(sin restricción de tipo)
tanh	(x)	(sin restricción de tipo)

donde los argumentos entre paréntesis pueden ser expresiones.

18.1.1 DESCRIPCIÓN

Estas funciones calculan conversiones trigonométricas. El resultado puede ser a su vez un término de una expresión.

sin(x) devuelve el seno de x (x en radianes).

cos(x) devuelve el coseno x (x en radianes).

tan (x) devuelve la tangente de x

sininv(x) devuelve el arcoseno de x

cosinv(x) devuelve el arcoseno de x

taninv(x) devuelve la arcotangente de x

sinh(x) devuelve el seno hiperbólico de x

cosh(x) devuelve el coseno hiperbólico de x

tanh (x) devuelve la tangente hiperbólica de x

19 OPERACIONES MATEMÁTICAS: FUNCIONES DE AMPLITUD

19.1 dbamp, ampdb

`dbamp` (x) (init- or control-rate args only)
`ampdb` (x) (no rate restriction)

donde los argumentos entre paréntesis pueden ser expresiones.

19.1.1 DESCRIPCIÓN

Estas funciones realizan conversiones entre valores de amplitud y sus equivalentes en decibelios. El resultado puede ser a su vez un término de otra expresión.

`dbamp`(x) devuelve el equivalente en decibelios de la amplitud x.

`ampdb`(x) devuelve la amplitud equivalente al valor en decibelios x. Así:

60 dB = 1000
66 dB = 1995.262
72 dB = 3891.07
78 dB = 7943.279
84 dB = 15848.926
90 dB = 31622.764

20 OPERACIONES MATEMÁTICAS: FUNCIONES ALEATORIAS

20.1 rnd, birnd

rnd	(x)	(sólo argumentos i- o k-)
birnd	(x)	(sólo argumentos i- o k-)
ftlen	(x)	(sólo argumentos i-)
ftlptim	(x)	(sólo argumentos i-)
ftsr	(x)	(sólo argumentos i-)
nsamp	(x)	(sólo argumentos i-)

donde el argumento entre paréntesis puede ser a su vez una expresión.

20.1.1 DESCRIPCIÓN

Estos convertidores de valor muestrean una secuencia aleatoria global. El resultado puede usarse a su vez como término en otra expresión. Estos operadores no tienen en cuenta **seed**.

20.1.2 EJECUCIÓN

rnd(x) - devuelve un número aleatorio en el rango unipolar de 0 a x.

birnd(x) - devuelve un valor aleatorio en el rango bipolar de -x a x. Estas unidades obtienen los valores de un *generador de números pseudo-aleatorios* y luego los normaliza dentro de un rango determinado. Así, este único generador global pasará a todas estas unidades las secuencias aleatorias que éstas requieran a lo largo de su ejecución, en cualquier orden en que lleguen las llamadas.

20.1.3 AUTOR

Barry Vercoe
M.I.T., Cambridge, Mass
1997

21 FUNCIONES MATEMÁTICAS: OPCODES EQUIVALENTES A FUNCIONES

21.1 sum

```
ar sum asig1, asig2[, asig3...asigN]
```

21.1.1 DESCRIPCIÓN

Suma cualquier número de señales de audio.

21.1.2 EJECUCIÓN

asig1, etc. – señales de audio a sumar (mezcladas o añadidas).

21.1.3 AUTOR

Gabriel Maldonado
Italia
Abril, 1999
Nuevo en la versión 3.54

21.2 product

ar **product** asig1, asig2[, asig3...asigN]

21.2.1 DESCRIPCIÓN

Multiplica cualquier número de señales de audio.

21.2.2 EJECUCIÓN

asig1, etc. – señales de audio a multiplicar.

21.2.3 AUTOR

Gabriel Maldonado
Italia
Abril, 1999
Nuevo en la versión 3.54

21.3 pow

```
ir    pow    iarg, ipow
ir    =      iarg ^ ipow
kr    pow    karg, kpow, [inorm]
ar    pow    aarg, kpow, [inorm]
```

21.3.1 DESCRIPCIÓN

Calcula *xarg* elevado a *kpow* (o *ipow*) y normaliza el resultado según *inorm*.

21.3.2 INICIALIZACIÓN

inorm - el número por el que se divide el resultado (1 por defecto). Esto es especialmente útil si vas a calcular potencias de señales a- o k-, para evitar las muestras fuera de rango que suelen darse a menudo.

iarg – base (tipo i-).

21.3.3 EJECUCIÓN

karg – base (tipo k-)

aarg – base (tipo a-)

21.3.4 EJEMPLO

1. `i2t2 pow 2,2 ;` Calcula 2^2 .
2. `kline line 0, 1, 4`
`kexp pow kline, 2, 4`

Aquí, **pow** recibe una función lineal como argumento y normaliza el resultado según el valor más alto de la recta. La salida será una curva exponencial en el mismo intervalo que la recta de entrada.

3. `iamp pow 10, 2`
`a1 oscil iamp, 100, 1`
`a2 pow a1, 2, iamp`
`out a2`

Este ejemplo devolverá una función seno con sus valores negativos convertidos en positivos. El valor de pico será $iamp = 10^2 = 100$.

La línea 1 podría haberse escrito así:

```
i2t2 = 2 ^ 2
```

Usa `^` con cuidado en sentencias aritméticas porque la precedencia puede no funcionar bien. Nuevo en la versión 3.493.

21.3.5 NOMBRES DESECHADOS

pow estaba formado originalmente por tres opcodes llamados **ipow**, **kpow**, y **apow**. A partir de la versión 3.47 estos nombres fueron desechados y los tres opcodes fueron agrupados en **pow**.

21.3.6 AUTOR

Paris Smaragdis
MIT, Cambridge
1995

21.4 taninv2

ir	taninv2	ix, iy
kr	taninv2	kx, ky
ar	taninv2	ax, ay

21.4.1 DESCRIPCIÓN

Devuelve la arcotangente de iy/ix , ky/kx , o ay/ax . Si x o y son 0, **taninv2** devuelve 0.

21.4.2 INICIALIZACIÓN

ix , iy – valores a convertir.

21.4.3 EJECUCIÓN

kx , ky – señales a frecuencia de control a ser convertidas

ax , ay – señales a frecuencia de audio a ser convertidas

21.4.4 AUTOR

John ffitc

Universidad de Bath/Codemist Ltd.

Bath, Reino Unido

Abril, 1998 (Nuevi en la versión 3.48)

21.5 mac, maca

```
ar mac      asig1, ksig1, asig2, ksig2, asig3, ...
ar maca     asig1, asig2, asig3, asig4, asig5, ...
```

21.5.1 DESCRIPCIÓN

Multiplica y acumula señales de control o audio.

21.5.2 EJECUCIÓN

ksig1, etc. - señales de control de entrada.

asig1, etc. - señales de audio de entrada.

mac multiplica y acumula señales de tipo a- y k-. Es equivalente a:

```
ar = asig1 + ksig1*asig2 + ksig2+asig3 + ...
```

maca multiplica y acumula señales sólo de tipo a-. Es equivalente a:

```
ar = asig1 + asig2*asig3 + asig4+asig5 + ...
```

21.5.3 AUTOR

John ffitch
Universidad de Bath, Codemist, Ltd.
Bath, Reino Unido
Mayo, 1999
Nuevo en la versión 3.55

22 CONVERTIDORES DE ALTURA: FUNCIONES

22.1 octpch, pchoct, cspch, octcps, cpsoct

octpch (pch) (sólo argumentos de tipo i- o k-)
pchoct (oct) (sólo argumentos de tipo i- o k-)
cspch (pch) (sólo argumentos de tipo i- o k-)
octcps (cps) (sólo argumentos de tipo i- o k-)
cpsoct (oct) (sin restricción de tipo)

donde el argumento entre paréntesis puede ser a su vez una expresión.

22.1.1 DESCRIPCIÓN

Estos sí son literalmente **convertidores de valor**, con funciones especiales para manipular los datos que representan la altura de las notas.

Los datos que representan alturas o frecuencias pueden expresarse un cualquiera de los siguientes formatos:

octave point pitch-class (8ve.pc)	pch	división temperada de la octava
octave point decimal	oct	división decimal de la octava
cycles per second	cps	ciclos por segundo (Herzios)

Los dos primeros formatos consisten en un número entero, que representa la octava, seguido por una parte fraccionaria que se interpreta de distintas formas según el caso. Para **pch**, la fracción se lee como dos dígitos decimales que representan cualquiera de las 12 divisiones de la octava en el sistema de temperamento igual, desde .00 (do) a .11 (si). Las dos interpretaciones de la parte fraccionaria (la de **pch** y la **oct**) quedan relacionadas por el factor 100/12. En ambos formatos, la fracción debe ser precedida por un número entero que representa la octava, de tal manera que 8.00 equivale al do central, 9.00 al do inmediatamente superior, etc. Así, un la 440 Hz puede ser representado como 440 (**cps**), 8.09 (**pch**), 8.75 (**oct**), ó 7.21 (**pch**), etc. Las divisiones microtonales del semitono pueden ser representadas en **pch** usando más de dos cifras decimales.

Los mnemónicos de las unidades de conversión de altura se derivan de los morfemas de las abreviaturas inglesas implicadas. El segundo morfema indica la fuente y el primero el resultado. Así:

cspch (8.09)

convertirá el argumento 8.09 en su equivalente en cps (es decir en Herzios), dando lugar a un valor de salida de 440 .

Como el argumento es constante a lo largo de toda la duración de la nota, la conversión se efectúa en la inicialización, antes de que se produzca ninguna muestra de dicha nota. Por el contrario, la conversión:

cpsoct (8.75 + K1)

que devuelve un resultado de 440 transportado a la octava especificada por $k1$, repetirá el cálculo cada pasada de control ($k-$), ya que esta es la frecuencia a la que varía $k1$.

NB: La conversión de **pch** u **oct** a **cps** no es una operación lineal sino que implica un proceso exponencial que puede consumir mucho tiempo de cálculo cuando se ejecuta repetidamente. **Csound** usa ahora un proceso interno de búsqueda en tabla que realiza esta tarea eficientemente incluso en frecuencias de audio.

23 CONVERTIDORES DE ALTURA: OPCODES DE AFINACIÓN

23.2 cps2pch, cpsxpch

```
icps  cps2pch  ipch, iequal
icps  cpsxpch  ipch, iequal, irepeat, ibase
```

23.2.1 DESCRIPCIÓN

Convierte una notación en formato de "octava.nota" a otra en formato de ciclos por segundo (Herzios), para divisiones iguales de la octava (**cps2pch**) o para divisiones iguales de cualquier otro intervalo (**cpsxpch**). Hay un límite de 100 divisiones iguales.

23.2.2 INICIALIZACIÓN

ipch - valor de entrada en la forma "octava.nota"

iequal - si es positivo, indica el número de intervalos iguales en que se dividirá la octava. Deber ser menor o igual que 100. Si es negativo, es el número de una tabla de multiplicadores de frecuencia.

irepeat - es un número que indica el intervalo que hará las veces de "octava". El entero 2 corresponde a divisiones reales de octava, 3 corresponde a divisiones de duodécima, 4 a doble octava y así sucesivamente. Este valor no tiene porqué ser un entero, pero deberá ser siempre un número positivo.

ibase - frecuencia que corresponde a la altura 0.0

NOTAS:

1. Las siguientes expresiones son idénticas

```
ia = cpspch(8.02)
ib cps2pch 8.02, 12
ic cpsxpch 8.02, 12, 2, 1.02197503906
```

2. Estas unidades son opcodes, no funciones.

3. Se permiten valores negativos de *ipch*, pero *irepeat*, *iequal* y *ibase* deben ser positivos.

23.2.3 EJEMPLO

```
inote cps2pch p5, 19 ; convierte oct.pch a cps en 19 intervalos iguales
inote cpsxpch p5, 12, 3, 261.62561 ; escala de Pierce centrada en el "la" central
inote cpsxpch p5, 21, 4, 16.35160062496 ; escala 10.5ET
```

El uso de una tabla permite diseñar escalas exóticas, mapeando las frecuencias dentro de dicha tabla. Por ejemplo la tabla:

f2 0 16 -2 1 1.1 1.2 1.3 1.4 1.6 1.7 1.8 1.9

puede ser usada con:

ip **cps2pch** p4, -2

para obtener una escala dividida en 10 intervalos desiguales.

23.2.4 AUTOR

John ffitch

Universidad de Bath/Codemist Ltd.

Bath, Reino Unido

1997

24 SOPORTE MIDI: CONVERTIDORES

24.1 notnum, veloc, cpsmidi, cpsmidib, octmidi, octmidib, pchmidi, pchmidib, ampmidi, aftouch, pchbend, midictrl

ival	notnum	
ival	veloc	[ilow, ihigh]
icps	cpsmidi	
icps	cpsmidib	[irange]
kcps	cpsmidib	[irange]
ioct	octmidi	
ioct	octmidib	[irange]
koct	octmidib	[irange]
ipch	pchmidi	
ipch	pchmidib	[irange]
kpch	pchmidib	[irange]
iamp	ampmidi	iscal[, ifn]
kaft	aftouch	[imin[, imax]]
ibend	pchbend	[imin[, imax]]
kbend	pchbend	[imin[, imax]]
ival	midictrl	inum[imin[, imax]]
kval	midictrl	inum[imin[, imax]]

24.1.1 DESCRIPCIÓN

Lee un valor de evento MIDI que active un determinado instrumento, o el valor de un controlador MIDI continuo, y lo convierte a un formato local con el que poder trabajar.

24.1.2 INICIALIZACIÓN

iscal - factor de escala (tipo i-)

ifn (opcional) - número de la tabla de la función normalizadora de conversión que interpretará cada valor recibido. El valor por defecto es 0, lo que indica que no habrá conversión.

inum, ictlno - número del controlador MIDI.

initial - valor inicial del controlador.

ilow, ihigh - máximo y mínimo del rango del mapeado.

irange - rango del pitch bend en semitonos.

ichnl - canal MIDI

imin, imax – límites máximo y mínimo de los valores obtenidos.

24.1.3 EJECUCIÓN

notnum, veloc - lee el valor del byte MIDI (0-127) que indica el número de nota o la dinámica (velocidad) del evento actual respectivamente.

cpsmidi, octmidi, pchmidi - lee el número de nota del evento MIDI actual, expresado respectivamente en unidades cps, oct, o pch, para su posterior procesado local.

cpsmidib, octmidib, pchmidib - lee el número de nota del evento MIDI actual, lo modifica por el valor del pitch bend en ese momento y expresa el resultado en unidades cps, oct o pch respectivamente. Se puede usar como un valor i- o como un valor k- continuos.

ampmidi - lee la dinámica (velocidad) del evento MIDI actual, lo pasa opcionalmente por una tabla normalizadora y devuelve una amplitud en el rango 0 - *iscal*.

aftouch, pchbend - lee la postpulsación actual, la presión de canal o el valor del pitch bend para un determinado canal, normalizado en el rango de 0 a *iscal*. Observa que este acceso a los datos del pitch bend es completamente independiente de la altura MIDI, lo que permite que tales datos puedan ser empleados para cualquier otro uso.

midictrl - lee el valor actual (0-127) de un determinado controlador MIDI

24.1.4 AUTOR

Barry Vercoe - Mike Berry
MIT - Mills
Mayo 1997

24.2 cpstmid

icps cpstmid ifn

24.2.1 INICIALIZACIÓN

ifn - número de la tabla de función que contiene los parámetros (*numgrades*, *interval*, *basefreq*, *basekeymidi*) y las razones (proporciones) de afinación.

24.2.2 EJECUCIÓN

Sólo en la inicialización.

Esta unidad es parecida a **cpsmidi**, pero permite el uso de escalas microtonales hechas a medida. Requiere 5 parámetros: el primero, *ifn*, es el número de la tabla con las razones (coeficientes) de afinación, mientras que los otros 4 parámetros deben ser almacenados en la tabla de función misma. Esta puede ser generada por **GEN2** y los primeros cuatro valores almacenados en ella son:

1. *numgrades* (el número de grados de la escala microtonal).
2. *interval* (la frecuencia múltiplo que separa dos grados sucesivos, por ejemplo, 2 para la octava, 1.5 para la quinta, etc).
3. *basefreq* (la frecuencia base de la escala en Hz).
4. *basekeymidi* (el número de la nota MIDI a la que se asigna la frecuencia base de la escala).

Tras estos cuatro valores, el usuario puede empezar a insertar las razones (coeficientes) de afinación. Por ejemplo, para una escala temperada estándar con una frecuencia base de 261 Hz, asignada a la nota número 60, la **sentencia f** de la partitura que generaría la tabla correspondiente sería:

```
;      numgrades      basefreq      tuning-ratios (temperamento igual)
;      interval      basekeymidi
f1 0 64 -2 12 2 261 60 1 1.059463094359
1.122462048309 1.189207115003 ..etc...
```

Otro ejemplo con una escala de 24 grados, con una frecuencia de 440 asignada a la nota número 48 y un intervalo de repetición de 1.5 (una quinta):

```
;      numgrades      basefreq      tuning-ratios .....
;      interval      basekeymidi
f1 0 64 -2 24 1.5 440 48 1 1.01 1.02 1.03
..etc...
```

24.2.3 AUTOR

Gabriel Maldonado

Italia

1998 (Nuevo en la Versión 3.492)

25 SOPORTE MIDI: ENTRADA DE CONTROLADORES

25.1 *initc7*, *initc14*, *initc21*

```
initc7      ichan, ictrlno, ivalue
initc14     ichan, ictrlno1, ictrlno2, ivalue
initc21     ichan, ictrlno1, ictrlno2, ictrlno3, ivalue
```

25.1.1 DESCRIPCIÓN

Inicializa el controlador MIDI *ictrlno* con el valor *ivalue*.

25.1.2 INICIALIZACIÓN

ichan - canal MIDI.

ictrlno - número del controlador MIDI (*initc7*).

ictrlno1 - byte más significativo del número de controlador.

ictrlno2 - en ***initc14***, byte menos significativo del número de controlador; en ***initc21***, byte medio significativo del número de controlador.

ictrlno3 - byte menos significativo del número de controlador.

ivalue - valor en coma flotante (debe estar entre 0 y 1).

25.1.3 EJECUCIÓN

initc7, ***initc14*** y ***initc21*** pueden usarse junto con los opcodes **(i)midicXX** y **(i)ctrlXX** para inicializar el primer valor que tome el controlador. El argumento *ivalue* deber estar especificado tomando un valor entre 0 y 1. Si no es así ocurre un error. Usa la siguiente formula para configurar *ivalue* según los valores máximo y mínimo de **(i)midicXX** y **(i)ctrlXX**:

$$ivalue = (\text{valor inicial} - \text{mínimo}) / (\text{máximo} - \text{mínimo})$$

25.1.4 AUTOR

Gabriel Maldonado

Italia

Nuevo a la versión 3.47

25.2 **midic7, midic14, midic21, ctrl7, ctrl14, ctrl21**

```
idest midic7          ictrlno, imin, imax [, ifn]
kdest midic7          ictrlno, kmin, kmax [, ifn]
idest midic14         ictrlno1, ictrlno2, imin, imax [, ifn]
kdest midic14         ictrlno1, ictrlno2, kmin, kmax [, ifn]
idest midic21         ictrlno1, ictrlno2, ictrlno3, imin, imax [, ifn]
kdest midic21         ictrlno1, ictrlno2, ictrlno3, kmin, kmax [, ifn]
idest ctrl7           ichan, ictrlno, imin, imax [,ifn]
kdest ctrl7           ichan, ictrlno, kmin, kmax [,ifn]
idest ctrl14          ichan, ictrlno1, ictrlno2, imin, imax [,ifn]
kdest ctrl14          ichan, ictrlno1, ictrlno2, kmin, kmax [,ifn]
idest ctrl21          ichan, ictrlno1, ictrlno2, ictrlno3, imin, imax [,ifn]
kdest ctrl21          ichan, ictrlno1, ictrlno2, ictrlno3, kmin, kmax [,ifn]
```

25.2.1 DESCRIPCIÓN

Permite la entrada de señales de control MIDI.

25.2.2 INICIALIZACIÓN

idest - señal de salida.

ictrlno - número de controlador MIDI (1-127).

ictrlno1 - byte más significativo del número del controlador MIDI (1-127).

ictrlno2 - en **midic14**, byte menos significativo del número del controlador MIDI (1-127); en **midic21**, byte medio significativo del número del controlador MIDI (1-127).

ictrlno3 - byte menos significativo del número del controlador MIDI (1-127).

imin - valor mínimo definido por el usuario, en coma flotante, de la salida.

imax - valor máximo definido por el usuario, en coma flotante, de la salida.

ifn (opcional) - número de la tabla a leer cuando sea necesario utilizar un índice. Dicha tabla deberá estar normalizada. La salida será normalizada según los valores máximo y mínimo.

25.2.3 EJECUCIÓN

kdest - señal de salida.

kmin - valor mínimo definido por el usuario, en coma flotante, de la salida.

kmax - valor máximo definido por el usuario, en coma flotante, de la salida.

midic7 (control MIDI de 7 bits, tipos i- y k- respectivamente) permite señales MIDI de 7 bits en coma flotante, normalizadas dentro de un rango definido por sus valores máximo y mínimo. También

proporcionan opcionalmente la indexación de tablas sin interpolación. En **midic7** los valores máximo y mínimo pueden ser modificados en tiempo de ejecución (k-).

midic14 (control MIDI de 14 bits, tipos i- y k-) realizan el mismo trabajo que los anteriores pero con 14 bits de precisión.

midic21 (control MIDI de 21 bits, tipos i- y k-) igual que los anteriores pero con 21 bits de precisión.

midic14 y **midic21** pueden usar indexación de tablas con interpolación. Requieren dos o tres controladores MIDI como entrada.

ctrl7, **ctrl14** y **ctrl21** son muy parecidos a los opcodes **(i)midicXX**, con las únicas diferencias de que:

- los generadores **(i)ctrlXX** pueden utilizarse con instrumentos orientados a la partitura sin que Csound se cuelgue.
- necesitan un parámetro adicional *ichan* con el número de canal MIDI del controlador. El canal MIDI será el mismo para todos los controladores usados en un único opcode **(i)ctrl14** o **(i)ctrl21**.

25.2.4 NONMBRES DESECHADOS

Los opcodes **imidic7**, **imidic14**, **imidic21**, **ictrl7**, **ictrl14**, y **ictrl21** han sido desechados en la versión 3.52. En su lugar usa **midic7**, **midic14**, **midic21**, **ctrl7**, **ctrl14** y **ctrl21**, respectivamente, con salidas de tipo i-.

25.2.5 AUTOR

Gabriel Maldonado
Italia
Mayo 1997

25.3 chanctl

```
ival chanctl ichnl, ictlno[,ilow,ihigh]  
kval chanctl ichnl, ictlno[,ilow,ihigh]
```

25.3.1 DESCRIPCIÓN

Lee el valor actual de un controlador y lo mapea opcionalmente en un rango especificado. *ichnl* es y *ictlno* es el

25.3.2 INICIALIZATION

ichnl – número del canal MIDI

ictlno – número del controlador MIDI.

ilow, *ihigh* - valores inferior y superior para el mapeado.

25.3.3 AUTHOR

Mike Berry
Mills College
May 1997

26 SOPORTE MIDI: BANCOS DE MANDOS DESLIZANTES

26.1 slider8, slider16, slider32, slider64, slider8f, slider16f, slider32f, slider64f, s16b14, s32b14

i1, ..., i8	slider8	ichan, ictlnum1, imin1, imax1, ifn1,, \\ ictlnum8, imin8, imax8, ifn8
k1, ..., k8	slider8	ichan, ictlnum1, imin1, imax1, init1, ifn1, \\ ..., ictlnum8, imin8, imax8, init8, ifn8
i1, ..., i16	slider16	ichan, ictlnum1, imin1, imax1, ifn1,, \\ ictlnum16, imin16, imax16, ifn16
k1, ..., k16	slider16	ichan, ictlnum1, imin1, imax1, init1, ifn1, \\, ictlnum16, imin16, imax16, init16, ifn16
i1, ..., i32	slider32	ichan, ictlnum1, imin1, imax1, ifn1,, \\ ictlnum32, imin32, imax32, ifn32
k1, ..., k32	slider32	ichan, ictlnum1, imin1, imax1, init1, fn1, \\, ictlnum32, imin32, imax32, init32, ifn32
i1, ..., i64	slider64	ichan, ictlnum1, imin1, imax1, ifn1,, \\ ictlnum64, imin64, imax64, ifn64
k1, ..., k64	slider64	ichan, ictlnum1, imin1, imax1, init1, ifn1, \\, ictlnum64, imin64, imax64, init64, ifn64
k1, ..., k8	slider8f	ichan, ictlnum1, imin1, imax1, init1, ifn1, \\ icutoff1,, ictlnum8, imin8, imax8, init8, ifn8, \\ icutoff8
k1, ..., k16	slider16f	ichan, ictlnum1, imin1, imax1, init1, ifn1, \\ icutoff1,, ictlnum16, imin16, imax16, \\ init16, ifn16, icutoff16
k1, ..., k32	slider32f	ichan, ictlnum1, imin1, imax1, init1, ifn1, \\ icutoff1,, ictlnum32, imin32, imax32, \\ init32, ifn32, icutoff32
k1, ..., k64	slider64f	ichan, ictlnum1, imin1, imax1, init1, ifn1, \\ icutoff1,, ictlnum64, imin64, imax64, \\ init64, ifn64, icutoff64
i1, ..., i16	s16b14	ichan, ictlno_msbl, ictlno_lsbl, imin1, imax1, \\ initvalue1, ifn1,, ictlno_msbl6, \\ ictlno_lsbl6, imin16, imax16, initvalue16, ifn16
k1, ..., k16	s16b14	ichan, ictlno_msbl, ictlno_lsbl, imin1, imax1, \\ ifn1, ictlno_msbl6, ictlno_lsbl6, imin16, \\ imax16, ifn16
i1, ..., i32	s32b14	ichan, ictlno_msbl, ictlno_lsbl, imin1, imax1, \\ initvalue1, ifn1,, ictlno_msbl32, \\ ictlno_lsbl32, imin32, imax32, initvalue32, ifn32
k1, ..., k32	s32b14	ichan, ictlno_msbl, ictlno_lsbl, imin1, imax1, \\ ifn1,, ictlno_msbl32, ictlno_lsbl32, imin32, \\ imax32, ifn32

26.1.1 DESCRIPCIÓN

Bancos de mandos deslizantes de control MIDI.

26.1.2 INICIALIZACIÓN

i1 ... i64 – valores de salida.

ichan – canal MIDI (1-16).

ictlnum1 ... ictlnum64 – número de controlador MIDI.

ictlno_msb1 ... ictlno_msb32 – byte más significativo del número de controlador MIDI.

ictlno_lsb1 ... ictlno_lsb32 – byte menos significativo del número de controlador MIDI.

imin1 ... imin64 – valores mínimos para cada controlador.

imax1 ... imax64 – valores máximos para cada controlador.

init1 ... init64 – valor inicial para cada controlador.

ifn1 ... ifn64 – tabla de función de conversión para cada controlador.

icutoff1 ... icutoff64 – frecuencia de corte del filtro pasa bajos de cada controlador.

26.1.3 EJECUCIÓN

k1 ... k64 - valores de salida.

sliderN y **sliderNf** son bancos de controladores MIDI (útiles cuando se usa un mezclador MIDI como el Kawai MM-16 u otros similares para cambiar cualquier parámetro de sonido en tiempo real. Un banco de mandos deslizantes en software estará disponible en breve). Los mensajes MIDI de control en el puerto de entrada son convertidos de acuerdo con *iminN* y *imaxN*. Puede configurarse un valor inicial para cada uno de ellos. Del mismo modo, se puede usar una tabla de función, sin interpolación, que contenga una curva de conversión, lo que es útil, por ejemplo, cuando se requieren curvas de respuesta exponenciales.

Cuando no hace falta ninguna tabla de conversión, *ifnN* debe ponerse a 0. En caso contrario debe contener el número de la tabla de la función de conversión. Cuando dicha tabla es habilitada (es decir, cuando *ifnN* tiene un valor distinto a 0 que indica una tabla de función ya almacenada), el valor de *initN* debe ser igual a *iminN* o *imaxN*, si no, el valor inicial de salida no podrá ser el mismo que se especifica en el argumento *initN*.

slider8 proporciona un banco de 8 mensajes de control MIDI diferentes, **slider16** hace lo propio con un banco de 16 controladores y así sucesivamente.

sliderNf filtra la señal antes de su salida, para eliminar discontinuidades debidas a la baja resolución del MIDI (7 bits); la frecuencia de corte puede ser ajustada para cada controlador por separado (rango sugerido: .1 a 5 cps)

Como los argumentos de entrada y de salida son muchos, puedes partir la línea usando el carácter \ (nuevo en la versión 3.47) para mejorar la facilidad de lectura. Usar estos opcodes resulta bastante mejor que usar opcodes separados (**ctrl17** y **ktone**) si se necesitan más controladores.

En **isliderN** no hay un valor inicial de entrada para el argumento porque la salida es obtenida directamente del estado de la cadena interna de controladores de Csound en ese momento.

sNb14 es la versión de 14 bits de esta serie de controladores.

¡Atención! los opcodes **sliderNf** no envían el valor inicial requerido inmediatamente, sino sólo después de algunos ciclos de control (k-) porque el filtro retrasa levemente la señal.

26.1.4 NOMBRES DESECHADOS

Los opcodes **islider8**, **islider16**, **islider32**, **islider64**, **is16b14** y **is32b14** han sido desechados en la versión 3.52 de Csound. Usa **slider8**, **slider16**, **slider32**, **slider64**, **s16b14** y **s32b14**, respectivamente, para salidas de tipo i-.

26.1.5 AUTOR

Gabriel Maldonado
Italia
Diciembre 1998
Nuevo en la Versión 3.50

27 SOPORTE MIDI: E/S GENÉRICA

27.1 midiin

`kstatus`, `kchan`, `kdata1`, `kdata2` **midiin**

27.1.1 DESCRIPCIÓN

Devuelve un mensaje MIDI genérico recibido en el puerto de entrada MIDI.

27.1.2 EJECUCIÓN

`kstatus` - tipo de mensaje MIDI recibido. Puede ser:

- 128 (*note off*, nota desactivada),
- 144 (*note on*, nota activada),
- 160 (*polyphonic aftertouch*, postpulsación polifónica),
- 176 (*control change*, cambio de controlador),
- 192 (*program change*, cambio de programa),
- 208 (*channel aftertouch*, postpulsación),
- 224 (*pitch bend*, rueda de altura)

ó 0, si no hay ningún mensaje pendiente en el buffer de entrada MIDI.

`kchan` - canal MIDI (1-16).

`kdata1`, `kdata2` - valores de los datos (depende del mensaje).

midiin no tiene argumentos de entrada porque los lee precisamente del puerto MIDI. Trabaja en tiempo de ejecución (k-). Normalmente (es decir, cuando no hay mensajes pendientes) `kstatus` es 0. Sólo cuando hay datos MIDI en el buffer de entrada, `kstatus` toma el valor del mensaje MIDI operativo.

27.1.3 AUTOR

Gabriel Maldonado

Italia

1998 (Nuevo en la Versión 3.492)

27.2 midiout

`midiout` *kstatus*, *kchan*, *kdata1*, *kdata2*

27.2.1 DESCRIPCIÓN

Envía un mensaje MIDI genérico al puerto de salida MIDI

27.2.2 EJECUCIÓN

kstatus - tipo de mensaje MIDI. Puede ser:

128 (*note off*, nota desactivada),
144 (*note on*, nota activada),
160 (*polyphonic aftertouch*, postpulsación polifónica),
176 (*control change*, cambio de controlador),
192 (*program change*, cambio de programa),
208 (*channel aftertouch*, postpulsación),
224 (*pitch bend*, rueda de altura)
ó 0, si no hay ningún mensaje que enviar al puerto de salida MIDI.

kchan - canal MIDI (1-16).

kdata1, *kdata2* - valores de los datos (depende del mensaje).

midiout no tiene argumentos de salida porque los envía al puerto de salida MIDI directamente. Trabaja en tiempo de ejecución. Envía un mensaje MIDI sólo cuando *kstatus* es distinto de 0. ¡Precaución! Normalmente *kstatus* debe ser puesto a 0. Sólo cuando el usuario quiera mandar un mensaje MIDI debe *kstatus* contener el número de mensaje correspondiente.

27.2.3 AUTOR

Gabriel Maldonado
Italia 1998

28 SOPORTE MIDI: ACTIVACIÓN / DESACTIVACIÓN DE NOTA

28.1 **noteon**, **noteoff**, **noteondur**, **noteondur2**

noteon	<i>ichn</i> , <i>inum</i> , <i>ivel</i>
noteoff	<i>ichn</i> , <i>inum</i> , <i>ivel</i>
noteondur	<i>ichn</i> , <i>inum</i> , <i>ivel</i> , <i>idur</i>
noteondur2	<i>ichn</i> , <i>inum</i> , <i>ivel</i> , <i>idur</i>

28.1.1 DESCRIPCIÓN

Envía mensajes de activación y desactivación de nota al puerto de salida MIDI.

28.1.2 INICIALIZACIÓN

ichn - número de canal MIDI (0-15).

inum - número de nota (0-127).

ivel - velocidad (dinámica) (0-127).

28.1.3 EJECUCIÓN

noteon (nota activada en la inicialización) y **noteoff** (nota desactivada en la inicialización) son los opcodes más simples de salida MIDI. **noteon** envía un mensaje de nota activada al puerto de salida MIDI y **noteoff** un mensaje de nota desactivada. Un opcode **noteon** debe estar seguido siempre por un **noteoff** con el mismo número de canal y dentro del mismo instrumento. De otra manera se ejecutará sin fin. Estos opcodes son útiles cuando se introduce una sentencia **timout** para ejecutar una nota MIDI con una duración de distinta de 0. Para la mayoría de los propósitos, es mejor usar **noteondur** y **noteondur2**.

noteondur y **noteondur2** (nota activada especificando su duración en la inicialización) envía un mensaje MIDI de nota activada y nota desactivada respectivamente, ambos con el mismo canal, número y velocidad. El mensaje de nota desactivada se envía después de que hayan transcurrido *idur* segundos desde que se activó **noteondur**.

noteondur se diferencia de **noteondur2** en que **noteondur** trunca la duración de la nota cuando el instrumento actual es desactivado por la partitura o por una interpretación en tiempo real, mientras que **noteondur2** prolongará la ejecución del instrumento hasta que hayan transcurrido *idur* segundos. En la interpretación en tiempo real es aconsejable usar **noteondur** también para duraciones no definidas, dando un valor muy alto a *idur*.

Pueden aparecer cualquier número de opcodes **noteondur** o **noteondur2** en el mismo instrumento, lo que permite la ejecución de acordes en dicho único instrumento.

28.1.4 CAMBIO DE NOMBRES

Antes de la versión 3.52 (Febrero de 1999), estos opocodes se llamaban **ion**, **ioff**, **iondur** y **iodur2**. **ondur** y **ondur2** se cambiaron posteriormente por **noteondur** y **noteondur2** en la versión 3.53.

28.1.4 AUTOR

Gabriel Maldonado

Italia

Mayo 1997

28.2 moscil, midion

moscil	<i>kchn, knum, kvel, kdur, kpause</i>
midion	<i>kchn, knum, kvel</i>

28.2.1 DESCRIPCIÓN

Envía una cadena de mensajes de activación y desactivación de nota al puerto de salida MIDI.

28.2.2 EJECUCIÓN

kchn - número de canal MIDI (0-15).

knum - número de nota (0-127).

kvel - velocidad (dinámica) (0-127).

kdur - duración de la nota en segundos.

kpause - duración en segundos de la pausa entre cada mensaje de desactivación de nota y la nueva nota.

moscil y **midion** son los opcodes más potentes de salida MIDI. **moscil** (del inglés MIDI oscil, oscilador MDI) ejecuta una cadena de notas de *kdur* segundos de duración. El canal, la altura, la dinámica, la duración y la pausa pueden ser controlados en tiempo de ejecución (k-), permitiendo la generación de líneas melódicas complejas mediante algoritmos. Cuando el instrumento es desactivado, la nota que **moscil** está ejecutando en ese instante se ve forzada a truncarse (es decir, a cesar bruscamente).

midion (activación de nota MIDI en tiempo de ejecución) ejecuta notas MIDI con los valores *kchn*, *knum* y *kvel*. Estos argumentos pueden ser modificados en tiempo de ejecución (k-). Cada vez que el valor MIDI convertido de alguno de estos argumentos cambia, la nota MIDI ejecutada en última instancia por **midion** es inmediatamente desactivada y una nueva nota con sus nuevos argumentos es activada. Este opcode, como **moscil**, puede generar ricas texturas melódicas si es controlado por señales de tipo k- complejas.

Pueden aparecer cualquier número de opcodes **moscil** o **midion** en el mismo instrumento de Csound, lo que permite la realización de polifonías contrapuntísticas dentro de un mismo instrumento.

28.2.3 NOMBRES DESECHADOS

midion se llamaba originalmente **kon**. A partir de la versión 3.493, se desechó ese nombre sustituyéndolo por **midion**.

28.2.4 AUTOR

Gabriel Maldonado

Italia

Mayo de 1997 (**moscil** nuevo en la versión 3.47)

28.3 midion2

`midion2 kchn, knum, kvel, ktrig`

28.3.1 DESCRIPCIÓN

Envía mensajes de *nota activada* y *nota desactivada* al puerto de salida MIDI cuando se dispara al tomar un valor distinto a 0.

28.3.2 EJECUCIÓN

kchn - canal MIDI.

knum - número de nota MIDI.

kvel - velocidad de la nota.

ktrig - señal de disparo (normalmente 0).

Al igual que **midion**, este opcode envía mensajes de nota activada y nota desactivada al puerto MIDI, pero sólo cuando *ktrig* es distinto a 0. Este opcode puede trabajar junto a la salida producida por el opcode **trigger**.

28.3.3 AUTOR

Gabriel Maldonado

Italia

1998 (Nuevo en la Versión 3.492)

29 MIDI SOPORTE: SALIDA DE MENSAJES MIDI

29.1 outic, outkc, outic14, outkc14, outipb, outkpb, outiat, outkat, outipc, outkpc, outipat, outkpat

outic	<i>ichn, inum, ivalue, imin, imax</i>
outkc	<i>kchn, knum, kvalue, kmin, kmax</i>
outic14	<i>ichn, imsb, ilsb, ivalue, imin, imax</i>
outkc14	<i>kchn, kmsb, klsb, kvalue, kmin, kmax</i>
outipb	<i>ichn, ivalue, imin, imax</i>
outkpb	<i>kchn, kvalue, kmin, kmax</i>
outiat	<i>ichn, ivalue, imin, imax</i>
outkat	<i>kchn, kvalue, kmin, kmax</i>
outipc	<i>ichn, iprog, imin, imax</i>
outkpc	<i>kchn, kprog, kmin, kmax</i>
outipat	<i>ichn, inotenum, ivalue, imin, imax</i>
outkpat	<i>kchn, knotenum, kvalue, kmin, kmax</i>

29.1.1 DESCRIPCIÓN

Envía un mensaje de canal único al puerto de salida MIDI.

29.1.2 EJECUCIÓN

ichn, kchn - número de canal MIDI (0-15).

inum, knum - número del controlador MIDI (0-127, por ejemplo 1=Rueda de modulación etc...).

ivalue, kvalue - valor en coma flotante.

imin, kmin - valor mínimo en coma flotante (convertido en el valor entero MIDI 0).

imax, kmax - valor máximo en coma flotante (convertido en el valor entero MIDI 127 (para 7 bits) o 16383 (para 14 bit)).

imsb, kmsb - byte más significativo del número del controlador MIDI usando parámetros de 14 bits.

ilsb, klsb - byte menos significativo del número del controlador MIDI usando parámetros de 14 bits.

iprog, kprog - número de cambio de programa MIDI en coma flotante.

inotenum, knotenum - número de nota MIDI (usado en mensajes postpulsación polifónica).

outic y **outkc** (salida de controlador MIDI, tipos i- y k- respectivamente) envían mensajes de controlador a la salida MIDI.

outi14 y **outk14** (salida de controlador MIDI de 14 bits, tipos i- y k- respectivamente) envían un par de mensajes de control. Estos opcodes pueden pasar parámetros de 14 bits a aquellos instrumentos MIDI que los reconozcan. El primer mensaje de control contiene el byte más significativo del valor del argumento i- (o k-), mientras que el segundo contiene el byte menos significativo. $i(k)msb$ y $i(k)lsb$ son respectivamente el byte más y menos significativo del número del controlador MIDI.

outipb y **outkpb** (salida de pitch bend, tipos i- y k- respectivamente) envía mensajes de pitch bend.

outiat y **outkat** (salida de postpulsación, tipos i- y k- respectivamente) envía mensajes de postpulsación (aftertouch).

outipc y **outkpc** (salida de cambio de programa, tipos i- y k- respectivamente) envía mensajes de cambio de programa.

outipat y **outkpat** (salida de postpulsación polifónica, tipos i- y k- respectivamente) envía mensajes de postpulsación polifónica (*polyphonic aftertouch*).

Estos opcodes pueden pasar un valor distinto de parámetro para cada nota activa en ese momento. Sólo funcionan con los instrumentos MIDI que sean capaces de reconocerlos.

NB: todos estos opcodes pueden normalizar el valor i- (o k-) en coma flotante del argumento según los valores máximo $i(k)max$ y mínimo $i(k)min$. Por ejemplo, estableciendo $i(k)min = 1.0$ y $i(k)max = 2.0$, cuando el argumento i- (o k-) reciba el valor 2.0, el opcode enviará un valor 127 al dispositivo de salida MIDI, mientras que si recibe el valor 1.0 enviará un 0. Los opcodes de tipo i- envían sus mensajes una vez durante la inicialización del instrumento. Los opcodes de tipo k- envían sus mensajes cada vez que el valor convertido a MIDI del argumento i- (o k-) cambia.

29.1.3 NOMBRES DESECHADOS

Antes de la versión 3.52, estos opcodes se llamaban **ioutc**, **koutc**, **ioutc14**, **koutc14**, **ioutpb**, **koutpb**, **ioutat**, **koutat**, **ioutpc**, **koutpc**, **ioutpat**, y **koutpat**. Los nombres actuales fueron adoptados en la versión 3.52 (Febrero de 1999) para evitar confusión.

29.1.3 AUTOR

Gabriel Maldonado
Italia
Mayo 1997

29.2 nrpn

nrpn *kchan*, *kparmnum*, *kparmvalue*

29.2.1 DESCRIPCIÓN

Envía un mensaje de número de parámetro no registrado (NPRN) al puerto de salida MIDI cada vez que cambia uno de sus argumentos de entrada.

29.2.2 EJECUCIÓN

kchan - canal MIDI.

kparmnum - número del parámetro NRPN.

kparmvalue - valor del parámetro NRPN.

Este opcode envía un nuevo mensaje cuando el valor de uno de los argumentos de entrada cambia. Opera en tiempo de ejecución. Es útil con los instrumentos que reconocen los mensajes NRPN (por ejemplo, las nuevas tarjetas de sonido que incorporan un sintetizador MIDI interno, como la Sound Blaster AWE32, la SB AWE64, GUS, etc..., en las que cualquier parámetro de un instrumento puede ser cambiado durante la interpretación mediante mensajes NRPN)

29.2.3 AUTOR

Gabriel Maldonado

Italia

1998 (Nuevo en la Versión 3.492)

29.3 mdelay

`mdelay` *kstatus*, *kchan*, *kd1*, *kd2*, *kdelay*

29.3.1 DESCRIPCIÓN

Un opcode de retardo MIDI.

29.3.2 EJECUCIÓN

kstatus - estado del byte del mensaje MIDI a retardar.

kchan - canal MIDI (1-16).

kd1 - primer byte de datos MIDI.

kd2 - segundo byte de datos MIDI.

kdelay - tiempo de retardo en segundos.

Cada vez que *kstatus* es distinto de 0, **mdelay** manda un mensaje MIDI al puerto de salida MIDI después de *kdelay* segundos. Este opcode es útil para implementar retardos MIDI. Puede haber varias apariciones del opcode **mdelay** en el mismo instrumento con diferentes argumentos, de manera que se pueden implementar ecos MIDI ricos y complejos. Lo que es más, el tiempo de retardo puede ser variado en frecuencia de control.

29.3.3 AUTOR

Gabriel Maldonado

Italia

Noviembre, 1998 (Nuevo en la versión 3.492)

30 SOPORTE MIDI: MENSAJES EN TIEMPO REAL

30.1 `mclock`, `mrtmsg`

```
mclock ifreq  
mrtmsg imsgtype
```

30.1.1 DESCRIPCIÓN

Envía un mensaje de sistema en tiempo real al puerto de salida MIDI.

30.1.2 INICIALIZACIÓN

ifreq - frecuencia del mensaje de reloj en Herzios.

imsgtype - tipo de mensaje en tiempo real:

- 1 envía un mensaje de COMIENZO (0xFA);
- 2 envía un mensaje de CONTINUACIÓN (0xFB);
- 0 envía un mensaje de DETENCIÓN (0xFC);
- 1 envía un mensaje de REESTABLECIMIENTO DE SISTEMA (0xFF);
- 2 envía un mensaje de COMPROBACIÓN DE CONEXIÓN (0xFE).

30.1.3 EJECUCIÓN

`mclock` (MIDI clock, reloj MIDI) envía un mensaje de RELOJ MIDI (0xF8) cada $1/ifreq$ segundos. Así que *ifreq* es la frecuencia de mensaje de RELOJ en Herzios.

`mrtmsg` (mensaje de tiempo real) envía un mensaje de tiempo real una vez en la inicialización del instrumento actual. El parámetro *imsgtype* es un indicador del tipo de mensaje (ver arriba, en la descripción de los argumentos)

30.1.4 AUTOR

Gabriel Maldonado
Italia
Mayo 1997

31 SOPORTE MIDI: EXTENDEDORES DE EVENTO

31.1 xtratim, release

```
        xtratim iextradur
kflag  release
```

31.1.1 DESCRIPCIÓN

Prolonga la duración de eventos MIDI generados en tiempo real y se ocupa de su "vida extra" (ver también **linenr**).

31.1.2 INICIALIZACIÓN

iextradur - duración adicional del instrumento activo en ese momento.

31.1.3 EJECUCIÓN

xtratim prolonga la duración de la nota MIDI activada en ese momento en *iextradur* segundos, después de que el mensaje MIDI "note-off" haya desactivado dicha nota. Este opcode no tiene argumentos de salida.

release devuelve el estado actual de la nota. Si la nota se encuentra en la fase de caída (es decir, si su duración ha sido prolongada con **xtratim** y acaba de ser desactivada), el argumento de salida *kflag* se pone a 1, sino (es decir, si la nota esta en su estado de sostenimiento) se pone a 0. Estos dos opcodes son útiles para implementar envolventes dinámicos complejos en la caída de la nota.

31.1.4 EJEMPLO

```
instr 1 ; permite ADSRs complejos con eventos MIDI
inum  notnum
icps  cpsmidi
iamp  ampmidi 4000
;

;----- boque de envolvente complejo -----
        xtratim 1 ;duración extra, es decir, duración de la caída
krel  init 0
krel  release ;devuelve el indicador de caída(0=sostenimiento, 1=caída)
        if (krel .5) kgoto rel ;si está en fase de caída va a la sección de caída
;

;***** sección de ataque y sostenimiento *****
kmp1  linseg 0, .03, 1, .05, 1, .07, 0, .08, .5, 4, 1, 50, 1
kmp   = kmp1*iamp
        kgoto done
;
```

```
;----- sección de caída -----  
rel:  
kmp2 linseg 1, .3, .2, .7, 0  
kmp  = kmp1*kmp2*iamp  
done:  
;-----  
a1   oscili kmp, icps, 1  
      out a1
```

endin

31.1.5 AUTOR

Gabriel Maldonado

Italia

Mayo 1997

32 GENERADORES DE SEÑAL: GENERADORES LINEALES Y EXPONENCIALES

32.1 line, expon, linseg, linsegr, expseg, expsegr, expsega

kr	line	ia, idur1, ib
ar	line	ia, idur1, ib
kr	expon	ia, idur1, ib
ar	expon	ia, idur1, ib
kr	linseg	ia, idur1, ib[, idur2, ic[...]]
ar	linseg	ia, idur1, ib[, idur2, icI[...]]
kr	linsegr	ia, idur1, ib[, idur2, ic[...]], irel, iz
ar	linsegr	ia, idur1, ib[, idur2, icI[...]], irel, iz
kr	expseg	ia, idur1, ib[, idur2, ic[...]]
ar	expseg	ia, idur1, ib[, idur2, ic[...]]
kr	expsegr	ia, idur1, ib[, idur2, ic[...]], irel, iz
ar	expsegr	ia, idur1, ib[, idur2, ic[...]], irel, iz
ar	expsega	ia, idur1, ib[, idur2, ic[...]]

32.1.1 DESCRIPCIÓN

Los valores de salida *kr* o *ar* forman bien una línea recta (o una curva exponencial), bien una serie de segmentos lineales (o exponenciales) entre dos puntos dados.

32.1.2 INICIALIZACIÓN

ia- valor inicial. 0 es ilegal para las curvas exponenciales.

ib, *ic*, etc. - valor después de *dur1* segundos, etc. Para las curvas exponenciales debe ser distinto de 0 y concordar en signo con *ia*.

idur1 - duración en segundos del primer segmento. Un 0 ó un valor negativo hará que se omita la inicialización.

idur2, *idur3*, etc. - duración en segundos de los siguientes segmentos. Un valor 0 ó negativo terminará el proceso de inicialización con el punto anterior, permitiendo así a la última línea o curva definida prolongarse indefinidamente en la ejecución. El valor por defecto es 0.

irel, *iz* - duración en segundos y valor final del segmento de caída de una nota.

32.1.3 EJECUCIÓN

Estas unidades generan señales de control o de audio cuyas curvas (o rectas) pueden pasar por 2 o más puntos especificados. La suma de los valores *dur* puede o puede no ser igual a la duración de la ejecución del instrumento: una duración de la ejecución más corta que la de la curva causará un truncamiento de ésta, mientras que una más larga causará que el último segmento definido en la curva continúe en la misma dirección hasta agotar la duración total.

linsegr y **expsegr** se encuentran entre las unidades "r" de Csound que contienen un sensor de nota desactivada y un prolongador de la duración de la caída. Cuando perciben el fin de un evento MIDI o una desactivación de nota, inmediatamente extiende la duración de la ejecución de un determinado instrumento *irel* segundos y se programa para alcanzar el valor *iz* justo al final de ese período (sin importar en qué segmento se encuentre la unidad). Las unidades "r" pueden ser modificadas también por los mensajes MIDI de velocidad de nota desactivada (ver **veloffs**). Si hay dos o más prolongaciones en un mismo instrumento se optará por seguir la más larga.

expsega es casi idéntico a **expseg**, pero más preciso cuando se definen segmentos de duración muy corta (por ejemplo, en un ataque percusivo) a frecuencia de audio. Observa que **expseg** no opera correctamente a frecuencia de audio cuando los segmentos son más cortos que un ciclo de control. En esta situación, deberá usarse **expsega** en lugar de **expseg**.

32.1.4 EJEMPLO

```
k2 expseg 440, p3/2,880, p3/2,440
```

Esta sentencia crea una señal de control que se mueve exponencialmente, a lo largo de la duración especificada por p3, desde 440 a 880 y luego vuelve.

32.1.5 AUTOR

Gabriel Maldonado (**expsega**)

Italia

Junio, 1998

Nuevo en la versión 3.57

32.2 adsr, madsr, xadsr, mxadsr

kr	adsr	iatt, idec, islev, irel[, idel]
kr	madsr	iatt, idec, islev, irel[, idel]
kr	xadsr	iatt, idec, islev, irel[, idel]
kr	mxadsr	iatt, idec, islev, irel[, idel]
ar	adsr	iatt, idec, islev, irel[, idel]
ar	madsr	iatt, idec, islev, irel[, idel]
ar	xadsr	iatt, idec, islev, irel[, idel]
ar	mxadsr	iatt, idec, islev, irel[, idel]

32.2.1 DESCRIPCIÓN

Aplican un envolvente ADSR clásico (Ataque, Decaimiento, Sostenimiento, Caída).

32.2.2 INICIALIZACIÓN

iatt - duración del ataque.

idec - duración del decaimiento.

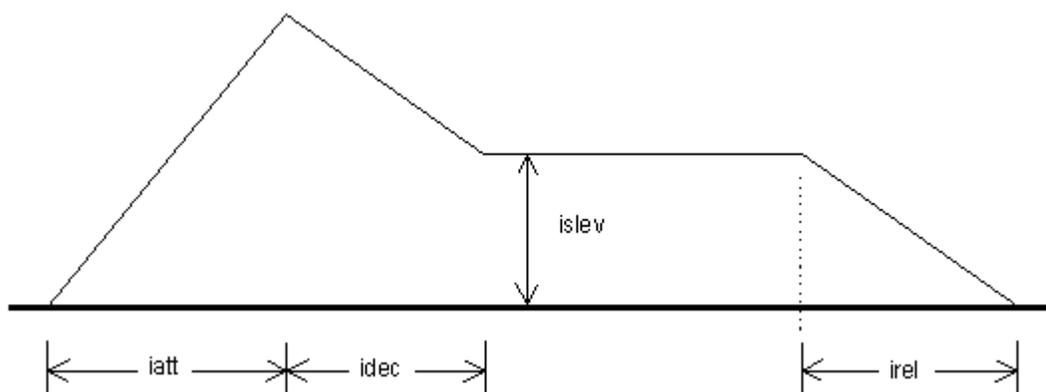
islev - nivel del período de sostenimiento.

irel - duración de la caída.

idel - período nulo antes de que comience el envolvente (es decir, el retardo del envolvente).

32.2.3 EJECUCIÓN

Los datos del envolvente están en el rango de 0 a 1 y pueden necesitar ser escalados posteriormente.



La duración del sostenimiento se calcula conforme a la longitud de la nota. Esto implica que **adsr** no viene bien para ser usado con eventos MIDI. El opcode **madsr** usa el mecanismo de **linsegr**, de manera que puede ser usado en algoritmos que trabajen con eventos MIDI. Los opcodes **xadsr** y **mxadsr** son idénticos a **adsr** y **madsr**, respectivamente, excepto en que usan segmentos exponenciales, no lineales.

33 GENERADORES DE SEÑAL: ACCESO A TABLAS

33.1 table, tablei, table3, oscill, oscilli, osciln

```
ir table      indx, ifn[, ixmode[, ixoff[, iwrap]]]
ir tablei    indx, ifn[, ixmode[, ixoff[, iwrap]]]
ir table3    indx, ifn[, ixmode[, ixoff[, iwrap]]]
kr table      kndx, ifn[, ixmode[, ixoff[, iwrap]]]
kr tablei    kndx, ifn[, ixmode[, ixoff[, iwrap]]]
kr table3    kndx, ifn[, ixmode[, ixoff[, iwrap]]]
ar table      andx, ifn[, ixmode[, ixoff[, iwrap]]]
ar tablei    andx, ifn[, ixmode[, ixoff[, iwrap]]]
ar table3    andx, ifn[, ixmode[, ixoff[, iwrap]]]
kr oscill    idel, kamp, idur, ifn
kr oscilli   idel, kamp, idur, ifn
ar osciln    kamp, ifrq, ifn, itimes
```

33.1.1 DESCRIPCIÓN

Se puede acceder a los valores de la tabla bien por indexación directa, bien por muestreo incremental.

33.1.2 INICIALIZACIÓN

ifn - número de la tabla de función. **tablei** y **oscilli** requieren un elemento límite añadido a la tabla (*extended guard point*).

ixmode (opcional) - modo *ndx*. El valor por defecto es 0.

- 0 significa un índice sin modificar.
- 1 significa un índice normalizado (entre 0 y 1).

ixoff (opcional) - elemento de la tabla en el que empieza a contar el índice (es decir, su desplazamiento u offset). Para una tabla con el origen en el centro, por ejemplo, se puede usar el valor $\text{tamaño_tabla}/2$ (para índices sin modificar) o .5 (para índices normalizados). El valor por defecto es 0.

iwrap (opcional) - indicador de bucle (modo cíclico) de *ndx*. El valor por defecto es 0.

- 0 significa que no hay bucle (los índices menores que 0 se tratan como iguales a 0 y no pueden pasar del tamaño de la tabla)
- 1 significa que hay bucle.

idel - retardo en segundos del muestreo incremental de **oscill**.

idur - duración en segundos que debe tardar **oscill** en hacer una única pasada por la tabla. Un valor 0 o negativo omitirá el proceso de inicialización.

ifrq, *itimes* - velocidad de muestreo y número de pasadas a través de la tabla.

33.1.3 EJECUCIÓN

table invoca un proceso de búsqueda en tabla según un índice (de tipo i-, k- o a-) dado. Estos índices pueden ser números enteros (0,1,2,...tamaño-1) o valores normalizados (de 0 a 1-e). Los índices son primero modificados por el offset (desplazamiento) y luego se comprueba que estén dentro del rango permitido antes de proceder a la búsqueda en la tabla (ver *wrap*). Si es probable que *ndx* alcance los máximos de normalización o se está usando interpolación, la tabla deberá tener un elemento límite añadido (*guard point*). Una unidad **table** indexada por un generador **phasor** periódico (ver **phasor**) se comportará como un oscilador.

oscil1 accede a los valores muestreando la tabla de función una vez a la velocidad determinada por *idur*. Para los primeros *idel* segundos, el puntero de escaneado empezará por el primer elemento de la tabla; entonces empezará a moverse hacia delante a través de la tabla a una velocidad constante, alcanzando el final a los *idur* segundos; desde ese momento en adelante (es decir, a partir de *idel + idur* segundos) permanecerá apuntando al último elemento de la tabla. Cada valor obtenido del sampleo es multiplicado por un factor de amplitud *kamp* antes de ser escrito en el resultado.

osciln muestreará la tabla varias veces a una frecuencia de *ifrq* Herzios, después de lo cual devolverá ceros. Sólo genera señales de audio, con valores de salida mutiplicados por el factor *kamp*.

tablei y **oscilli** son unidades que funcionan con interpolación, en las cuales la parte fraccional de *ndx* se usa para interpolar dos elementos consecutivos de la tabla. La suavidad que se obtiene por el proceso de interpolación se cobra algún tiempo de ejecución extra (ver también **oscili**, etc), pero se pueden intercambiar fácilmente las unidades que usan interpolación y las que no. Observa que cuando **tablei** usa un índice periódico, cuyo módulo *n* es menor que la potencia de 2 de la longitud de la tabla, el proceso de interpolación requiere que haya un elemento (*n + 1*)ésimo en la tabla, repetición del primero (ver **Sentencia f** de la partitura). **table3** es un opcode experimental idéntico a **tablei**, excepto en que usa interpolación cúbica. (Nuevo en la Versión 3.50)

34 GENERADORES DE SEÑAL: GENERADORES DE FASE

34.1 phasor

```
kr   phasor   kcps[ , iphs ]
ar   phasor   xcps[ , iphs ]
```

34.1.1 DESCRIPCIÓN

Produce un valor normalizado dinámico de fase (es decir, una fase normalizada que cambia de valor en el tiempo).

34.1.2 INICIALIZACIÓN

iphs (opcional) - fase inicial, expresada como la fracción de un ciclo (de 0 a 1). Un valor negativo causará que se omita la inicialización de la fase. El valor por defecto es 0.

34.1.3 EJECUCIÓN

Se va acumulando progresivamente una fase interna, según la frecuencia indicada por el argumento *cps*, para producir el efecto de un valor dinámico de fase, normalizado en el rango de 0 a *phs* (siendo *phs* siempre menor que 1).

Cuando se usa como índice de una unidad **table**, esta fase (multiplicada por la longitud de una tabla de función cualquiera) hará que dicha unidad se comporte como un oscilador.

Observa que **phasor** es un tipo especial de integrador que acumula los incrementos de fase sugeridos por la frecuencia.

34.1.4 EJEMPLO

```
kpch  phasor   1           ; recorre una vez por segundo
al    table   k1 * 12, 1   ; una tabla de 12 notas
al    oscil   p4, cpspch(kpch), 2 ; con sonido continuo
```

34.2 phasorbnk

```
kr   phasorbnk  kcps, kndx, icnt[, iphs]
ar   phasorbnk  xcps, kndx, icnt[, iphs]
```

34.2.1 DESCRIPCIÓN

Produce un número arbitrario de valores dinámicos de fase normalizados, accesibles mediante un índice.

34.2.2 INICIALIZACIÓN

icnt - máximo número de generadores de fase a ser usados.

iphs - fase inicial, expresada como una fracción de ciclo (de 0 a 1). Si es 1, se omitirá la inicialización. Si *iphs*>1 cada generador de fase será inicializado con un valor aleatorio.

34.2.3 EJECUCIÓN

kndx - valor del índice usado individualmente para acceder a los generadores de fase. Para cada generador de fase, una fase interna se va acumulando sucesivamente de acuerdo con la frecuencia *kcps* or *xcps* para producir un valor dinámico de fase, normalizado en el rango $0 \leq \text{phs} < 1$. Mediante el índice *kndx* se puede acceder a cada generador de fase individualmente.

Este banco de generadores de fase puede ser usado dentro de un bucle a frecuencia de control para generar voces múltiples independientes o juntas con el opcode **adsynt** para cambiar los parámetros de las tablas usadas por **adsynt**.

34.2.4 EJEMPLO

Genera múltiples voces con parciales independientes. Este ejemplo es mejor con **adsynt**. Ver también el ejemplo bajo **adsynt**, para el uso a frecuencia de control de **phasorbnk**.

```
giwave ftgen 1, 0, 1024, 10, 1 ; generate a sinewave table
instr 1
icnt      =          10          ; genera 10 voces
asum      =          0          ; vacía el buffer de salida
kindex    =          0          ; resetea el índice del bucle
loop:     ; bucle ejecutado
          ; cada ciclo de control
kcps      =          (kindex+1)*100 + 30 ; parciales no armónicos
aphas     phasorbnk kcps, kindex, icnt ; consigue la fase de cada voz
asig      table aphas, giwave, 1 ; y lee la onda de la tabla
asum      =          asum + asig ; acumula la salida
kindex    =          kindex + 1
          if (kindex < icnt) kgoto loop ; va al bucle
          out asum*3000
endin
```

34.2.5 AUTOR

Peter Neubäcker

Munich, Alemania

Agosto, 1999

Nuevo en la Versión 3.58

35 GENERADORES DE SEÑAL: OSCILADORES BÁSICOS

35.1 oscil, oscili, oscil3

```
kr oscil    kamp, kcps, ifn[, iphs]
kr oscili   kamp, kcps, ifn[, iphs]
kr oscil3   kamp, kcps, ifn[, iphs]
ar oscil    xamp, xcps, ifn[, iphs]
ar oscili   xamp, xcps, ifn[, iphs]
ar oscil3   xamp, xcps, ifn[, iphs]
```

35.1.1 DESCRIPCIÓN

La tabla *ifn* es muestreada incremental y cíclicamente en toda su longitud y los valores obtenidos son multiplicados por el factor *amp*.

35.1.2 INICIALIZACIÓN

ifn - número de la tabla de función. Requiere el elemento límite añadido.

iphs (opcional) - fase inicial del muestreo expresada como una fracción de ciclo (de 0 a 1). Un valor negativo omitirá la inicialización de la fase. El valor por defecto es 0.

35.1.3 EJECUCIÓN

Las unidades **oscil** producen señales periódicas de control (k-) o de audio (a-), que consisten en el valor devuelto por el muestreo, realizado a una frecuencia de control (k-) o de audio (a-), de una tabla de función almacenada, multiplicado por un factor *kamp* (*xamp*). La fase interna avanza simultáneamente según el valor del argumento de entrada *cps*. Mientras que los argumentos de entrada de amplitud y de frecuencia de la unidad **oscil** que trabaja con señales de control (k-) son únicamente escalares, los mismos argumentos del **oscil** que trabaja con señales de audio (a-) pueden ser tanto escalares como vectores, permitiendo así la modulación de frecuencia o amplitud tanto a frecuencias de control (k-) como de audio (a-).

oscili y **foscili** se diferencian de **oscil** y **foscil** respectivamente, en que el procedimiento estándar de usar una fase truncada como índice se reemplaza por un proceso de interpolación entre dos valores sucesivos de la tabla. Los generadores que usan interpolación producirán una señal mucho más limpia, pero también pueden tardar el doble en compilarse. Otro medio de conseguir más exactitud, pero sin coste de tiempo, es sustituir la interpolación por el uso de tablas más grandes de 2K, 4K u 8K si se dispone de memoria suficiente. **oscil3** es un opcode experimental, idéntico a **oscili**, excepto en que usa interpolación cúbica (Nuevo en la Versión 3.50).

35.1.4 EJEMPLO

```
k1 oscil 10, 5, 1 ; vibrato de 5 Hz  
a1 oscil 5000, 440 + k1, 1 ; sobre un la 440 + -10 Hz
```

35.2 poscil, poscil3

ar	poscil	kamp, kcps, ifn [, iphs]
kr	poscil	kamp, kcps, ifn [, iphs]
ar	poscil3	kamp, kcps, ifn [, iphs]
kr	poscil3	kamp, kcps, ifn [, iphs]

35.2.1 DESCRIPCIÓN

Osciladores de alta precisión. **poscil3** usa interpolación cúbica.

35.2.2 INICIALIZACIÓN

ifn - número de la tabla de función.

iphs (opcional) - fase inicial (epresada en muestras).

35.2.3 EJECUCIÓN

ar - señal de salida.

kamp - amplitud.

kcps - frecuencia.

kfregratio - factor de multiplicación de la tabla de frecuencia (por ejemplo: 1 = frecuencia original, 1.5 = una quinta arriba, .5 = una octava abajo).

kloop - punto inicial del bucle (expresado en muestras).

kend - punto final del bucle (expresado en muestras)

poscil (oscilador preciso) es idéntico a **oscili** pero permite un control mucho más preciso sobre la frecuencia, especialmente cuando se usan tablas largas y valores de baja frecuencia. Su método de indexación en tabla usa índices en coma flotante, en lugar de índices enteros como los usados por **oscil** y **oscili**. Es tan sólo un poco más lento que **oscili**.

35.2.4 AUTORES

Gabriel Maldonado (**poscil**)
Italia
1998 (Nuevo en la versión 3.52)

John ffitch (**poscil3**)
Universidad de Bath/Codemist Ltd. Bath, Reino Unido
Febrero, 1999 (Nuevo en la versión 3.52)

35.3 lfo

```
kr    lfo    kamp, kcps[, itype]  
ar    lfo    kamp, kcps[, itype]
```

35.3.1 DESCRIPCIÓN

Oscilador de baja frecuencia (LFO) con varias formas de onda.

35.3.2 INICIALIZACIÓN

itype -- determina la forma de onda del oscilador. El valor por defecto es 0.

0: sinusoidal
1: triangular
2: cuadrada (bipolar)
3: cuadrada (unipolar)
4: diente de sierra
5: diente de sierra (invertida)

La onda sinusoidal se implementa como una tabla de 4096 elementos con interpolación lineal. Las otras son calculadas.

35.3.3 EJECUCIÓN

kamp – amplitud de la salida.

kcps - frecuencia del oscilador.

35.3.4 EJEMPLO

```
instr 1  
kp    lfo    10, 5, 4  
ar    oscil  p4, p5+kp, 1  
      out    ar  
endin
```

35.3.5 AUTOR

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Noviembre, 1998 (Nuevo en la versión 3.491)

36 GENERADORES DE SEÑAL: OSCILADORES DE ESPECTRO DINÁMICO

36.1 buzz, gbuzz

```
ar buzz      xamp, xcps, knh, ifn[, iphs]  
ar gbuzz     xamp, xcps, knh, klh, kr, ifn[, iphs]
```

36.1.1 DESCRIPCIÓN

La salida es una serie de parciales coseno armónicamente relacionados.

36.1.2 INICIALIZACIÓN

ifn - número de una tabla de función que contiene una onda sinusoidal (para **buzz**) o una onda cosinusoidal (para **gbuzz**). En cualquiera de los casos se recomienda usar una tabla grande, de al menos 8192 elementos.

iphs (opcional) - fase inicial de la frecuencia fundamental, expresada como fracción de un ciclo (de 0 a 1). Un valor negativo omitirá la inicialización. El valor por defecto es 0.

36.1.3 EJECUCIÓN

El opcode **buzz** genera una serie aditiva de parciales coseno armónicamente relacionados, con una frecuencia fundamental *xcps*, y cuyas amplitudes se normalizan de tal manera que el valor de pico sea igual *xamp*. La selección y la fuerza de los parciales viene determinada por los siguientes parámetros:

knh - número total de armónicos requeridos. A partir de la versión 3.57, *knh* tiene un valor por defecto de 1. Si *knh* es negativo, se usará su valor absoluto.

klh - armónico más bajo presente. Puede ser positivo, 0 o negativo. En **gbuzz**, la serie de parciales puede empezar en cualquier número de parcial para proceder luego hacia arriba; si *klh* es negativo, todos los parciales por debajo de 0 se reflejarán como positivos sin cambio de fase (ya que la función coseno es una función uniforme) y se añadirán a todos los demás parciales positivos presentes.

kr - especifica el factor múltiplo en la serie de coeficientes de amplitud. Es una serie exponencial: si el parcial *klh*ésimo tiene un coeficiente de amplitud *A*, el (*klh* + *n*)ésimo parcial tendrá un coeficiente de $A * (kr \wedge n)$, es decir, los valores de amplitud trazan una curva exponencial. *kr* puede ser positivo, 0 o negativo, y no tiene porqué ser un valor entero.

buzz y **gbuzz** son útiles para obtener sonidos complejos que pueden usarse como fuente en síntesis substractiva. **buzz** es un caso especial de **gbuzz**, en el que se cumple que $klh = kr = 1$, produciendo así una serie de *knh* parciales armónicos de igual fuerza, empezando por la fundamental. Esto es idéntico a un tren de ondas pulso con un ancho de banda limitado; si los parciales sobrepasan la frecuencia Nyquist, es decir, si *knh* es igual a la parte entera de " $sr / 2 /$ frecuencia fundamental", entonces el resultado es un tren de

ondas pulso real con una amplitud *xamp*. Aunque ambos *knh* y *klh* pueden ser variados durante la ejecución, sus valores internos son necesariamente enteros y pueden causar "pops" (esto es, ruiditos) debido a las discontinuidades de la salida. *kr*, en cambio, puede ser variado durante la ejecución obteniendo buenos resultados. Ambos, **buzz** y **gbuzz**, pueden ser modulados, en frecuencia o amplitud, por cualquier señal de control o de audio.

N.B: estas dos unidades tienen sus análogas en **GEN11**, donde la misma serie de cosenos puede ser almacenada en una tabla de función para samplearla luego con un oscilador. Aunque computacionalmente más eficientes, las tablas que almacenan un tren de ondas pulso tienen un contenido espectral fijo, no variable como proporcionan estos opcodes.

36.2 vco

ar vco kamp, kfqc, iwave, kpw, ifn, imaxd

36.2.1 DESCRIPCIÓN

Implementación de un oscilador analógico de banda limitada, basada en la integración de impulsos de banda limitada. **vco** puede ser usado para simular formas de onda conseguidas por medios analógicos.

36.2.2 INICIALIZACIÓN

iwave – determina la forma de onda:

- 1: diente de sierra
- 2: cuadrada
- 3: triangular

36.2.3 EJECUCIÓN

kamp – determina la amplitud.

kfqc – frecuencia de la onda.

kpw – cuando *iwave* es 2, determina la anchura del pulso; cuando *iwave* es 3, determina el carácter de la "rampa de la sierra". El valor de *kpw* debería estar entre 0 y 1. Un valor de .5 generará una onda cuadrada o triangular dependiendo de *iwave*.

ifn – debe ser el número de una tabla que contenga una onda sinusoidal.

imaxd – tiempo de retardo máximo. Para las ondas cuadrada y triangular puede requerirse un valor de $1/ikfqc$. Para disminuir la frecuencia, este valor debe ser tan grande como "1/(frecuencia mínima)".

36.2.4 EJEMPLO

```
instr 10
idur = p3           ; Duración
iamp = p4           ; Amplitud
ifqc = cpspch(p5)  ; Frecuencia
iwave = p6         ; forma de onda 1=Saw, 2=Square/PWM, 3=Tri/Saw-Ramp-Mod
isine = 1
imaxd = 1/ifqc*2   ; permite deslizamientos de altura de hasta dos octavas
                  ; descendentes

kpw1 oscil .25, ifqc/200, 1
kpw = kpw1 + .5
asig vco iamp, ifqc, iwave, kpw, 1, imaxd
outs asig, asig    ; salida y amplificación
endin
```

```
f1 0 65536 10 1
; Sta Dur Amp Pitch Wave
i10 0 2 20000 5.00 1
i10 + . . . 2
```

;	Sta	Dur	Amp	Pitch	Wave
i10	3
i10	.	2	20000	7.00	1
i10	2
i10	3
i10	.	2	20000	9.00	1
i10	2
i10	3
i10	.	2	20000	11.00	1
i10	2
i10	3

e

36.2.5 AUTOR

Hans Mikelson
Diciembre, 1998
Nuevo en la Versión 3.50

37 GENERADORES DE SEÑAL: SÍNTESIS Y RESÍNTESIS ADITIVA

37.1 adsyn

ar **adsyn** kamod, kfmod, ksmod, ifilcod

37.1.1 DESCRIPCIÓN

La salida es una serie aditiva de ondas sinusoidales controladas individualmente por un banco de osciladores.

37.1.2 INICIALIZACIÓN

ifilcod - entero o cadena de caracteres que indican un fichero de control derivado del análisis de una señal de audio. Un entero indica la extensión de un fichero *adsyn.m* o *pvoc.m*; una cadena de caracteres (entre comillas) proporciona el nombre de un fichero, opcionalmente con la ruta completa. Si no se especifica la ruta completa, el fichero se busca primero en el directorio actual y luego en el que indique la variable de entorno **SADIR** (si está definida). **adsyn** requiere los valores de los puntos de inflexión de los envolventes de amplitud y frecuencia, organizados para su resíntesis con los osciladores. El uso de la memoria depende del tamaño de los ficheros involucrados, que son leídos y mantenidos en memoria durante el proceso de computación, aunque compartidos por múltiples llamadas.

37.1.3 EJECUCIÓN

adsyn resintetiza timbres complejos, que varían en el tiempo, mediante la técnica de la síntesis aditiva. Cualquier número de ondas sinusoidales, cada una controlada individualmente en frecuencia y amplitud, pueden ser sumadas mediante aritmética de alta velocidad para producir un resultado de alta fidelidad.

Las ondas sinusoidales que componen el sonido son descritas en un fichero de control que indica la evolución temporal, en milisegundos, de sus amplitudes y frecuencias, señalando los puntos de inflexión de los envolventes. Las pistas que contienen los datos vienen definidas por secuencias de enteros de 16 bits:

- 1, tiempo, amplitud, tiempo, amplitud,...
- 2, tiempo, frecuencia, tiempo, frecuencia,...

tal y como los presenta el fichero de análisis por filtrado heterodino de cualquier fichero de audio (para los detalles, ver **hetro**). Los valores instantáneos de las amplitudes y las frecuencias son usados por un oscilador interno fijo que suma cada parcial activo a una señal de salida que se va acumulando. Aunque hay un límite práctico para el número de parciales (que desapareció en la Versión 3.47), no hay ninguna restricción en su evolución en el tiempo. Cualquier sonido que pueda ser descrito en términos del comportamiento de sus parciales sinusoidales, puede ser sintetizado por el opcode **adsyn** por sí solo.

El sonido descrito por un fichero de control de **adsyn** también puede ser modificado durante el proceso de resíntesis. Las señales *kamod*, *kfmod*, y *ksmod* modificarán, respectivamente, la amplitud, la frecuencia y la velocidad de los parciales constitutivos del sonido. Estas señales son multiplicadas, con *kfmod* modificando la frecuencia en Hz y *ksmod* la velocidad en milisegundos a la que son leídos los segmentos lineales. Así *kamod* = .7, *kfmod* = 1.5 y *ksmod* = 2 darán lugar a un sonido más suave, una quinta justa más alto y con la mitad de duración. Los valores 1,1,1 dejarán el sonido tal cual. Cada uno de estos argumentos puede ser una señal de control (k-).

Kfmod es un factor de transposición a frecuencia de control. Un valor de 1 significa que no hay transposición, 1.5 transporta una quinta justa ascendente y .5 una octava descendente.


```

instr 1                                ; genera parámetros en la inic.
icnt = 10                                ; genera 10 voces
index = 0                                 ; incializa el índice del bucle
loop:                                     ; sólo se ejecuta el bucle
                                           ; en la inicialización
ifreq pow index + 1, 1.5                 ; define los parciales
                                           ; no armónicos
iamp = 1 / (index+1)                       ; define amplitudes
tableiw ifreq, index, gifrqs             ; escribe a las tablas
tableiw iamp, index, giamps             ; usadas por adsynt
index = index + 1
if (index < icnt) igoto loop             ; va al bucle
asig adsynt 5000, 150, giwave, gifrqs, giamps, icnt
out asig
endin
instr 2                                ; genera los parámetros
                                           ; cada ciclo de control
icnt = 10                                ; genera 10 voces
kindex = 0                                ; resetea el índice del bucle
loop:                                     ; ejecuta el bucle cada
                                           ; ciclo de control
kspeed pow kindex + 1, 1.6               ; genera un lfo para
                                           ; las frecuencias
kphas phasorbnk kspeed * 0.7, kindex, icnt ; fase individual para cada voz
klfo table kphas, giwave, 1
kdepth pow 1.4, kindex                   ; rotación arbitraria
                                           ; de parámetros
kfreq pow kindex + 1, 1.5
kfreq = kfreq + klfo*0.006*kdepth
tablew kfreq, kindex, gifrqs             ; escribe frecs. a la tabla
                                           ; para adsynt
kspeed pow kindex + 1, 0.8
kphas phasorbnk kspeed*0.13, kindex, icnt, 2 ; fases individuales para
                                           ; cada voz
klfo table kphas, giwave, 1
kamp pow 1 / (kindex + 1), 0.4           ; rotación arbitraria
                                           ; de parámetros
kamp = kamp * (0.3+0.35*(klfo+1))
tablew kamp, kindex, giamps             ; escribe las amplitudes en
                                           ; la tabla para adsynt
kindex = kindex + 1
if (kindex < icnt) kgoto loop            ; va al bucle
giwave ftgen 1, 0, 1024, 10, 1          ; genera una onda sinusoidal
asig adsynt 5000, 150, giwave, gifrqs, giamps, icnt
out asig
endin

```

37.2.5 AUTOR

Peter Neubäcker
Munich, Alemania
Agosto, 1999
Nuevo en la Versión 3.58

37.3 hsboscil

```
ar    hsboscil    kamp, ktone, kbrite, ibasfreq, iwfn, ioctfn \\  
      [, ioctcnt[, iphs]]
```

37.3.1 DESCRIPCIÓN

Oscilador con dos argumentos, tonalidad y brillo, relativos a una frecuencia base.

37.3.2 INICIALIZACIÓN

ibasfreq – frecuencia base de referencia para la tonalidad y el brillo.

iwfn – tabla de función de una forma de onda, normalmente sinusoidal.

ioctfn – tabla de función usada para ponderar las octavas, normalmente algo como:

```
f1 0 1024 -19 1 0.5 270 0.5
```

ioctcnt – número de octavas usadas para la mezcla de brillo. Debe estar en el rango de 2 a 10. El valor por defecto es 3.

iphs – fase inicial del oscilador. Si *iphs* = -1, se omite la inicialización.

37.3.3 EJECUCIÓN

kamp – amplitud de la nota

ktone – parámetro cíclico de tonalidad relativo a *ibasfreq* en una octava logarítmica, rango de 0 a 1. Pueden usarse valores mayores que 1, que serán internamente reducidos a **frac** (*ktone*).

kbrite – parámetro para el brillo relativo a *ibasfreq*, conseguido ponderando *ioctcnt* octavas. Se escala de tal manera que un valor 0 corresponde al valor original de *ibasfreq*, 1 corresponde a una octava por encima de *ibasfreq*, -2 corresponde a dos octavas por debajo de *ibasfreq*, etc... *kbrite* puede ser fraccional.

hsboscil. toma tonalidad y brillo como parámetros relativos a una frecuencia base (*ibasfreq*). La tonalidad es un parámetro cíclico en la octava logarítmica. El brillo se calcula mezclando múltiples octavas ponderadas. Es útil cuando se comprende el espacio tonal como un concepto con coordenadas polares. Si *ktone* fuera una línea recta, y *kbrite* una constante, produciría el glissando de Risset. La tabla *iwfn* se lee siempre con interpolación. El tiempo de ejecución es aproximadamente *ioctcnt* * **oscili**.

37.3.4 EJEMPLO

```
giwave      ftgen      1, 0, 1024, 10, 1, 1, 1, 1 ; onda sinusoidal
giblend     ftgen      2, 0, 1024, -19, 1, 0.5, 270, 0.5 ; ventana de mezcla
instr 1                                           ; glissando infinito
ktona       line       0,10,1
asig        hsboscil   10000, ktona, 0, 200, giwave, giblend, 5
out
asig
endin
instr 2                                           ; instrumento MIDI: todas las octavas
                                           ; suenan parecidas
                                           ; la velocidad se asocia con el brillo
itona       octmidi
ibrite      ampmidi   3
ibase      =      cpsoct(6)
kenv       expon     20000, 1, 100
asig       hsboscil   kenv, itona, ibrite, ibase, giwave, giblend, 5
out
asig
endin
```

37.3.5 AUTOR

Peter Neubäcker
Munich, Alemania
Agosto, 1999
Nuevo en la Versión 3.58

38 GENERADORES DE SEÑAL: SÍNTESIS FM

38.1 foscil, foscili

```
ar foscil  xamp, kcps, xcar, xmod, kndx, ifn[, iphs]  
ar foscili xamp, kcps, xcar, xmod, kndx, ifn[, iphs]
```

38.1.1 DESCRIPCIÓN

Osciladores de frecuencia modulada básicos.

38.1.2 INICIALIZACIÓN

ifn - número de la tabla de función. Requiere el elemento límite añadido.

iphs (opcional) - fase inicial del muestreo expresada como una fracción de ciclo (de 0 a 1). Un valor negativo omitirá la inicialización de la fase. El valor por defecto es 0.

38.1.3 EJECUCIÓN

foscil es una unidad compuesta que implementa eficientemente dos osciladores según la conocida configuración FM de Chowning, en la que la salida de audio de un generador ("modulador") se usa para modular la frecuencia de otro ("portador"). La frecuencia real de la onda portadora es $kcps * kcar$, mientras que la de la onda moduladora es $kcps * xmod$. Para valores enteros de *xcar* y *xmod*, la fundamental percibida es el valor mínimo positivo de $kcps * (xcar - n * xmod)$, donde $n = 1, 1, 2, \dots$. El argumento *kndx* es el índice de modulación (normalmente varía en el tiempo entre valores de 0 a 4 aproximadamente) que determina la dispersión de la energía acústica en los parciales indicados por $n = 0, 1, 2, \dots$, etc. *ifn* debe apuntar a una tabla que contenga una función sinusoidal. Antes de la versión 3.50, *xcar* y *xmod* sólo podían ser valores k-.

38.2 fmvoice

```
ar    fmvoice    kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate, ifn1, \  
ifn2, ifn3, ifn4, ivibfn
```

38.2.1 DESCRIPCIÓN

Síntesis de canto vocal mediante FM.

38.2.2 INICIALIZACIÓN

ifn1, ifn2, ifn3, ifn3 -- tablas usadas, normalmente de ondas sinusoidales.

38.2.3 EJECUCIÓN

kamp -- amplitud de la nota.

kfreq -- frecuencia de la nota.

kvowel -- vocal que se va a cantar, en el rango de 0 a 64.

ktilt -- ajuste espectral del sonido en el rango de 0 a 99.

kvibamt -- profundidad del vibrato.

kvibrate -- frecuencia del vibrato.

38.2.4 EJEMPLO

```
k1    line        0, p3, 64  
a1    fmvoice    31129.60, 110, k1, 0, 0.005, 6, 1,1,1,1,1
```

38.2.5 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

38.3 fmbell, fmrhode, fmwurlie, fmmetal, fmb3, fmpercfl

a1	fmbell	kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \\ ifn3, ifn4, ivfn
a1	fmrhode	kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \\ ifn3, ifn4, ivfn
a1	fmwurlie	kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \\ ifn3, ifn4, ivfn
a1	fmmetal	kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \\ ifn3, ifn4, ivfn
a1	fmb3	kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \\ ifn3, ifn4, ivfn
a1	fmpercfl	kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, \\ ifn3, ifn4, ivfn

38.3.1 DESCRIPCIÓN

Son una familia de sonidos FM, usando todos 4 osciladores básicos pero implementando varias arquitecturas, tal y como se usan en el sintetizador TX81Z.

38.3.2 INICIALIZACIÓN

Todos estos opcodes requieren 5 tablas para su inicialización. Las primeras 4 son las entradas básicas y la última es el oscilador de baja frecuencia (LFO) que se usa para el vibrato. Normalmente esta última tabla será una onda sinusoidal.

Los valores iniciales de las tablas deben ser:

	ifn1	ifn2	ifn3	ifn4
fmbell	sinewave	sinewave	sinewave	sinewave
fmrhode	sinewave	sinewave	sinewave	fwavblnk
fmwurlie	sinewave	sinewave	sinewave	fwavblnk
fmmetal	sinewave	twopeaks	twopeaks	sinewave
fmb3	sinewave	sinewave	sinewave	sinewave
fmpercfl	sinewave	sinewave	sinewave	sinewave

donde *sinewave* es una onda sinusoidal, *twopeaks* es una de dos picos y *fwavblnk* es una tabla en blanco.

Los sonidos producidos son:

fmbell	Campana Tubular
fmrhode	Piano Eléctrico Rhodes Fender
fmwurlie	Piano Eléctrico Wurlitzer
fmmetal	"Heavy Metal"
fmb3	Órgano Hammond B3
fmpercfl	Flauta Percusiva

38.3.3 EJECUCIÓN

kamp - amplitud de la nota.

kgreq - frecuencia de la nota.

kc1, *kc2* - controles para el sintetizador, según la siguiente tabla:

<i>kc1</i>	<i>kc2</i>	<i>Algoritmo</i>
fm_{bell}	índice de modulación 1	Crossfade de dos salidas 5
fm_{rhode}	índice de modulación 1	Crossfade de dos salidas 5
fm_{wurlie}	índice de modulación 1	Crossfade de dos salidas 5
fm_{metal}	índice de modulación total	Crossfade de dos moduladores 3
fm_{b3}	índice de modulación total	Crossfade de dos moduladores 4
fm_{percfl}	índice de modulación total	Crossfade de dos moduladores 4

kvdepth - profundidad del vibrato.

kvrate - frecuencia del vibrato.

38.3.4 AUTOR

John ffitc
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

39 GENERADORES DE SEÑAL: REPRODUCCIÓN DE MUESTRAS

39.1 loscil, loscil3

```
ar[ ,ar2]    loscil    xamp, kcps, ifn[, ibas[,imod1,ibeg1,iend1 \\  
              [, imod2,ibeg2,iend2]]]  
ar[ ,ar2]    loscil3   xamp, kcps, ifn[, ibas[,imod1,ibeg1,iend1 \\  
              [, imod2,ibeg2,iend2]]]
```

39.1.1 DESCRIPCIÓN

Lee el sonido sampleado (mono o estéreo) almacenado en una tabla, con bucles opcionales en el sostenimiento y en la caída.

39.1.2 INICIALIZACIÓN

ifn - número de la tabla de función, normalmente indicando un segmento del sonido sampleado en formato AIFF, con los puntos de bucle configurados. El fichero fuente puede ser mono o estéreo.

ibas (opcional) - frecuencia base, en Hz, del sonido grabado. Opcionalmente, este valor puede reemplazar a la frecuencia proporcionada por el fichero AIFF, pero es necesario siempre que el fichero no especifique una. El valor por defecto es 0 (sin reemplazar).

imod1, *imod2* (opcional) - ejecuta de modos distintos los bucles de sostenimiento y caída. Un valor 1 indica un bucle normal; 2 indica un bucle de ida y vuelta (hacia delante y hacia atrás); 0 indica que no se produzca bucle alguno. El valor por defecto (-1) indica que se tomarán del fichero fuente los puntos de bucle y su modo de ejecución.

ibeg1, *iend1*, *ibeg2*, *iend2* (opcional, dependiente de *mod1*, *mod2*) - puntos de comienzo y final del los bucles de sostenimiento y caída. Se miden en *sample frames* desde el principio del fichero, así que serán los mismos sin importar que el segmento de sonido sea mono o estéreo.

39.1.3 EJECUCIÓN

loscil muestrea la tabla de audio a una frecuencia indicada por *kcps* y luego multiplica el resultado por *xamp*. El incremento del puntero de muestreo para cada *kcps* depende de la frecuencia base de la tabla *ibas*, y se configura automáticamente si el valor **sr** de la cabecera de la orquesta difiere de aquel en que fue grabada la fuente. En esta unidad, la tabla de función es muestreada siempre con interpolación.

Si el muestreo alcanza el punto final del bucle de sostenimiento, y el modo de bucle está activado, el puntero de muestreo será modificado y **loscil** continuará leyendo dentro del segmento señalado por el bucle. Una vez que el instrumento reciba una señal de desactivación (desde la partitura o desde un mensaje MIDI de nota desactivada), el siguiente punto final del bucle de sostenimiento que se encuentre será ignorado y el muestreo continuará hacia el punto final del bucle de caída o hasta la última muestra.

loscil es la unidad básica para construir un sampler. Teniendo, por ejemplo, suficientes muestras de varias notas de un piano, esta unidad se encargará de resamplearlas para simular las notas que faltan. La localización de la fuente de sonido con la altura más próxima a la nota deseada puede realizarse mediante búsqueda en tablas. Una vez que el muestreo del instrumento ha comenzado, su desactivación puede ser impredecible y requerir un envolvente de caída externo, lo que se consigue pasando el sonido muestreado por una unidad **linenr**, que se encargará, una vez que el instrumento haya sido desactivado, de prolongar su duración el tiempo deseado, al mismo tiempo que aplica un envolvente dinámico de caída.

loscil3 es un opcode experimental. Es idéntico a **loscil**, excepto en que usa interpolación cúbica. Nuevo en la versión 3.50.

39.1.4 EJEMPLO

```
inum notnum
icps cpsmidi
iamp ampmidi 3000, 1
ifno table inum, 2 ;número de nota para escoger una muestra de audio
ibas table inum, 3
kamp linenr iamp, 0, .05, .01 ;prolonga la nota 50 ms. después de la desactivación
asig loscil kamp, icps, ifno, cpsoct(ibas/12. + 3)
```

39.2 **lposcil**, **lposcil3**

```
ar    lposcil    kamp, kfregratio, kloop, kend, ifn [,iphs]  
ar    lposcil3  kamp, kfregratio, kloop, kend, ifn [,iphs]
```

39.2.1 DESCRIPCIÓN

Osciladores de alta precisión. **lposcil3** usa interpolación cúbica.

39.2.2 INICIALIZACIÓN

ifn - número de la tabla de función.

iphs (opcional) - fase inicial (expresada en muestras).

39.2.3 EJECUCIÓN

ar - señal de salida.

kamp - amplitud.

kcps - frecuencia.

kfregratio - factor de multiplicación de la tabla de frecuencia (por ejemplo: 1 = frecuencia original, 1.5 = una quinta arriba, .5 = una octava abajo).

kloop - punto inicial del bucle (expresado en muestras).

kend - punto final del bucle (expresado en muestras)

lposcil (oscilador preciso en bucle) permite variar a frecuencia de control los puntos de inicio y final del bucle de una muestra almacenada en una tabla (ver **GEN01**). Esto puede ser útil cuando se quiera leer el bucle de una tabla de onda sampleada previamente, en la cual la velocidad de repetición del bucle puede ser variada durante su ejecución.

39.2.4 AUTORES

Gabriel Maldonado (**lposcil**)
Italia
1998 (Nuevo en la versión 3.52)

John ffitch (**lposcil3**)
Universidad de Bath/Codemist Ltd. Bath, Reino Unido
Febrero, 1999 (Nuevo en la versión 3.52)

40 GENERADORES DE SEÑAL: SÍNTESIS GRANULAR

40.1 fof, fof2

```
ar   fof      xamp, xfund, xform, koct, kband, kris, kdur, kdec, \\
      iolaps, ifna, ifnb, itotdur[, iphs[, ifmode]]
ar   fof2     xamp, xfund, xform, koct, kband, kris, kdur, kdec, \\
      iolaps, ifna, ifnb, itotdur, kphs, kgliss
```

40.1.1 DESCRIPCIÓN

El audio de salida es una sucesión de "explosiones" sinusoidales, iniciadas a una frecuencia *xfund*, con un pico espectral en *xform*. Para *xfund* mayores que 25 Hz los estallidos producen un formante parecido a los de la voz humana, con características determinadas por los parámetros k- de entrada. Para fundamentales por debajo de ese valor, este generador proporciona una técnica especial de síntesis granular.

fof2 implementa la indexación incremental, a una determinada frecuencia de control (k-), de una tabla de función *ifna* para cada explosión sucesiva.

40.1.2 INICIALIZACIÓN

iolaps - número de espacios preasignados, necesarios para alojar los datos del solapamiento de las explosiones. El solapamiento es dependiente de la frecuencia y el espacio requerido depende del valor máximo de $xfund * kdur$. Puede ser sobreestimado sin coste adicional de tiempo de ejecución. Se usan menos de 50 bytes de memoria para cada *iolap*.

ifna, ifnb- números de las tablas de 2 funciones almacenadas. La primera es una función sinusoidal para sintetizar explosiones sinusoidales (se recomienda un tamaño de al menos 4096). La segunda es una función de ataque, usada hacia adelante y hacia atrás para dibujar el ataque y la caída de las explosiones sinusoidales. Puede ser lineal (**GEN07**) o sinusoidal (**GEN19**).

itotdur - tiempo total durante el cual **fof** estará activo. Normalmente concuerda con el campo p3 de la partitura. No se crean nuevas explosiones sinusoidales si no se pueden completar sus *kdur* dentro de la *itotdur* restante.

iphs (opcional) - fase inicial de la fundamental, expresada como una fracción de ciclo (de 0 a 1). El valor por defecto es 0.

ifmode (opcional) - modo de frecuencia de formantes. Si es 0, cada explosión sinusoidal mantiene la frecuencia *xform* con la que fue lanzado. Si no es 0, cada una de ellas será influenciada por *xform* constantemente. El valor por defecto es 0.

40.1.3 EJECUCIÓN

xamp - amplitud de pico de cada explosión sinusoidal, medida al final de su patrón de ataque. Dicho ataque puede exceder este valor si se da un ancho de banda grande (por lo menos, $Q < 10$) y/o cuando las explosiones se solapan.

xfund - frecuencia fundamental (en Hz) de los impulsos que crean nuevas explosiones sinusoidales.

xform - frecuencia del formante, es decir, la frecuencia de la explosión sinusoidal inducida por cada impulso *xfund*. Esta frecuencia puede ser fija para cada explosión o variar continuamente (ver *ifmode*).

koct - índice de "octavación", normalmente 0. Si es mayor que 0, disminuye la frecuencia efectiva de *xfund* atenuando las explosiones sinusoidales de número impar. Los números enteros representan octavas completas, siendo los intervalos intermedios representados por fracciones (decimales).

kband - ancho de banda del formante (a -6dB), expresado en Hz. El ancho de banda determina la velocidad de la caída exponencial a lo largo de la explosión sinusoidal, antes de que se aplique el envolvente descrito abajo.

kris, *kdur*, *kdec* - tiempo, en segundos, del ataque, la duración total y la caída de la explosión sinusoidal, respectivamente. Estos valores aplican un envolvente a cada explosión, de un modo similar a como lo hace el generador **linen**, pero con patrones de ataque y caída derivados del argumento *ifnb*. A la inversa, *kris* determina el ancho de la base (a -40 dB) de la región del formante inducido. *kdur* afecta a la densidad del solapamiento de las explosiones sinusoidales, y, de esta forma, también a la velocidad del proceso de computación. Los valores típicos para la imitación de la voz humana son .003, .02 y .007. En la implementación de **fof2**, *kphs* permite la indexación, en frecuencia de control (k-), de la tabla de función *ifna* con cada explosión sucesiva, siendo esto apropiado para aplicar repeticiones de bucles en el tiempo. Los valores de *kphs* son normalizados en el rango de 0 a 1, siendo 1 el final de la tabla de función *ifna*.

kgliss - ajusta la altura final de cada gránulo con respecto a la altura inicial, indicada en octavas. Así *kgliss* = 2 significa que el gránulo termina dos octavas por encima de su altura inicial, mientras que *kgliss* = -5/3 indica que el gránulo terminará una sexta mayor natural por debajo. **Nota:** No hay parámetros opcionales en **fof2**.

El generador de Csound **fof** está basado ligeramente en la codificación a lenguaje C, de Michael Clarke, del programa CHANT del IRCAM (Xavier Rodet y colaboradores). Cada **fof** produce un sólo formante y las salidas de 4 o más de estas unidades pueden ser sumadas para producir una imitación vocal rica. La síntesis con **fof** es una forma especial de síntesis granular, y su implementación proporciona un método de transformación (*morphing*) de imitaciones vocales en texturas granulares y viceversa. La velocidad del proceso de computación depende de los valores de *kdur* y *xfund*, así como de la densidad de los solapamientos.

40.2 fog

```
ar    fog    xamp, xdens, xtrans, xspd, koct, kband, kris, kdur, \\
      kdec, iolaps, ifna, ifnb, itotdur[, iphs[, itmode]]
```

40.2.1 DESCRIPCIÓN

La salida de audio es una sucesión de gránulos, derivada de los datos almacenados en una tabla de función *ifna*. El envoltente local de estos gránulos y su duración y posición en el tiempo están basados en el modelo de síntesis con **fof**, permitiendo un detallado control sobre el proceso de síntesis granular.

40.2.2 INICIALIZACIÓN

iolaps - número de espacios preasignados necesarios para almacenar los datos del solapamiento de los gránulos. Los solapamientos dependen de la densidad, mientras que el espacio requerido depende del valor máximo de $xdens * kdur$. Puede ser sobreestimado sin coste adicional de tiempo de computación. Se usan menos de 50 bytes de memoria para cada solapamiento *iolaps*.

ifna, *ifnb* - números de dos tablas de función almacenadas. La primera contiene los datos usados en el proceso de granulación, normalmente procedentes de un fichero de sonido (**GEN01**). La segunda es la curva del ataque, usada hacia delante y hacia atrás, para dibujar el ataque y la caída de cada gránulo; normalmente es un sigmoide (**GEN19**) pero puede ser también lineal (**GEN07**).

itotdur - tiempo total durante el cual **fof** estará activo. Normalmente concuerda con p3. No se crea ningún gránulo nuevo si no se va a poder completar su *kdur* (duración del gránulo) en lo que queda de *itotdur* (duración total).

iphs (opcional) - fase inicial de la fundamental, expresada como una fracción de ciclo (de 0 a 1). El valor por defecto es 0.

itmode (opcional) - modo de transposición. Si es 0, cada gránulo mantiene el valor *xtrans* con el que fue lanzado. Si no es 0, cada uno de ellos es continuamente influenciado por el valor de *xtrans*. El valor por defecto es 0.

40.2.3 EJECUCIÓN

xamp - factor de amplitud. La amplitud también depende del número de gránulos solapados, de la interacción de la curva de ataque (*ifnb*), de la caída exponencial (*kband*) y de la normalización de la forma de onda de cada gránulo (*ifna*). La amplitud real puede exceder el valor de *xamp*.

xdens - densidad, esto es, la frecuencia de gránulos por segundo.

xtrans - factor de transposición, es decir, la velocidad a la cual se lee, para cada gránulo, la tabla de función almacenada *ifna*. Tiene el mismo efecto que transportar el material original. Un valor de 1, reproduce la altura original. Un valor mayor, transportará la altura hacia arriba, uno menor hacia abajo. Valores negativos provocan que la tabla de función sea leída al revés.

xspd - velocidad a la que los sucesivos gránulos avanzan a través de la tabla de función *ifna* almacenada. *xspd* toma la forma de un índice (de 0 a 1) en la tabla *ifna*. Esto determina el movimiento de un puntero, usado como punto de comienzo para leer los datos de cada gránulo. (*xtrans* determina la velocidad a la cual se leen los datos, empezando por la posición indicada por dicho puntero).

koct - índice de "octavación". La operación de este parámetro es idéntica a la del parámetro de **fof**.

kband, *kris*, *kdur*, *kdec* - curva del envolvente de cada gránulo. Estos parámetros determinan la duración de la caída exponencial (*kband*), del ataque (*kris*), de la duración total (*kdur*) y de la caída (*kdec*) del envolvente de cada gránulo. Su operación es idéntica a la del parámetro del envolvente local de **fof**.

El generador **fog** de Csound fue escrito por Michael Clarke, desarrollando su trabajo previo basado en el algoritmo **fof** del IRCAM.

40.2.4 EJEMPLO

```

; p4 = factor de transposición
; p5 = factor de velocidad
; p6 = tabla de función para los datos de los gránulos
i1 = sr/ftlen(p6) ; normalización para reflejar la frecuencia de muestreo
; y la longitud de la tabla
a1 phasor i1*p5 ; índice de velocidad
a2 fog 5000, 100, p4, a1, 0, 0, , .01, .02, .01, 2, p6, 1, p3, 0, 1
```

40.2.5 AUTOR

Michael Clark
Huddersfield
Mayo 1997

40.3 grain

```
ar      grain      xamp, xpitch, xdens, kampoﬀ, kpitchoﬀ, kgdur, igfn, \\
        iwfn, imgdur [, igrnd]
```

40.3.1 DESCRIPCIÓN

Genera texturas de síntesis granular.

40.3.2 INICIALIZACIÓN

igfn - número de la tabla de función de la forma de onda. Puede ser desde una onda sinusoidal hasta un sonido muestreado.

iwfn - número de la tabla de función del envolvente de amplitud usado para los gránulos (ver también **GEN20**).

imgdur - duración máxima de los gránulos (en segundos). Es el mayor valor asignado a *kgdur*.

igrn - (opcional) si no es 0, desactiva la aleatoriedad del offset de cada gránulo. Esto significa que todos los gránulos empezarán a leer desde el principio de la tabla *igfn*. Si es 0 (valor por defecto), los gránulos empezarán a leer desde posiciones aleatorias de la tabla *igfn*.

40.3.3 EJECUCIÓN

xamp - amplitud de cada gránulo.

xpitch - altura del gránulo. Para usar la frecuencia original del sonido de entrada, usa la fórmula: " $snds$ r / $ftlen(igfn)$ " donde *snds*r es la frecuencia de muestreo original del sonido *igfn*.

xdens - densidad de los gránulos, medida en gránulos por segundo. Si es constante, entonces la salida es un proceso de síntesis granular síncrona, muy parecida a la de **fof**. Si *xdens* tiene un elemento de aleatoriedad (como ruido añadido, por ejemplo) el resultado es un proceso de síntesis granular asíncrona.

kampoﬀ - desviación máxima de la amplitud *kamp*. Esto significa que la amplitud máxima que un gránulo puede tener es *kamp* + *kampoﬀ* y la mínima es *kamp*. Si *kampoﬀ* es puesto a 0 no habrá amplitud aleatoria para cada gránulo.

kpitchoﬀ - desviación máxima de la altura *kpitch* expresada en Hz. Similar a *kampoﬀ*.

kgdur - duración del gránulo (en segundos). El valor máximo debe ser declarado en *imgdur*. Si *kgdur* llegase a ser en algún punto mayor que *imgdur*, sería truncado al valor de este último.

El generador de gránulos está basado principalmente en el trabajo y los escritos de Barry Truax y Curtis Roads.

40.3.4 EJEMPLO

Una textura con gránulos gradualmente más cortos y con mayor amplitud y rango de alturas.

```
;;;;;;;;;;;;; graintest.orc
instr          1
insnd          =   10
ibasfrq = 32000 / ftlen(insnd) ; Usa la frecuencia de muestreo original del fichero
insnd
kamp expseg 8000, p3/2, 8000, p3/2, 16000
kpitch line      ibasfrq, p3, ibasfrq * .8
kdens      line 600, p3, 200
kaoff      line 0, p3, 5000
kpoff      line 0, p3, ibasfrq * .5
kgdur      line .4, p3, .1
imaxgdur = .5
ar grain kamp, kpitch, kdens, kaoff, kpoff, kgdur, insnd, 5, imaxgdur, 0.0
out ar
endin

;;;;;;;;;;;;; graintest.sco
f5 0 512 20 2 ; Hanning window
f10 0 65536 1 "Sound.wav" 0 0 0
i1 0 10
e
```

40.3.5 AUTOR

Paris Smaragdis
MIT
Mayo 1997

40.4 granule

```
asig granule      xamp, ivoice, iratio, imode, ithd, ifn, ipshift, \\
                   igskip, igskip_os, ilength, kgap, igap_os, kgsiz, \\
                   igsiz_os, iatt, idec [,iseed[,ipitch1[,ipitch2\\
                   [,ipitch3[,ipitch4[,ifnenv]]]]]]]
```

40.4.1 DESCRIPCIÓN

El opcode **granule** es más complejo que **grain**, pero también añade nuevas posibilidades. **granule** es un opcode que emplea una tabla de onda como entrada para producir una salida de audio granular. Los datos de la tabla de onda pueden ser generados por cualquiera de las rutinas GEN, incluida la **GEN01**, que lee un fichero de sonido y lo convierte en una tabla de función. Esto permite que pueda usarse un sonido sampleado como fuente para los gránulos. Se pueden implementar internamente hasta 128 voces. El número de voces máximo puede ser incrementado redefiniendo la variable MAXVOICE en el fichero "grain4.h". **granule** tiene un generador de número aleatorios propio para manejar los parámetros de offsets aleatorios.

También se ha implementado un proceso de búsqueda de umbrales (*thresholding*) en el escaneo de inicialización de la tabla de función. Esta característica facilita, por ejemplo, los procesos de eliminación de pasajes de silencio entre frases.

Las características de la síntesis se controlan con 22 parámetros. *xamp* es la amplitud de la salida y puede ser una variable tanto de audio (a-) como de control (-k).

40.4.2 EJECUCIÓN

xamp - amplitud.

ivoice - número de voces.

iratio - cociente de la velocidad del puntero *gskip* y la frecuencia de muestreo. Por ejemplo, 0.5 será la mitad de dicha velocidad.

imode - un valor de +1 mueve el puntero del gránulo hacia delante (es decir, en la misma dirección que el puntero *gskip*). Un valor de -1 mueve el puntero hacia atrás (en la dirección opuesta al puntero *gskip*). Un valor 0 lo moverá aleatoriamente.

ithd - umbral. Si la señal sampleada en la tabla de onda es menor que *ithd*, será omitida.

ifn - número de la tabla de función de la fuente de sonido.

ipshift - control de desplazamiento de altura. Si *ipshift* es 0, la altura será calculada aleatoriamente fluctuando una octava arriba y abajo. Si *ipshift* es 1,2,3 ó 4, se pueden definir hasta cuatro alturas diferentes para el número de voces indicado por *ivoice*. Los parámetros opcionales *ipitch1*, *ipitch2*, *ipitch3* y *ipitch4* se usan para cuantificar los desplazamientos de altura.

igskip - duración en segundos del segmento omitido inicialmente, medida desde el principio de la tabla de función.

igskip_os - offset aleatorio del puntero *gskip* (en segundos). 0 significa que no hay offset.

ilength - longitud, en segundos, de la tabla usada empezando desde *igskip*.

kgap - espacio entre gránulos (en segundos).

igap_os - offset aleatorio del espacio entre gránulos (medido en % del tamaño del espacio). 0 indica que no hay offset.

*kgsiz*e - tamaño del gránulo (en segundos).

igsize_os - offset aleatorio del gránulo (medido en % del tamaño del gránulo). 0 indica que no hay offset.

iatt - ataque del envolvente del gránulo (en % del tamaño del gránulo).

idec - caída del envolvente del gránulo (en % del tamaño del gránulo).

[iseed] - opcional, valor germinal del generador de números aleatorios. El valor por defecto es 0.5.

[ipitch1], *[ipitch2]*, *[ipitch3]*, *[ipitch4]*- (opcional) parámetro de desplazamiento de altura, usado cuando *ipshift* tiene un valor de 1,2,3 ó 4. Se usa una técnica de escalado temporal para el desplazamiento de altura con interpolación entre datos consecutivos. El valor por defecto es 1, es decir, la altura original.

40.4.3 EJEMPLO

40.4.3.1 Orquesta:

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
instr 1
k1    linseg    0,0.5,1,(p3-p2-1),1,0.5,0
a1    granule   p4*k1,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,\
           p16,p17,p18,p19,p20,p21,p22,p23,p24
a2    granule   p4*k1,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,\
           p16,p17,p18,p19, p20+0.17,p21,p22,p23,p24
      outs      a1,a2
endin
```

40.4.3.2 Partitura:

```
; la sentencia f lee un fichero de sonido llamado sine.aiff
; en el directorio SFDIR y lo almacena en la tabla de función 1
f1 0 524288 1 "sine.aiff" 1 0
i1 0 10 2000 64 0.5 0 0 1 4 0 0.005 10 0.01 50 0.02 50 30 30 0.39 \
1 1.42 0.29 2
e
```

El ejemplo anterior lee un fichero de sonido llamado *sine.aiff* y lo almacena en la tabla de función número 1 que contendrá 524,288 muestras. Genera 10 segundos de audio estéreo, usando dicha tabla de función. En el fichero orquesta, todos los parámetros requeridos para controlar el proceso de síntesis se pasan desde el fichero partitura. Se usa una función **linseg** para generar un envolvente con un ataque y una

caída lineales de 0.5 segundos. El efecto estéreo es generado usando diferentes valores germinales para las dos llamadas a la función **granule**. En el ejemplo, se añade 0.17 al campo p20 antes de pasar a la segunda llamada a **granule**, para asegurar que todos los offsets aleatorios son diferentes a los de la primera llamada.

En el fichero partitura, los parámetros se interpretan así:

p5 (ivoice) el número de voces es 64
p6 (iratio) es 0.5, es decir, escanea la tabla de onda a la mitad de la frecuencia de muestreo
p7 (imode) es 0, es decir, el puntero del gránulo sólo se mueve hacia delante
p8 (ithd) es 0, es decir, omite el proceso de búsqueda de umbrales
p9 (ifn) es 1, es decir, se usa la tabla de función número 1
p10 (ipshift) es 4, es decir, se van a generar 4 diferentes alturas
p11 (igskip) es 0 y el campo
p12 (igskip_os) es 0.005, es decir, no hay omisiones en la tabla de onda y se usa un offset aleatorio de 5 milisegundos.
p13 (ilength) es 10, es decir, se usan 10 segundos de la tabla de onda
p14 (kgap) es 0.01 y el campo
p15 (igap_os) es 50, es decir, hay un espacio de 10 msecs con un offset aleatorio del 50%
p16 (kgsiz) es 0.02 y el campo
p17 (igsiz_os) es 50, es decir, se usan gránulos de 20 msecs con un offset aleatorio del 50%
p18 (iatt) y **p19** (idec) son 30, es decir, se aplica un ataque y una caída lineales del 30% al gránulo
p20 (iseed) el valor germinal del generador de números aleatorios es 0.39.
p21 - p24 son las alturas: la primera es 1 (la altura original), la segunda es 1.42 (una quinta ascendente), la tercera es 0.29 (una séptima descendente) y la cuarta es 2 (una ocatava ascendente)

40.4.4 AUTOR

Allan Lee
Belfast
1996

40.5 **sndwarp**, **sndwarpst**

<code>ar[,ac]</code>	sndwarp	<code>xamp, xtimewarp, xresample, ifn1, ibeg, \\</code> <code>iwsiz, irandw, ioverlap, ifn2, itimemode</code>
<code>ar1,ar2[,ac1,ac2]</code>	sndwarpst	<code>xamp, xtimewarp, xresample, ifn1, ibeg, \\</code> <code>iwsiz, irandw, ioverlap, ifn2, itimemode</code>

40.5.1 DESCRIPCIÓN

sndwarp lee las muestras de sonido almacenadas en una tabla y efectúa una dilatación temporal y/o una modificación de la altura. Ambas son independientes. Por ejemplo, se puede alargar la duración de un sonido y subir su altura al mismo tiempo. Los argumentos que indican el tamaño y solapamiento de las ventanas son importantes para el resultado y es bueno experimentar con ellos. En general, deben ser tan pequeños como sea posible. Por ejemplo, empieza por *iwsiz=sr/10* y *ioverlap=15*. Prueba con *irandw=iwsiz*.2*. Si te las puedes arreglar con menos solapamientos, el programa será más rápido. Pero demasiados pocos pueden causar un efecto de aleteo audible en la amplitud. El algoritmo reacciona de diferente manera dependiendo del sonido de entrada y no se pueden dar reglas fijas para su óptimo uso en distintas circunstancias. Sin embargo, mediante el método de prueba y error, se pueden conseguir excelentes resultados.

40.5.2 INICIALIZACIÓN

ifn1 - número de la tabla que contiene las muestras de sonido que serán sometidas al procesado de **sndwarp**. La rutina **GEN01** es la que se utiliza para almacenar los datos de un fichero de sonido preexistente en una tabla de función.

ibeg - duración en segundos después de la cual empezar a leer la tabla (o el fichero de sonido). Cuando *itimemode* es distinto de 0, el valor de *xtimewarp* es desplazado según *ibeg*.

iwsiz - tamaño de la ventana (medido en número de muestras) que se usa en el algoritmo de normalización.

irandw - ancho de banda del generador de números aleatorios. Dichos números serán añadidos a *iwsiz*.

ioverlap - determina la densidad de solapamiento de las ventanas.

ifn2 - función usada para dibujar la ventana. Normalmente se utiliza para crear una especie de rampa empezando desde 0 y volviendo, al final de cada ventana, al mismo valor 0. Prueba a usar la mitad de una función seno (es decir, f1 0 16384 9 .5 1 0), que funciona bastante bien. Se pueden usar también otras curvas.

40.5.3 EJECUCIÓN

asig es el único canal de salida del opcode **sndwarp**, mientras que *asig1* y *asig2* son los canales izquierdo y derecho de la salida estéreo de **sndwarpst**. **sndwarp** asume que la tabla de función que contiene la señal muestreada es mono, mientras que **sndwarpst** asume que es estéreo. Esto significa simplemente que **sndwarp** indexará la tabla con incrementos muestra a muestra y **sndwarpst** lo hará de dos en dos. El usuario debe ser consciente de que si una señal mono se usa con **sndwarpst**, o una estéreo con **sndwarp**, la duración y la altura se verán por consiguiente alteradas.

acmp en **sndwarp** y *acmp1*, *acmp2* en **sndwarpst**, son versiones de una sólo capa (es decir, sin solapamientos) y sin ventanas de la señal alterada en duración y/o altura. Se proporcionan como medio para de equilibrar la amplitud de la señal de salida, que normalmente contiene muchos solapamientos, de acuerdo con una señal libre de alteraciones de duración o altura. El proceso de **sndwarp** puede causar cambios perceptibles en la amplitud (arriba y abajo), debidos a la diferencia de duración entre los solapamientos, cuando se realiza desplazamiento de tiempo. Cuando se usan con una señal equilibrada, *acmp*, *acmp1* y *acmp2* pueden mejorar mucho la calidad del sonido. Son opcionales, pero observa que en **sndwarpst**, ambas deben estar presentes en la sintaxis. Luego veremos un ejemplo de su uso.

xamp es el valor según el cual se normaliza la amplitud (ver la nota sobre su uso cuando se utilizan los argumentos *acmp*, *acmp1*, y *acmp2*).

xtimewarp determina como será dilatada o comprimida la duración de la señal de entrada. Hay dos maneras de usar este argumento dependiendo del valor que demos a *itimemode*. Cuando el valor de *itimemode* es 0, *xtimewarp* escalará la duración del sonido. Por ejemplo, un valor de 2 alargará el sonido el doble. Cuando *itimemode* tome cualquier valor distinto de 0, se usará *xtimewarp* como puntero temporal, que funcionará de manera similar a como trabaja el puntero de **lpread** y **pvoc**. El ejemplo de abajo ilustra este proceso. En ambos casos, la altura no se verá de ninguna manera afectada por este proceso. El desplazamiento de altura se realiza independientemente, usando el parámetro *xresample*.

xresample es el factor por el cual se modifica la altura del sonido. Por ejemplo, un valor de 2 producirá una salida una octava más alta que el original. La duración del sonido, en cambio, no se verá alterada.

40.5.4 EJEMPLO

El ejemplo siguiente muestra una desaceleración (o dilatamiento de la duración) de un sonido almacenado en la tabla *ifn1*. Durante la duración de la nota, la dilatación pasará de 0 a ser un sonido 10 veces más "lento" que el original. A la vez, la altura total ascenderá una octava a lo largo de toda la duración del sonido.

```

iwindfun    =          1
isampfun    =          2
ibeg        =          0
iwindsize   =         2000
iwindrand   =          400
ioverlap    =          10
awarp       line      1, p3, 1
aresamp     line      1, p3, 2
kenv        line      1, p3, .1
asig        sndwarp   kenv,awarp,aresamp,isampfun,ibeg,iwindsize,iwindrand, \
                    ioverlap,iwindfun,0

```

Ahora, un ejemplo estéreo usando *xtimewarp* como puntero temporal:

```

itimemode   =          1
atime       line      0, p3, 10
asig1, asig2 sndwarpst kenv, atime, aresamp, sampfun, ibeg, \
                    iwindsize, iwindrand, ioverlap, \
                    iwindfun, itimemode

```

En este ejemplo que hemos visto, *atime* hace avanzar al puntero usado en **sndwarp** desde 0 hasta 10 a lo largo de la duración de la nota. Si p3 es 20, entonces el sonido será 2 veces más lento que el

original. Por supuesto, se puede usar una función más compleja que una simple recta para controlar el factor temporal.

Ahora, lo mismo que antes pero usando esta vez la función de balance con las salidas opcionales:

asig,acmp	sndwarp	1,awarp,aresamp,isampfun,\ ibeg,iwindsize,iwindrand,\ ioverlap,iwindfun,itimemode
abal	balance	asig,acmp
asig1,asig2,acmp1,acmp2	sndwarpst	1,atime,aresamp,sampfun,\ ibeg,iwindsize,iwindrand,\ ioverlap,iwindfun,itimemode
abal1	balance	asig1,acmp1
abal2	balance	asig2,acmp2

En los anteriores dos ejemplos se puede observar el uso de la unidad de balance. La salida de **balance** puede ser normalizada, enviada a una salida mono (**out**) o estéreo (**outs**), etc. Observa que los argumentos referentes a la amplitud de **sndwarp** y **sndwarpst** son puestos a 1 en estos ejemplos. Normalizando la señal después del proceso de **sndwarp**, *abal*, *abal1*, y *abal2* deberían contener señales casi con la misma amplitud que la señal original de entrada del procesado de **sndwarp**. Esto facilita la predicción de los niveles de pico y la prevención de las muestras fuera de rango (o demasiado débiles).

Más consejos: Usa la versión estéreo sólo cuando realmente necesites procesar un fichero estéreo. Es un poco más lenta que la versión mono y, además, si usas **balance** para equilibrar la señal se hace todavía más lenta. No hay nada malo en usar la versión **sndwarp** mono en una orquesta estéreo, ya que podemos enviar la señal de salida a uno o a ambos canales del estéreo.

40.5.5 AUTOR

Richard Karpen
Seattle, Wash
1997

41 SEÑAL GENERADORES DE: FÍSICO MODELADO POR GUÍA DE ONDAS

41.1 pluck

ar **pluck** kamp, kcps, icps, ifn, imeth [, iparm1, iparm2]

41.1.1 DESCRIPCIÓN

El sonido de salida es bien el de la caída natural de una cuerda punteada, o bien un sonido de percusión, basados en los algoritmos de Karplus-Strong.

41.1.2 INICIALIZACIÓN

icps - valor de altura deseado en Hz, usado para configurar un buffer con muestras de 1 ciclo de audio, que serán suavizadas a lo largo de la duración del sonido por el método de caída escogido. Normalmente, *icps* anticipa el valor de *kcps*, pero puede tomar un valor artificialmente alto o bajo para influir en el tamaño del buffer de muestras.

ifn - número de la tabla de función almacenada usada para inicializar el buffer de caída cíclica. Si *ifn* = 0, se usará una secuencia aleatoria en su lugar.

imeth - método de caída natural. Hay 6, algunos de los cuales usan los parámetros siguientes:

1. promedio simple. Un proceso simple de suavizado, no influenciado por ningún parametro.
2. promedio dilatado. Como el anterior, pero con una duración dilatada del proceso de suavizado según un factor *iparm1* ($= 1$).
3. percusión simple. El rango, que va desde el sonido de un tono simple a un ruido, se controla por el factor de "aspereza" *iparm1* (que toma valores de 0 a 1). 0 da un efecto de cuerda punteada, mientras que 1 invierte la polaridad de cada muestra (octava abajo, armónicos impares). El valor 0.5 da un valor óptimo para imitar una caja.
4. percusión prolongada. Combina los factores de aspereza (*iparm1*, que toma valores de 0 a 1) y dilatación (*iparm2* = 1).
5. media ponderada. Funciona como el método 1, pero con *iparm1* ponderando la muestra actual (el status quo) y *iparm2* haciendo lo propio con la muestra previa adyacente. $iparm1 + iparm2$ debe ser ≤ 1 .
6. filtro recursivo de primer orden, de coeficientes .5. No se ve afectado por el valor de los parámetros. *iparm1*, *iparm2* son opcionales y son usados por los algoritmos de suavizado. Los valores por defecto son 0.

41.1.3 EJECUCIÓN

Un buffer de audio interno, relleno en la inicialización según *ifn*, es continuamente remuestreado con una periodicidad *kcps* y la salida resultante es multiplicada por *kamp*. Al mismo tiempo que se efectúa el proceso de muestreo, el buffer es suavizado para simular el efecto de una caída natural.

Los efectos de cuerdas punteadas (1,2,5 y 6) se consiguen mejor empezando con una fuente de ruido aleatoria, rica en armónicos iniciales. Los sonidos percusivos (3,4) trabajan mejor con una fuente plana (un pulso ancho, por ejemplo), que produce un ruido de ataque profundo y una caída marcada.

El algoritmo original de Karplus-Strong usaba un número fijo de muestras por ciclo, que causaba una severa cuantización de las alturas disponibles y su afinación. Esta implementación remuestrea un buffer a la frecuencia exacta especificada por *kcps*, que puede ser variada para conseguir efectos de vibrato o gilssando. Para valores bajos de la frecuencia de muestreo de la orquesta (por ejemplo, *sr*=10000), las altas frecuencias almacenarán sólo unas pocas muestras (*sr / icps*). Debido a que esto puede causar un ruido perceptible en el proceso de remuestreo, el buffer interno tiene un tamaño mínimo de 64 muestras. Este valor puede ser alargado ajustando *icps* a un valor artificialmente bajo.

41.2 wgpluck

ar **wgpluck** *icps, iamp, kpick, iplk, idamp, ifilt, axcite*

41.2.1 DESCRIPCIÓN

Una simulación de alta fidelidad del efecto de una cuerda punteada, usando interpolación de líneas de retardo.

41.2.2 INICIALIZACIÓN

icps –frecuencia de la cuerda punteada.

iamp – amplitud de la cuerda punteada.

iplk –punto de la cuerda en el que será punteada, especificado en el rango de 0 a 1. Un valor 0 indica que no hay punteo.

idamp –caída de la nota. Controla la caída total de la cuerda. Cuanto mayor el valor de *idamp*, más rápida la caída. Valores negativos darán lugar a una subida de la salida progresiva en el tiempo.

ifilt –controla la atenuación del filtro en el puente. Valores altos darán lugar a que los armónicos altos caigan más rápidamente.

41.2.3 EJECUCIÓN

kpick –proporción del tamaño de la cuerda en relación con el punto donde se sampleará la oscilación de salida.

axcite –señal que excita la cuerda.

Una cuerda de frecuencia *icps* es punteada con una amplitud *iamp* en el punto *iplk*. La caída de la cuerda virtual se controla mediante *idamp* y *ifilt* que simulan el puente. La oscilación se samplea en el punto *kpick*, entendiendo que la cuerda ha sido excitada por la señal *axcite*.

41.2.4 EJEMPLOS

El siguiente ejemplo produce una nota moderadamente larga con una caída pronunciada de sus parciales superiores:

```
instr 1
axcite      oscil      1, 1, 1
apluck      wgpluck    220, 120, .5, 0, 10, 1000, axcite
            out        apluck
endin
```

mientras que el siguiente produce una nota corta y brillante:

```
instr 1
axcite
apluck
endin

oscil      1, 1, 1
wgpluck   220, 120, .5, 0, 30, 10, axcite
out       apluck
```

41.3 repluck, wGPLUCK2

ar	wGPLUCK2	<i>iplk, xam, icps, kpick, krefl</i>
ar	repluck	<i>iplk, xam, icps, kpick, krefl, axcite</i>

41.3.1 DESCRIPCIÓN

wGPLUCK2 es una implementación del modelo físico de una cuerda punteada, con control sobre el punto en que ésta es punteada, la fuerza del punteo y el filtro. **repluck** realiza la misma operación pero con una señal de audio adicional, *axcite*, usada para excitar la cuerda. Ambos opcodes se basan en los algoritmos de Karplus-Strong.

41.3.2 INICIALIZACIÓN

icps - la cuerda vibrará a una frecuencia *icps*.

iplck - el punto donde se puntea la cuerda es *iplck*, que se expresa como una fracción de la longitud de la cuerda (en el rango de 0 a 1). Un valor 0 indica que no hay punteado inicial.

41.3.3 EJECUCIÓN

xamp es la ganancia y *kpick* indica qué proporción de la cuerda ha de reproducirse en la salida. La reflexión en el puente se controla por el coeficiente de reflexión *krefl*, para el que 1 significa reflexión total y 0 reflexión nula.

41.3.4 AUTOR

John ffitcH
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
1997

41.4 wgbow

ar **wgbow** kamp, kfreq, kpres, krat, kvibf, kvamp, ifn[, iminfreq]

41.4.1 DESCRIPCIÓN

La salida de audio es un sonido parecido al de una cuerda frotada, usando un modelo físico desarrollado por Perry Cook, pero recodificado para Csound.

41.4.2 INICIALIZACIÓN

ifn - tabla de la curva de vibrato, normalmente con una función sinusoidal.

iminfreq - frecuencia más baja a la que el instrumento podrá tocar. Si se omite, se usa el valor inicial de *kfreq*. Si *iminfreq* es negativo, se omitirá la inicialización.

41.4.3 EJECUCIÓN

Se ejecuta una nota en un instrumento de cuerda frotada, con los argumentos que siguen:

kamp - amplitud de la nota.

kfreq - frecuencia de la nota.

kpres - un parámetro que controla la presión del arco sobre la cuerda. Los valores deben estar alrededor de 3. El rango útil es aproximadamente de 1 a 5.

krat - la posición del arco a lo largo de la cuerda. Normalmente se toca en una posición de 0.127236. El rango sugerido va desde 0.025 a 0.23.

kvibf - frecuencia del vibrato en Hz. Rango recomendado: de 0 a 12.

kvamp - amplitud del vibrato.

41.4.4 EJEMPLO

```
kv      linseg      0, 0.5, 0, 1, 1, p3-0.5, 1
a1      wgbow      31129.60, 440, 3.0, 0.127236, 6.12723, kv*0.01, 1
         out        a1
```

41.4.5 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

41.5 wgflute

```
ar    wgflute    kamp, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, \\
                    ifn[, iminfreq[, kjetrf[, kendraf]]]
```

41.5.1 DESCRIPCIÓN

La salida de audio es un sonido parecido al de la flauta, obtenido mediante un modelo físico desarrollado por Perry Cook, pero recodificado para Csound.

41.5.2 INICIALIZACIÓN

iatt - tiempo en segundos necesario para alcanzar la presión máxima del chorro de aire. 0.1 parece corresponder a una ejecución natural.

idetk - tiempo en segundos que se tarda en dejar de soplar. 0.1 es un final suave.

ifn - tabla de la curva del vibrato, normalmente una función seno.

iminfreq - frecuencia más baja a la que se puede tocar el instrumento. Si se omite, se usa el valor inicial de *kfreq*. Si *iminfreq* es negativo, se omitirá la inicialización.

41.5.3 EJECUCIÓN

Se ejecuta una nota en un instrumento similar a una flauta, con los parámetros siguientes:

kamp - amplitud de la nota.

kfreq - frecuencia de la nota. Aunque puede ser variada durante su ejecución, esta opción todavía no ha sido comprobada.

kjet - un parámetro que controla el chorro de aire. Los valores deben ser positivos y alrededor de 0.3. El rango útil va de 0.08 a 0.56 aproximadamente.

kngain - amplitud del componente de ruido, alrededor de 0 a 0.5

kvibf - frecuencia del vibrato en Hz. Rango recomendado: de 0 a 12.

kvamp - amplitud del vibrato.

41.5.4 EJEMPLO

```
a1    wgflute    31129.60, 440, 0.32, 0.1, 0.1, 0.15, 5.925, 0.05, 1
      out      a1
```

41.5.5 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

41.6 wgb Brass

ar **wgb Brass** kamp, kfreq, ktens, iatt, kvibf, kvamp, ifn[, iminfreq]

41.6.1 DESCRIPCIÓN

La salida de audio es un sonido parecido al de un instrumento de metal, usando un modelo físico desarrollado por Perry Cook, pero recodificado para Csound. Nota: Es más bien pobre aún, y con pocas posibilidades de control por el momento. Necesita una revisión y quizás la inclusión de más parámetros.

41.6.2 INICIALIZACIÓN

iatt -- tiempo tomado para alcanzar la presión máxima.

ifn - tabla de la curva de vibrato, normalmente una función sinusoidal.

iminfreq - frecuencia más baja a la que se puede tocar el instrumento. Si se omite, se usa el valor inicial de *kfreq*. Si *iminfreq* es negativo, se omitirá la inicialización.

41.6.3 EJECUCIÓN

Se ejecuta una nota en un instrumento similar a uno de metal, con los argumentos siguientes:

kamp - amplitud de la nota.

kfreq - frecuencia de la nota.

kvibf - frecuencia del vibrato en Hz. Rango recomendado: de 0 a 12.

kvamp - amplitud del vibrato.

41.6.4 EJEMPLO

```
a1    wgb Brass    31129.60, 440, 0.1, 6.137, 0.05, 1  
      out        a1
```

41.6.5 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

41.7 wgclar

```
ar    wgclar    kamp, kfreq, kstiff, iatt, idetk, kngain, kvibf, \\  
kvamp, ifn[, iminfreq]
```

41.7.1 DESCRIPCIÓN

La salida de audio es un sonido parecido al del clarinete, obtenido mediante un modelo físico desarrollado por Perry Cook, pero recodificado para Csound.

41.7.2 INICIALIZACIÓN

iatt - tiempo en segundos necesario para alcanzar la máxima presión del chorro de aire. 0.1 parece corresponder a una ejecución natural. Un valor más largo proporciona el ataque inicial bien definido que se produce de manera natural al tomar aliento.

idetk - tiempo en segundos que se tarda en dejar de soplar. 0.1 es un final suave.

ifn - tabla de la curva de vibrato, normalmente una función sinusoidal.

iminfreq - frecuencia más baja a la que se puede tocar el instrumento. Si se omite, se usa el valor inicial de *kfreq*. Si *iminfreq* es negativo, se omitirá la inicialización.

41.7.3 EJECUCIÓN

Se ejecuta una nota en un instrumento similar al clarinete, con los parámetros siguientes:

kamp - amplitud de la nota.

kfreq - frecuencia de la nota.

kstiff - un parámetro de rigidez para la lengüeta. Los valores deben ser negativos y rondando -0.3. El rango útil va desde -0.44 a -0.18 aproximadamente.

kngain - amplitud del componente de ruido. Alrededor de 0 a 0.5.

kvibf - frecuencia del vibrato en Hz. El rango sugerido va de 0 a 12.

kvamp - amplitud del vibrato.

41.7.4 EJEMPLO

```
a1    wgclar    31129.60, 440, -0.3, 0.1, 0.1, 0.2, 5.735, 0.1, 1  
out   a1
```

41.7.5 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

42 GENERADORES DE SEÑAL: MODELOS Y EMULACIONES

42.1 moog

```
a1      moog      kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, \\
        iwfn, ivfn
```

42.1.1 DESCRIPCIÓN

Una emulación de un sintetizador mini-Moog.

42.1.2 INICIALIZACIÓN

iafn, *iwfn*, *ivfn* -- los tres números de las tablas que contienen la forma de onda del ataque (sin bucle), la forma de onda del bucle principal y la forma de onda del vibrato, respectivamente. Los ficheros *mandpluk.aiff* y *impuls20.aiff* sirven muy bien para los dos primeros argumentos, mientras que una onda sinusoidal es la más apropiada para el último.

42.1.3 EJECUCIÓN

kamp - amplitud de la nota.

kfreq - frecuencia de la nota.

kfiltq - factor Q del filtro, en el rango de 0.8 a 0.9.

kfiltrate - control de frecuencia para el filtro en el rango de 0 a 0.0002.

kvibf - frecuencia del vibrato en Hz. El rango sugerido es de 0 a 12.

kvamp - amplitud del vibrato.

42.1.4 AUTOR

John ffitc
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

42.2 shaker

ar **shaker** kamp, kfreq, kbeans, kdamp, ktimes[, idecay]

42.2.1 DESCRIPCIÓN

El audio de salida es un sonido parecido al de las maracas u otros instrumentos similares con forma de calabaza. El método usado es un modelo físico desarrollado por Perry Cook, pero recodificado aquí para Csound.

42.2.2 INICIALIZACIÓN

idecay - Si está presente, indica por cuanto tiempo será amortiguada la maraca al final de la nota. El valor por defecto es 0.

42.2.3 EJECUCIÓN

Se ejecuta una nota en un instrumento similar a una maraca, con los parámetros siguientes:

kamp - amplitud de la nota.

kfreq -frecuencia de la nota.

kbeans -número de guijarritos dentro de la calabaza. Un valor de 8 va bien.

kdamp -- valor de amortiguación de la sacudida de la maraca. Los valores de 0.98 a 1 van bien, siendo 0.99 un valor por defecto bastante razonable.

ktimes -- número de sacudidas de la calabaza. Los valores mayores que 64 se consideran infinitos.

El argumento *knum* era redundante, así que fue eliminado a partir de la versión 3.49

42.2.4 EJEMPLO

```
a1    shaker    31129.60, 440, 8, 0.999, 0, 100, 0  
      outs     a1, a1
```

42.2.5 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

42.3 marimba, vibes

ar	marimba	kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec
ar	vibes	kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec

42.3.1 DESCRIPCIÓN

La salida de audio es un sonido parecido al que se produce al golpear un bloque de madera o metal, tal y como sucede en una marimba o un vibráfono. El método usado es un modelo físico desarrollado por Perry Cook, pero recodificado aquí para Csound.

42.3.2 INICIALIZACIÓN

ihrd -- dureza de la baqueta utilizada en el golpe. Se usa un rango de 0 a 1. Un valor de 0.5 da buenos resultados.

ipos -- indica el punto en el que el bloque es golpeado, en el rango de 0 a 1.

imp - una tabla con los impulsos del golpe. El fichero "marmstk1.wav" es una función que viene bien como medida, y puede ser cargado en una tabla por la rutina GEN01.

ivfn - curva del vibrato, normalmente una tabla de función sinusoidal.

idec - instante, antes del final de la nota, en el que se introducirá la amortiguación.

42.3.3 EJECUCIÓN

kamp - amplitud de la nota.

kfreq - frecuencia de la nota.

kvibf - frecuencia del vibrato en Hz. El rango sugerido va de 0 a 12.

kvamp - amplitud del vibrato.

42.3.4 EJEMPLO

```
a1  marimba  31129.60, 440, 0.5, 0.561, 2, 6.0, 0.05, 1, 0.1
a2  vibes    31129.60, 440, 0.5, 0.561, 2, 4.0, 0.2, 1, 0.1a1
outs a1, a2
```

42.3.5 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

42.4 mandol

ar **mandol** kamp, kfreq, kpluck, kdetune, kgain, ksize, ifn [, iminfreq]

42.4.1 DESCRIPCIÓN

Emulación de una mandolina.

42.4.2 INICIALIZACIÓN

ifn -- número de la tabla que contiene la forma de onda del punteo. El fichero "mandpluk.aiff" viene bien para rellenar dicha tabla.

iminfreq -- frecuencia más baja a la que puede ser tocado el instrumento. Si se omite, se toma del valor inicial de *kfreq*.

42.4.3 EJECUCIÓN

kamp - amplitud de la nota.

kfreq - frecuencia de la nota.

kpluck - posición del punteo, en el rango de 0 a 1. Se sugiere 0.4.

kgain - bucle de ganancia del modelo, en el rango de 0.97 a 1.

kdetune - desafinación proporcional entre dos cuerdas. Se sugiere un rango de 0.9 a 1.

ksize - tamaño del cuerpo de la mandolina. El rango va de 0 a 2.

42.4.4 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

42.5 gogobel

ar **gogobel** kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn

42.5.1 DESCRIPCIÓN

El audio de salida es un sonido parecido al del cencerro u otros instrumentos similares cuando son golpeados. El método usado es un modelo físico desarrollado por Perry Cook, pero recodificado aquí para Csound.

42.5.2 INICIALIZACIÓN

ihrd -- la dureza de la baqueta usada en el golpe. Se usa un rango de 0 a 1. Un valor de 0.5 va bien.

ipos -- indica el punto en el que el bloque es golpeado, en el rango de 0 a 1.

imp - una tabla con los impulsos del golpe. El fichero "marmstk1.wav" es una función que viene bien como medida, y puede ser cargado en una tabla por la rutina GEN01.

ivfn - curva del vibrato, normalmente una tabla de función sinusoidal.

42.5.3 EJECUCIÓN

kamp - amplitud de la nota.

kfreq - frecuencia de la nota.

kvibf - frecuencia del vibrato en Hz. El rango sugerido va de 0 a 12.

kvamp - amplitud del vibrato.

42.5.4 EJEMPLO

```
a1    gogobel    31129.60, 440, p4, 0.561, 3, 6.0, 0.3, 1  
      outs        a1, a2
```

42.5.5 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1997

42.6 voice

ar **voice** *kamp*, *kfreq*, *kphoneme*, *kform*, *kvibf*, *kvamp*, *ifn*, *ivfn*

42.6.1 DESCRIPCIÓN

Emulación de la voz humana.

42.6.2 INICIALIZACIÓN

ifn, *ivfn* -- números de las dos tablas que contienen la forma de la onda portadora y la del vibrato. Los ficheros "*impuls20.aiff*", "*ahh.aiff*", "*eee.aiff*" y "*ooo.aiff*" vienen bien para la primera tabla, y una función sinusoidal es la mejor opción para la segunda.

42.6.3 EJECUCIÓN

kamp - amplitud de la nota.

kfreq - frecuencia de la nota. Puede ser variada durante su ejecución.

kphoneme - un entero en el rango de 0 a 16, que selecciona los formantes de los sonidos.

"eee" "ihh" "ehh" "aaa"
"ahh" "aww" "ohh" "uhh"
"uuu" "ooo" "rrr" "lll",
"mmm" "nnn" "nng" "ngg".

En este momento, los fonemas:

"fff" "sss" "thh" "shh",
"xxx" "hee" "hoo" "hah",
"bbb" "ddd" "jjj" "ggg",
"vvv" "zzz" "thz" "zhh"

no están disponibles (!)

kform - ganancia del fonema. Se recomiendan valores entre 0 y 1.2.

kvibf - frecuencia del vibrato en Hz. El rango sugerido va de 0 a 12.

kvamp - amplitud del vibrato.

42.6.4 AUTOR

John ffitich

Universidad de Bath, Codemist Ltd.

Bath, Reino Unido

1997

42.7 lorenz

ax, ay, az **lorenz** ks, kr, kb, kh, ix, iy, iz, iskip

42.7.1 DESCRIPCIÓN

Implementa el sistema de ecuaciones de Lorenz. El sistema de Lorenz es un sistema caótico dinámico que fue originalmente usado para simular el movimiento de partículas en corrientes de convección y en sistemas meteorológicos simplificados. Pequeñas diferencias en las condiciones iniciales conducían rápidamente a valores totalmente divergentes. Este efecto se llama a veces "efecto mariposa". Si una mariposa bate sus alas en Australia afectará al clima de Alaska. Este sistema es uno de los hitos en el desarrollo de la teoría del caos. Es útil como fuente caótica de audio o como fuente de modulación de baja frecuencia.

42.7.2 INICIALIZACIÓN

ix, iy, iz – coordenadas iniciales de la partícula.

iskip – se usa para saltarse algunos valores generados. Si *iskip* es 5, sólo uno de cada cinco valores generados se enviará a la señal. Esto es útil para generar notas de más alta frecuencia.

42.7.3 EJECUCIÓN

ksv – número de Prandtl o sigma.

krv – número de Rayleigh.

kbv – proporción entre la longitud y la anchura de la caja en la que las corrientes de convección son generadas.

kh – resolución para la aproximación de la ecuación diferencial. Puede ser usada para controlar la altura de los sistemas. Lo normal es que tome valores de .1 a .001.

Las ecuaciones son las siguientes:

$$\begin{aligned}x &=x + h*(s*(y - x)) \\y &=y + h*(-x*z + r*x - y) \\z &=z + h*(x*y - b*z)\end{aligned}$$

Los valores históricos de estos parámetros son:

$$\begin{aligned}ks &= 10 \\kr &= 28 \\kb &= 8/3\end{aligned}$$

42.7.4 EJEMPLO

```
instr 20
ksv      =      p4
krv      =      p5
kbv      =      p6
ax, ay, az lorenz ksv, krv, kbv, .01, .6, .6, .6, 1
endin
```

```
;score
;      start dur   S     R     V
i20   5      1    10    28    2.667
e
```

42.7.5 AUTOR

Hans Mikelson
Febrero 1999 (Nuevo en la versión 3.53)

42.8 planet

```
ax, ay, az      planet      kmass1, kmass2, ksep, ix, iy, iz, ivx, \\  
ivy, ivz, idelta, ifriction
```

42.8.1 DESCRIPCIÓN

planet simula una órbita planetaria en un sistema estelar binario. Los valores de salida son las coordenadas "x", "y" y "z" de la órbita. Es posible para el planeta alcanzar velocidades de escape causadas por el encuentro cercano con una estrella. Esto hará que el sistema sea un tanto inestable.

42.8.2 INICIALIZACIÓN

ix, iy, iz – coordenadas iniciales del planeta.

ivx, ivy, ivz – los valores iniciales de las componentes del vector de velocidad del planeta.

idelta – valor usado para aproximar la ecuación diferencial.

ifriction – valor de fricción, que puede ser usado para prevenir la explosión del planeta.

42.8.3 EJECUCIÓN

kmass1 – masa de la primera estrella.

kmass2 – masa de la segunda estrella.

ksep – distancia entre las dos estrellas.

ax, ay, az – las coordenadas de salida del planeta.

42.8.4 EJEMPLO

instr 1

```
idur      = p3
iamp      = p4
km1       = p5
km2       = p6
ksep      = p7
ix        = p8
iy        = p9
iz        = p10
ivx       = p11
ivy       = p12
ivz       = p13
ih        = p14
ifric     = p15
kamp      linseg    0, .002, iamp, idur-.004, iamp, .002, 0
ax, ay, az planet  km1, km2, ksep, ix, iy, iz, ivx, ivy, ivz, ih, ifric
outs             ax*kamp, ay*kamp
endin
```

; Sta	Dur	Amp	M1	M2	Sep	X	Y	Z	VX	VY	VZ	h	Frict
i1 0	1	5000	.5	.35	2.2	0	.1	0	.5	.6	-.1	.5	-0.1
i1 +	.	.	.5	0	0	0	.1	0	.5	.6	-.1	.5	0.1
i14	.3	2	0	.1	0	.5	.6	-.1	.5	0.0
i13	.3	2	0	.1	0	.5	.6	-.1	.5	0.1
i125	.3	2	0	.1	0	.5	.6	-.1	.5	1.0
i12	.5	2	0	.1	0	.5	.6	-.1	.1	1.0

42.8.5 AUTOR

Hans Mikelson
Diciembre 1998
Nuevo en la Versión 3.50

43 GENERADORES DE SEÑAL: RESÍNTESIS STFT (VOCODING)

43.1 pvoc, vpvoc

```
ar    pvoc  ktmpnt, kfmod, ifilcod [,ispecwp, iextractmode, ifreqlim, igatefn]  
ar    vpvoc ktmpnt, kfmod, ifile[, ispecwp]
```

43.1.1 DESCRIPCIÓN

La salida es una serie aditiva de ondas sinusoidales controladas individualmente usando un proceso de resíntesis con un **phase vocoder** (codificador vocal de fase).

43.1.2 INICIALIZACIÓN

ifilcod - entero o cadena de caracteres que indican un fichero de control derivado del análisis de una señal de audio. Un entero indica la extensión de un fichero *adsyn.m* o *pvoc.m*; una cadena de caracteres (entre comillas) proporciona el nombre de un fichero, opcionalmente con la ruta completa. Si no se especifica la ruta completa, el fichero se busca primero en el directorio actual y luego en el que indique la variable de entorno **SADIR** (si está definida). **adsyn** requiere los valores de los puntos de inflexión de los envolventes de amplitud y frecuencia, organizados para su resíntesis con los osciladores, mientras que **pvoc** requiere los mismos datos organizados para que puedan ser usados en la resíntesis con la Transformada Rápida de Fourier. El uso de la memoria depende del tamaño de los ficheros involucrados, que son leídos y mantenidos en memoria durante el proceso de computación, aunque compartidos por múltiples llamadas (ver también **lpread**).

ispecwp (opcional) - si no es 0, intenta preservar el envolvente espectral mientras su frecuencia es variada por *kfmod*. El valor por defecto es 0.

iextractmode (opcional) - determina si la extracción espectral será llevada a cabo y, de ser así, si los componentes que han cambiado en frecuencia por debajo de *ifreqlim* o por encima *ifreqlim* serán descartados o no. Si *iextractmode* toma un valor 1 hará que **pvadd** sintetice sólo aquellos componentes en los que la diferencia de frecuencia de los *frames* del análisis sea mayor que *ifreqlim*. Si *iextractmode* toma un valor de 2 hará que **pvadd** sintetice sólo aquellos componentes en los que la diferencia de frecuencia de los *frames* del análisis sea menor que *ifreqlim*. Los valores por defecto para *iextractmode* e *ifreqlim* son 0, en cuyo caso será ejecutado un proceso de resíntesis simple. Ver los ejemplos bajo **pvadd** para entender cómo usar la extracción espectral.

igatefn - es el número de una tabla de función que será aplicada a las amplitudes de las pistas del análisis antes de que tome lugar el proceso de resíntesis. Si *igatefn* es mayor que 0, las amplitudes de cada pista serán escaladas por *igatefn* mediante un simple proceso de mapeado. Primero, las amplitudes de todas las pistas en todos los frames del fichero de análisis completo se comparan para determinar el mayor valor de amplitud. Este valor se usa entonces para generar amplitudes normalizadas como índices en la función almacenada *igatefn*. La amplitud máxima apuntará a la última posición de la tabla y una amplitud 0

apuntará al primer elemento de la tabla. Los valores entre 0 y 1 apuntarán a los elementos intermedios. Ver ejemplos bajo **pvadd** para entender cómo usar esta técnica.

ifn (opcional) – tabla de función opcional que contiene información de control para **vpvoc**. Si *ifn* = 0, el control se deriva internamente de una unidad **tableseg** o **tablexseg** previa. El valor por defecto es 0. (Nuevo en la Versión 3.59).

43.1.3 EJECUCIÓN

pvoc implementa la reconstrucción de la señal usando el **phase vocoder** basado en la Transformada Rápida de Fourier (FFT). Los datos de control surgen de un fichero de análisis, realizado previamente a una frecuencia conocida. El instante (puntero temporal) a lo largo del fichero es especificado por *ktimpnt*, que representa una duración en segundos. *ktimpnt* debe ser siempre positivo, pero puede moverse hacia delante o hacia atrás en el tiempo, ser estacionario o discontinuo, ya que funciona como un puntero al fichero de análisis. *kfmod* es un factor de transposición que trabaja a frecuencia de control (k-): un valor de 1, por ejemplo, indica ausencia de transposición, 1.5 transportaría el sonido una quinta justa ascendente, mientras que .5 lo transportaría una octava descendente.

Esta implementación de **pvoc** fue escrita por Dan Ellis y está basada, en parte, en el sistema de Mark Dolson, aunque el concepto del pre-análisis es original. Las opciones de extracción espectral y el direccionamiento de amplitudes (nuevas en la versión 3.56) fueron añadidas por Richard Karpen y están basadas en funciones del SoundHack de Tom Erbe.

vpvoc es idéntico a **pvoc** excepto en que toma el resultado de una llamada previa a **tableseg** o **tablexseg**, y usa la tabla de función resultante (pasada internamente al mismo **vpvoc**) como envolvente sobre las magnitudes de los datos de los canales del análisis. El resultado es, entonces, un envolvente espectral. El tamaño de la tabla de función usada en **tableseg** deberá ser "*tamaño-frame/2*", donde *tamaño-frame* es el número de pistas en el fichero de análisis del **phase vocoder** que se están usando con **vpvoc**. Cada posición en la tabla será usada para normalizar una única pista del análisis. Usando diferentes funciones para *ifn1*, *ifn2*, etc.. en **tableseg**, se puede conseguir que el envolvente espectral varíe en el tiempo dinámicamente.

43.1.4 EJEMPLO

El siguiente ejemplo, que usa **vpvoc**, muestra el uso de funciones tales como:

```
f 1 0 256 5 .001 128 1 128 .001
f 2 0 256 5 1 128 .001 128 1
f 3 0 256 7 1 256 1
```

para escalar las amplitudes de frames separados del análisis

```
ptime      line      0, p3,3      ; puntero temporal, en segundos, dentro del
              ; fichero
              tablexseg 1, p3*.5, 2, p3*.5, 3
apv        vpvoc      ptime,1, "pvoc.file"
```

El resultado sería un envolvente espectral variable en el tiempo, aplicado al análisis de los datos del codificador vocal de fase. Debido a que así amplificamos o atenuamos la cantidad de señal en las frecuencias que concuerdan con las amplitudes escaladas por estas funciones, esto tiene el mismo efecto que aplicar filtros muy exactos a la señal. En este ejemplo, la primera tabla tendría el efecto de un filtro

pasa banda, gradualmente siendo rechazado en una banda durante la mitad de la duración de la nota, y yendo luego hacia la “no modificación” de las magnitudes durante la segunda mitad.

42.1.5 AUTORES

Dan Ellis (**pvoc**)
Seattle
1997

Richard Karpen (**vpvoc**)
Washington

43.2 pvread, pvburead, pvinterp, pvcross, tableseg, tablexseg

kfr, kamp	pvread	ktimpnt, ifile, ibin
	pvburead	ktimpnt, ifile
ar	pvinterp	ktimpnt, kfmmod, ifile, kfregscale1, kfregscale2, \\kampscale1, kampscale2, kfreginterp, kampinterp
ar	pvcross	ktimpnt, kfmmod, ifile, kamp1, kamp2[, ispecwp]
	tableseg	ifn1, idur1, ifn2[, idur2, ifn3[...]]
	tablexseg	ifn1, idur1, ifn2[, idur2, ifn3[...]]

43.2.1 DESCRIPCIÓN

pvread lee un fichero de **pvoc** y devuelve la frecuencia y la amplitud de un único canal o pista del análisis. Los valores devueltos pueden ser usados en cualquier otro lugar dentro de un instrumento de Csound. Por ejemplo, se los puede usar como argumentos de un oscilador para sintetizar un sólo componente de la señal analizada, o bien se puede usar un banco de opcodes **pvread** para resintetizar el sonido analizado y luego pasar las frecuencias y amplitudes a un banco de osciladores para su síntesis aditiva final.

pvburead lee un fichero de **pvoc** y dispone los datos para que sean usados con cualquier opcode **pvinterp** o **pvcross** que aparezca a continuación en el mismo instrumento (exactamente de la misma manera en que trabajan juntos **lpread** y **lpreson**). Los datos se pasan mediante un proceso interno y la unidad no tiene resultado de salida propio.

pvinterp y **pvcross** permiten el procesado conjunto de dos ficheros de análisis del **phase vocoder** antes de proceder a su resíntesis (que también realizan ellos mismos). Ambas unidades reciben los datos de uno de los ficheros al que previamente se ha accedido con **pvburead**. El otro fichero es leído por las unidades **pvinterp** y/o **pvcross**. Debido a que cada una de estas unidades tiene su propio puntero de tiempo, los ficheros de análisis pueden ser leídos en diferentes direcciones y a velocidades distintas. **pvinterp** no permite el uso de *ispecwp* en la misma forma que lo hacen **pvoc** y **vpvoc**.

pvinterp interpola los valores de amplitud y frecuencia, pista a pista, de dos ficheros de análisis del **phase vocoder** (a uno de los cuales se ha accedido previamente con **pvburead**, y el otro se especifica en la lista de argumentos), permitiendo transiciones definidas por el usuario entre dos sonidos analizados (*morphing*). También permite la normalización de las amplitudes y las frecuencias de cada fichero por separado antes de que los valores interpolados sean calculados y enviados a las rutinas de resíntesis. El argumento *kfmmod* en **pvinterp** efectúa la normalización de los valores de frecuencia, después de realizar la normalización y subsiguiente interpolación previas, así que, en realidad, actúa como si fuera un factor de normalización general para todos los nuevos componentes de frecuencia.

pvcross aplica las amplitudes de un fichero de análisis del **phase vocoder** a los datos de un segundo fichero y luego realiza su resíntesis. Los datos se pasan, como arriba, mediante una llamada previa de **pvburead**. Los dos argumentos de control (k-) de amplitud se usan para normalizar las amplitudes de cada fichero por separado, antes de que sean unidos y usados en la resíntesis (ver abajo una explicación más detallada). Las frecuencias del primer fichero no se usan para nada en este proceso. Este opcode proporciona un método simple de síntesis cruzada, aplicando las amplitudes del espectro de una señal a las frecuencias de una segunda. A diferencia de **pvinterp** y **pvcross**, permite usar *ispecwp* como en **pvoc** y **vpvoc**.

tableseg y **tablexseg** son similares a **linseg** y **expseg** respectivamente, con la diferencia de que interpolan los valores sucesivos contenidos en una tabla de función almacenada. El resultado es una nueva tabla de función que se pasa, mediante un proceso interno, a cualquier opcode **vpvoc** que venga a continuación (de la misma manera en que trabajan juntos **lpread** y **lpreson**). Los usos de estos opcodes se describirán más adelante, bajo la entrada de **vpvoc**.

43.2.2 EJECUCIÓN

ifile es el número de **pvoc** (n en pvoc.n) o el nombre entre comillas del fichero de análisis creado con **pvanal** (ver **pvoc**).

kfreq y *kamp* son los valores devueltos por el opcode **pvread**. Estos valores, leídos de un fichero de análisis del **phase vocoder**, representan los valores de frecuencia y amplitud de un único canal de análisis, especificado en el argumento *ibin*. La interpolación entre los cuadros (frames) del análisis se realiza a frecuencia de control (k-) y depende de la velocidad y la dirección especificadas por *ktimpnt*.

ktimpnt, *kfmod*, y *ispecwp*, usados en **pvread** y **vpvoc**, son exactamente iguales que para **pvoc** (ver la descripción anterior de **pvinterp** para el uso especial de *kfmod*).

ibin es el número del canal de análisis del que se toman las frecuencias, en Hz, y las amplitudes.

kfreqscale1, *kfreqscale2*, *kampscale1*, y *kampscale2* se usan en **pvinterp** para normalizar las frecuencias y las amplitudes almacenadas en cada *frame* del fichero de análisis del **phase vocoder**. *kfreqscale1* y *kampscale1* normalizan las frecuencias y amplitudes de los datos del fichero leído mediante una llamada previa a **pvbufread** (estos datos se pasarán internamente a una unidad **pvinterp**). *kfreqscale2* y *kampscale2* normalizan las frecuencias y amplitudes leídas por dicha unidad del fichero indicado con *ifile* en la lista de argumentos de **pvinterp**. Usando estos argumentos es posible ajustar estos valores antes de aplicar la interpolación. Por ejemplo, si el primer fichero suena más fuerte que el segundo, puede ser deseable escalar las amplitudes del primero a un valor más bajo, o bien las del segundo a un valor más alto, antes de realizar la interpolación. Así mismo, se pueden ajustar las frecuencias de cada uno para aproximarlas (o justo lo contrario, según se quiera) antes de ejecutar el proceso de interpolación.

kfreqinterp y *kampinterp*, usados en **pvinterp**, determinan la distancia de interpolación entre los valores de un fichero de análisis del **phase vocoder** y los valores de otro. Cuando el valor de *kfreqinterp* es 0, los valores de frecuencia serán los del primer fichero (leído por **pvbufread**), después de la normalización indicada por el argumento *kfreqscale1*. Cuando el valor de *kfreqinterp* es 1, los valores de frecuencia serán los del segundo fichero (leídos por la unidad **pvinterp** misma), después de normalizarlos según *kfreqscale2*. Cuando *kfreqinterp* está entre 0 y 1, los valores de frecuencia serán calculados, pista a pista, como el porcentaje entre cada par de frecuencias (en otras palabras, *kfreqinterp*=0.5 causará que los valores de frecuencia estén a mitad de camino entre los valores de la serie de datos del primer fichero y la serie de los del segundo). *kampinterp1* y *kampinterp2* trabajan de la misma manera sobre las amplitudes de los dos ficheros. Ya que son argumentos de tipo k-, los porcentajes pueden cambiar en tiempo de ejecución, haciendo posible crear muchos tipos de transición entre dos sonidos.

ifn1, *ifn2*, *ifn3*, etc... en **tableseg** y **tablexseg** son tablas de función almacenadas, creadas por una **sentencia f** de la partitura. *ifn1*, *ifn2*, y siguientes, deben ser todas del mismo tamaño.

idur1, *idur2*, etc...en **tableseg** y **tablexseg** son las duraciones durante las cuales tendrá lugar la interpolación de una tabla a la siguiente.

43.2.3 EJEMPLO

El ejemplo siguiente muestra el uso de **pvread** para sintetizar un único componente de un fichero de análisis del **phase vocoder**. Debe ser recalcado que los resultados *kfreq* y *kamp* pueden ser usados para cualquier tipo de síntesis, filtrado, u otro tipo de procesado.

```
ptime          line 0, p3, 3
kfreq, kamp    pvread ktime, "pvoc.file", 7 ; lee
               ; datos de la pista 7 del análisis.
asig           oscili kamp, kfreq, 1 ; la función 1
               ; es una onda sinusoidal almacenada.
```

El ejemplo siguiente muestra el uso de **pvbufread** con **pvinterp** para "interpolarse" el sonido de un oboe y el de un clarinete. El valor de *kinterp* devuelto por **linseg** se usa para determinar la evolución de la transición entre los dos sonidos. La interpolación de las frecuencias y las amplitudes se controla mediante un mismo factor en este ejemplo, pero para conseguir otros efectos puede ser interesante no tenerlos sincronizados de esta manera. En este ejemplo, el sonido empezará siendo el de un clarinete y se transformará en el de un oboe, volviendo luego al sonido de clarinete original.

El valor de *kfreqscale2* es 1.065 porque el oboe en este caso está un semitono más alto que el clarinete y con este valor conseguimos aproximarlos casi a la misma altura. El valor de *kampscale2* es .75 porque el clarinete analizado sonaba un poco más fuerte que el oboe. La configuración de estos dos parámetros hacen que la transición en este caso sea muy suave, pero estos ajustes no tienen porque ser siempre necesarios o deseados.

```
ptime1         line 0, p3, 3.5 ; usado como indice en el fichero "oboe.pvoc"
ptime2         line 0, p3, 4.5 ; usado como indice en el fichero "clar.pvoc"
kinterp        linseg 1, p3*.15, 1, p3*.35, 0, p3*.25, 0, p3*.15, 1, p3*.1, 1
               pvbufread ktime1, "oboe.pvoc"
apv            pvinterp ktime2,1,"clar.pvoc",1,1.065,1,.75,1-kinterp,1-kinterp
```

El siguiente es un ejemplo del uso de **pvbufread** con **pvcross**. En este caso las amplitudes usadas en la resíntesis cambian gradualmente de las del oboe a las del clarinete. Las frecuencias, por supuesto, permanecen constantes durante todo el proceso y toman el valor de aquellas del clarinete, ya que **pvcross** no usa los datos de frecuencia del fichero leído por **pvbufread**.

```
ptime1         line 0, p3, 3.5 ; usado como indice en el fichero "oboe.pvoc"
ptime2         line 0, p3, 4.5 ; usado como indice en el fichero "clar.pvoc"
kcross         expon .001, p3, 1
               pvbufread ktime1, "oboe.pvoc"
apv            pvcross ktime2, 1, "clar.pvoc", 1-kcross, kcross
```

En el siguiente ejemplo usamos **vpvoc** con las funciones:

```
f 1 0 256 5 .001 128 1 128 .001
f 2 0 256 5 1 128 .001 128 1
f 3 0 256 7 1 256 1
```

para normalizar las amplitudes de las pistas separadas del análisis.

```
ptime          line 0, p3,3 ; puntero de tiempo, en segundos, dentro del fichero
               tablexseg 1, p3*.5, 2, p3*.5, 3
apv            vpvoc ktime,1, "pvoc.file"
```

El resultado sería un envolvente espectral que varía en el tiempo, aplicado a los datos del análisis del **phase vocoder**. Debido a que este proceso amplifica o atenúa la señal en las frecuencias emparejadas según las amplitudes normalizadas por estas funciones, se produce el mismo efecto que al aplicar filtros muy exactos a la señal. En este ejemplo, la primera tabla tendría el efecto de un filtro pasa banda, convirtiéndose gradualmente en un filtro para banda en la primera mitad de la duración de la nota, y luego, en la segunda mitad, iría hacia la "no modificación" progresiva de las amplitudes.

43.2.4 AUTOR

Richard Karpen
Seattle, Wash
1997

43.3 pvadd

```
ar    pvadd    ktmpnt, kfmod, ifilcod, ifn, ibins[, ibinoffset, \\
        ibinincr, iextractmode, ifreqlim, igatefn]
```

43.3.1 DESCRIPCIÓN

pvadd lee un fichero de **pvoc** y usa sus datos para realizar un proceso de síntesis aditiva usando una serie interna de osciladores con interpolación. El usuario proporciona la forma de onda (normalmente un período de una onda sinusoidal) y puede escoger qué pistas del análisis usar en la resíntesis.

43.3.2 INICIALIZACIÓN

ifilcod - entero o cadena de caracteres que indican un fichero de control derivado del análisis de una señal de audio. Un entero indica la extensión de un fichero *adsyn.m* o *pvoc.m*; una cadena de caracteres (entre comillas) proporciona el nombre de un fichero, opcionalmente con la ruta completa. Si no se especifica la ruta completa, el fichero se busca primero en el directorio actual y luego en el que indique la variable de entorno **SADIR** (si está definida). Los ficheros de control de **pvoc** están organizados para que puedan ser usados en la resíntesis con la Transformada Rápida de Fourier. El uso de la memoria depende del tamaño de los ficheros involucrados, que son leídos y mantenidos en memoria durante el proceso de computación, aunque compartidos por múltiples llamadas (ver también **lpread**).

ifn - número de la tabla de una función almacenada que contiene una onda sinusoidal.

ibins - número de pistas que serán usadas en la resíntesis (cada pista cuenta como un oscilador en la resíntesis).

ibinoffset - es la primera pista que se va a usar (es opcional y el valor por defecto es 0).

ibinincr - define el incremento con el cual **pvadd** cuenta cada una de las *ibins* (pistas) de la resíntesis, empezando por la pista especificada por *ibinoffset* (una explicación más detallada más adelante).

iextractmode (opcional) - determina si la extracción espectral será llevada a cabo y, de ser así, si los componentes que han cambiado en frecuencia por debajo de *ifreqlim* o por encima *ifreqlim* serán descartados o no. Si *iextractmode* toma un valor 1 hará que **pvadd** sintetice sólo aquellos componentes en los que la diferencia de frecuencia de los *frames* del análisis sea mayor que *ifreqlim*. Si *iextractmode* toma un valor de 2 hará que **pvadd** sintetice sólo aquellos componentes en los que la diferencia de frecuencia de los *frames* del análisis sea menor que *ifreqlim*. Los valores por defecto para *iextractmode* e *ifreqlim* son 0, en cuyo caso será ejecutado un proceso de resíntesis simple. Ver los ejemplos más abajo.

igatefn - es el número de una tabla de función que será aplicada a las amplitudes de las pistas del análisis antes de que tome lugar el proceso de resíntesis. Si *igatefn* es mayor que 0, las amplitudes de cada pista serán escaladas por *igatefn* mediante un simple proceso de mapeado. Primero, las amplitudes de todas las pistas en todos los frames del fichero de análisis completo se comparan para determinar el mayor valor de amplitud. Este valor se usa entonces para generar amplitudes normalizadas como índices en la función almacenada *igatefn*. La amplitud máxima apuntará a la última posición de la tabla y una amplitud 0 apuntará al primer elemento de la tabla. Los valores entre 0 y 1 apuntarán a los elementos intermedios. Ver los ejemplos más abajo.

43.3.3 EJECUCIÓN

ktimpnt y *kfmod* se usan de la misma manera que en **pvoc**.

43.3.4 EJEMPLO

```
ptime      line 0, p3, p3
asig       pvadd ktime, 1, "oboe.pvoc", 1, 100, 2
```

En el ejemplo de arriba, *ibins* es 100 y *ibinoffset* es 2. Usando estos valores, la resíntesis contendrá 100 componentes (pistas) empezando a contar desde la pista 2 (que se tomará como pista 0, es decir, pista inicial). Esto significa que la resíntesis se realizará usando las pistas de la 2 a la 101 inclusive. Normalmente, es buena idea empezar en la pista 1 ó 2, ya que a menudo las pistas 0 o 1 contienen datos que no son ni necesarios ni útiles para obtener un resultado limpio en el proceso de resíntesis.

```
ptime      line 0, p3, p3
asig       pvadd ktime, 1, "oboe.pvoc", 1, 100, 2, 2
```

Este otro ejemplo es idéntico al anterior pero añadiendo el valor 2 como argumento opcional *ibinincr*. Esto también produce 100 componentes en la resíntesis, pero ahora **pvadd** contará las pistas de dos en dos. Es decir, usará las pistas 2, 4, 6, 8, 10 etc. Para valores *ibins*=10, *ibinoffset*=10, y *ibinincr*=10, por ejemplo, **pvadd** usaría las pistas 10, 20, 30, 40, etc hasta 100 inclusive.

A continuación presentamos un ejemplo que usa la extracción espectral. En dicho ejemplo *iextractmode* es 1 y *ifreqlim* 9. Esto hará que **pvadd** sintetice sólo aquellas pistas donde la desviación de frecuencia, calculada mediante la media de 6 frames sucesivos, sea mayor que 9.

```
ptime      line 0, p3, p3
asig       pvadd ktime, 1, "oboe.pvoc", 1, 100, 2, 2, 1, 9
```

Si *iextractmode* fuera 2 el ejemplo anterior, sólo serían sintetizadas aquellas pistas con una desviación de frecuencia menor que 9. Si se afina correctamente, esta técnica puede ser usada para separar los componentes "musicales" (de frecuencia determinada) de los componentes de ruido (de frecuencia aleatoria) del espectro. En la práctica, esto depende en gran parte del tipo de sonido, la calidad de la grabación y digitalización y también del tamaño de la ventana de análisis y del incremento de frames.

El siguiente es un ejemplo que usa el direccionamiento de amplitudes. El 2 final en la lista de argumentos apunta a *f2* en la partitura.

```
asig       pvadd ktime, 1, "oboe.pvoc", 1, 100, 2, 2, 0, 0, 2
```

Supongamos que la partitura de lo anterior fuera:

```
f2 0 512 7 0 256 1 256 1
```

Entonces aquellas pistas con amplitudes de al menos el 50% del valor máximo permanecerían inalteradas, mientras que aquellas con amplitudes menores del 50% del máximo serían escaladas hacia abajo. En este caso cuanto más baja la amplitud más severo el proceso de escalado. Pero supongamos que la partitura fuera:

```
f2 0 512 5 1 512 .001
```

En este caso, las amplitudes más bajas quedarían inalteradas y las mayores serían escaladas hacia abajo, poniendo el sonido "boca abajo" en términos del espectro de amplitudes. Las funciones pueden ser tan complejas como se quiera. Tan sólo recuerda que los valores de amplitud normalizados del análisis son en sí mismos los índices de la tabla de función.

Finalmente, ambas técnicas de extracción espectral y redireccionamiento de amplitudes pueden ser usadas juntas. El siguiente ejemplo sintetizará sólo aquellos componentes con una desviación de frecuencia de menos de 5 Hz por frame y escalará las amplitudes de acuerdo con f2.

```
asig pvadd ktime, 1, "oboe.pvoc", 1, 100, 1, 1, 2, 5, 2
```

43.3.5 CONSEJOS ÚTILES

Usando varias unidades **pvadd** juntas, se pueden resaltar gradualmente partes diferentes del sonido en la resíntesis, pudiéndose obtener muchos efectos de filtrado. El autor usa **pvadd** para sintetizar una pista cada vez, lo que proporciona un grado de control total sobre cada una de las pistas de la resíntesis por separado.

Si cualquier combinación de *ibins*, *ibinoffset*, y *ibinincr*, crea una situación en la que se pide a **pvadd** que use un número de pista mayor que el número de pistas reales del análisis, usará todas las pistas disponibles, sin ningún mensaje de error por su parte. Así que para usar todas las pistas, lo único que tienes que hacer es darle a *ibins* un valor suficientemente grande (por ejemplo, 2000).

Por cierto, observa que tendrás que normalizar las amplitudes en un rango de 10 a 100.

43.3.6 AUTOR

Richard Karpen
Seattle, Wash
1997 (Nuevo en la versión 3.48)

44 GENERADORES DE SEÑAL: RESÍNTESIS LPC

44.1 lpread, lpreson, lpfreson

krmsr,krms0,kerr,kcps	lpread	ktimpnt, ifilcod[, inpoles[, ifrmrate]]
ar	lpreson	asig
ar	lpfreson	asig, kfrqratio

44.1.1 DESCRIPCIÓN

Estas unidades, usadas como un par **read/reson**, se sirven de un fichero de control que contiene los datos de variación de los coeficientes del filtro a lo largo del tiempo, proporcionando un método para modificar dinámicamente el espectro de una señal de audio.

44.1.2 INICIALIZACIÓN

ifilcod - entero o cadena de caracteres que indican un fichero de control (con los coeficientes de reflexión y cuatro valores de parámetro), derivado de un análisis espectral predictivo lineal (**lp**) de n-polos realizado sobre una señal de audio. Un valor entero indica la extensión de un fichero "*lp.m*"; una cadena de caracteres (entre comillas) proporciona un nombre de fichero, opcionalmente con su ruta completa. Si no se indica la ruta completa, el fichero se busca primero en el directorio actual y luego en el que especifica la variable de entorno SADIR (si está definida). El uso de la memoria depende del tamaño del fichero, que es mantenido enteramente en memoria durante el proceso de computación, aunque compartido por varias llamadas (ver también **adsyn**, **pvoc**).

inpoles, *ifrmrate* (opcional) - número de polos, y frecuencia de cuadros (*frames*) por segundo del análisis **lpc**. Estos argumentos son requeridos sólo cuando el fichero de control no tiene cabecera. Si la tiene, dichos argumentos son ignorados. El valor por defecto es 0 para ambos.

44.1.3 EJECUCIÓN

lpread accede a un fichero de control que contiene frames de información ordenados en el tiempo, cada uno conteniendo los coeficientes de un filtro n-polar derivados de un análisis predictivo lineal realizado a una señal fuente, a intervalos regulares de tiempo (por ejemplo, cada centésima de segundo), más cuatro valores de parámetros, que son:

```
krmsr - rms del análisis residual,  
krms0 - rms de la señal original,  
kerr - la señal de error normalizada,  
kcps - altura en Hz.
```

lpread obtiene sus valores de un fichero de control, de acuerdo con el valor de entrada *ktimpnt* (expresado en segundos). Si *ktimpnt* concuerda con la frecuencia del análisis, resultará un proceso de síntesis sin modificación temporal; si señala una frecuencia más rápida, más lenta, o variable, ocasionará alteraciones temporales. Cada período de control (k-), **lpread** interpola los valores de *frames* adyacentes para

determinar más exactamente los valores de los parámetros y los coeficientes del filtro (pasados internamente a un opcode **lpreson** posterior).

La señal de error *kerr* (que toma valores entre 0 y 1), derivada del análisis predictivo, refleja la naturaleza determinista o estocástica de la fuente analizada. Tomará valores bajos para sonidos melódicos (es decir, periódicos) o valores altos para ruidos (es decir, sonidos aperiódicos). Durante el proceso de síntesis, el valor de la señal de error puede ser usado para determinar la naturaleza de la función controladora del opcode **lpreson**, por ejemplo, escogiendo arbitrariamente entre señales de entrada ruidosas o no ruidosas, o incluso determinando una mezcla de las dos. En la síntesis normal de voz, la entrada de frecuencias en **lpreson** se compone de una señal periódica de banda ancha o de un tren de ondas pulso, derivados de una unidad como **buzz**, por ejemplo, y la fuente de ruido puede conseguirse usando una unidad **rand**. Sin embargo, cualquier señal de audio puede servir de función controladora, asumiendo el análisis que tendrá una curva de respuesta plana.

lpfreson es un opcode **lpreson** con desplazamiento de formantes, en el que *kfrqratio* es la proporción (en Hz) entre las posiciones de los formantes desplazados y los originales. Esto permite modelos de síntesis en los que la fuente de sonido puede cambiar su "extensión" acústica aparente. **lpfreson** con un valor *kfrqratio* = 1 es idéntico a **lpreson**.

Generalmente, **lpreson** proporciona un medio mediante el cual la variación en el tiempo del contenido de una señal de audio compleja, así como la evolución de su curva espectral, pueden ser controladas por el contenido dinámico espectral de otra. Puede haber cualquier número de pares **lpread/lpreson** (o **lpfreson**) en un mismo instrumento u orquesta ya que estos opcodes pueden leer independientemente del mismo o de diferentes ficheros de control.

44.2 lpslot, lpinterp

lpslot	islot
lpinterp	islot1, islot2, kmix

44.2.1 DESCRIPCIÓN

Estos opcodes realizan un proceso de interpolación entre dos ficheros de análisis **lpc**.

44.2.2 INICIALIZACIÓN

islot - número de posiciones a seleccionar [$0 < islot < 20$].

lpslot selecciona la posición que usarán los subsiguientes opcodes **lp**. Esta es la forma en la que se cargan y referencian varios análisis al mismo tiempo.

islot1 - posición en que se almacenó el primer análisis.

islot2 - posición en que se almacenó el segundo análisis.

kmix - valor de la mezcla entre los dos análisis. Debe estar entre 0 y 1. Un valor 0 significa que se tomarán valores sólo del primer análisis, mientras que un valor 1 indica que se tomarán valores sólo del segundo. Cualquier valor intermedio producirá la interpolación en el filtrado.

lpinterp calcula una nueva serie de polos interpolando los datos de los dos análisis. Dichos polos se guardarán en la posición que indique en ese momento **lpslot** y serán usados por el siguiente opcode **lpreson** que aparezca en la orquesta.

44.2.3 EJEMPLO

He aquí una orquesta típica que usa estos opcodes:

```
ipower      init 50000                ; define el generador de sonido
ifreq       init 440
asrc        buzz ipower,ifreq,10,1
ktime       line 0,p3,p3              ; define la linea temporal
            lpslot 0                  ; lee los datos de los polos de square.pol
krmsr,krms0,kerr,kcps lpread ktime,"square.pol"
            lpslot 1                  ; lee los datos de los polos detriangle.pol
krmsr,krms0,kerr,kcps lpread ktime,"triangle.pol"
kmix        line 0,p3,1               ; calcula el resultado de la mezcla
            lpinterp 0,1,kmix        ; y equilibra la potencia
ares        lpreson asrc
aout        balance ares,asrc
out         out aout
```

44.2.4 AUTOR

Mark Resibois
Bruselas
1996

45 GENERADORES DE SEÑAL: GENERADORES ALEATORIOS (RUIDO)

45.1 rand, randh, randi

kr	rand	xamp [, iseed][, isize]
kr	randh	kamp, kcps[, iseed][, isize]
kr	randi	kamp, kcps[, iseed][, isize]
ar	rand	xamp [, iseed][, isize]
ar	randh	xamp, xcps[, iseed][, isize]
ar	randi	xamp, xcps[, iseed][, isize]

45.1.1 DESCRIPCIÓN

La salida es una serie controlada de números aleatorios entre $+amp$ y $-amp$.

45.1.2 INICIALIZACIÓN

iseed (opcional) - valor germinal para la fórmula pseudo-aleatoria recursiva. Un valor entre 0 y +1 producirá una salida inicial de $kamp * iseed$. Un valor mayor que 1 será usado directamente, sin normalizar. Un valor negativo omitirá la reinicialización del valor germinal. El valor por defecto es .5.

imeth - si es 0, se genera un número de 16 bits. Si es distinto de 0, se genera uno de 32 bits. El valor por defecto es 0.

45.1.3 EJECUCIÓN

La fórmula interna de números pseudo-aleatorios produce valores uniformemente distribuidos a lo largo del rango de $kamp$ a $-kamp$. **rand** generará así un ruido blanco uniforme, con un valor inicial igual a " $kamp/raíz\ de\ 2$ ".

Las restantes unidades producen ruido de banda limitada. El parámetro *kcps* permite al usuario especificar si los números aleatorios se generarán a una frecuencia menor que la de muestreo o la de control. **randh** mantendrá el mismo valor para el período completo de cada ciclo especificado; **randi** producirá una interpolación lineal entre cada nuevo número aleatorio y el siguiente.

seed proporciona el valor germinal del generador de números pseudo-aleatorios, especificado por *ival*, para el instrumento en el que será usado. **seed** sólo puede ser usado una vez en un mismo instrumento.

45.1.4 EJEMPLO

```
il = octpch(p5) ; center pitch, to be modified
k1 randh 1,10 ;10 time/sec by random displacements up to 1 octave
a1 oscil 5000, cpsoct(il+k1), 1
```

45.2 generadores de ruido de clase x

ir	linrand	krange
kr	linrand	krange
ar	linrand	krange
ir	trirand	krange
kr	trirand	krange
ar	trirand	krange
ir	exprand	krange
kr	exprand	krange
ar	exprand	krange
ir	bexprnd	krange
kr	bexprnd	krange
ar	bexprnd	krange
ir	cauchy	kalpha
kr	cauchy	kalpha
ar	cauchy	kalpha
ir	pcauchy	kalpha
kr	pcauchy	kalpha
ar	pcauchy	kalpha
ir	poisson	klambda
kr	poisson	klambda
ar	poisson	klambda
ir	gauss	krange
kr	gauss	krange
ar	gauss	krange
ir	weibull	ksigma, ktau
kr	weibull	ksigma, ktau
ar	weibull	ksigma, ktau
ir	betarand	krange, kalpha, kbeta
kr	betarand	krange, kalpha, kbeta
ar	betarand	krange, kalpha, kbeta
ir	unirand	krange
kr	unirand	krange
ar	unirand	krange

45.2.1 DESCRIPCIÓN

Todos los siguientes opcodes pueden operar a frecuencias de inicialización, control o audio.

linrand *krange* - generador de números aleatorios con distribución lineal. *krange* es el rango de los números aleatorios (de 0 a *krange*). Devuelve sólo números positivos.

trirand *krange* - igual que el anterior excepto en que la salida puede estar compuesta de números positivos y negativos.

exprand *krange* - generador de números aleatorios con distribución exponencial. *krange* es el rango de los números aleatorios (de 0 a *krange*). Devuelve sólo números positivos.

bexprnd *krange* - igual que el anterior excepto en que la salida puede estar compuesta de números positivos y negativos, con una distribución exponencial.

cauchy *kalpha* - generador de números aleatorios con la distribución de Cauchy. *kalpha* controla la dispersión desde 0 (mayor *kalpha* = mayor dispersión). Devuelve números positivos y negativos.

pcauchy *kalpha* - igual que el anterior pero limitando la salida a números positivos.

poisson *klambda* - generador de números aleatorios con la distribución de Poisson. *klambda* es el término medio de la distribución. Devuelve sólo números positivos.

gauss *krange* - generador de números aleatorios con la distribución de Gauss. *krange* es el rango de los números aleatorios (de *-krange* a *krange*). Devuelve números positivos y negativos.

weibull *ksigma*, *ktau* - generador de números aleatorios con la distribución de Weibull. La dispersión de la distribución se normaliza según *ksigma*. Si *ktau* es mayor que 1, se favorecen los valores cercanos a *ksigma*, si es menor, se favorecen valores pequeños, y si es igual a 1, la distribución es exponencial. Devuelve solamente números positivos.

betarand *krange*, *kalpha*, *kbeta* - generador de números aleatorios con distribución beta. *krange* es el rango de los números aleatorios (de 0 a *krange*). Si *kalpha* es menor que 1, los valores más pequeños favorecen resultados cercanos a 0. Si *kbeta* es menor que 1, los valores más pequeños favorecen resultados cercanos a *krange*. Si ambos, *kalpha* y *kbeta* son iguales a 1, obtenemos una distribución uniforme. Si ambos son mayores que 1, tenemos un tipo de distribución gaussiana. Devuelve sólo números positivos.

unirand *krange* - generador de números aleatorios con distribución uniforme. *krange* es el rango de los números aleatorios (de 0 a *krange*).

Para una explicación más detallada, ver:

1. C. Dodge - T.A. Jerse 1985. Computer music. Schirmer books. pp.265 - 286
2. D. Lorrain. A panoply of stochastic cannons. In C. Roads, ed. 1989. Music machine . Cambridge, Massachusetts: MIT press, pp. 351 - 379.

45.2.2 EJEMPLO

```
a1  trirand      32000 ; Ruido con distribución triangular
k1  cauchy       10000 ; Ruido de control con distribución de Cauchy
i1  betarand     30000, .5, .5 ; valor aleatorio de inicialización, distrib. Beta
```

45.2.3 NOMBRES DESECHADOS

Estos opcodes empezaban antes por las letras "i", "k" o "a" para indicar la frecuencia a la que operaban. Estos nombres fueron desechados en la versión 3.49 de Csound. Deberá usarse los nuevos opcodes, ya que los anteriores no funcionarán.

45.2.4 AUTOR

Paris Smaragdis
MIT, Cambridge
1995

46 CONTROL DE TABLAS DE FUNCIÓN: CONSULTAS A TABLA

46.1 **ftlen, ftlptim, ftsr, nsamp**

ftlen	(x)	(sólo argumentos i-)
ftlptim	(x)	(sólo argumentos i-)
ftsr	(x)	(sólo argumentos i-)
nsamp	(x)	(sólo argumentos i-)

donde el argumento entre paréntesis puede ser a su vez una expresión.

46.1.1 DESCRIPCIÓN

Estos convertidores de valor devuelven información sobre una tabla de función almacenada. El resultado puede usarse a su vez como término en otra expresión.

46.1.2 EJECUCIÓN

ftlen(x) - devuelve el tamaño (número de elementos, sin contar el elemento límite añadido) de la tabla de función x. Aunque la mayoría de las unidades que acceden a una tabla almacenada tienen automáticamente en cuenta su tamaño (de manera que las tablas pueden tener una longitud arbitraria), esta sentencia devuelve dicho tamaño en caso de que se necesite para algo.

ftlptim(x) - devuelve el instante inicial (expresado en segundos) del bucle de repetición de una tabla de función x. Esto proporciona la duración del ataque y el decaimiento de una muestra de sonido grabada directamente, antes del comienzo del bucle de repetición. Devuelve 0 (y un mensaje de aviso) si la muestra no contiene puntos de bucle.

ftsr(x) - devuelve la frecuencia de muestreo de una tabla generada con **GEN01** o **GEN22**. La frecuencia de muestreo viene determinada por la cabecera del fichero original. Si el fichero original no tiene cabecera, o la tabla no fue creada por ninguna de esas dos GENS, **ftsr** devuelve 0. Nuevo en la versión 3.49 de Csound.

nsamp(x) - devuelve el número de muestras almacenadas por **GEN01** o **GEN23** en la tabla de función x. Es útil cuando una muestra es más corta que la tabla de función que la contiene (cuyo tamaño será una potencia de 2). Nuevo en la versión 3.49 de Csound.

46.1.3 AUTORES

Barry Vercoe
M.I.T., Cambridge, Mass
1997

Gabriel Maldonado (**ftsr, nsamp**)
Italia
Octubre, 1998

46.2 **tableng**

```
ir   tableng   ifn
kr   tableng   kfn
```

46.2.1 DESCRIPCIÓN

Devuelve la longitud de una tabla.

46.2.2 INICIALIZACIÓN

ifn, *kfn* - número de la tabla a consultar.

46.2.3 EJECUCIÓN

tableng devuelve la longitud de la tabla especificada, que será una potencia de 2 en la mayoría de los casos, ya que no indica si la tabla tiene un elemento límite añadido (parece que esta información no está disponible en la estructura de los datos de la tabla). Si no se encuentra la tabla, se devolverá un 0.

Probablemente será útil para configurar el código del instrumento a la hora de realizar operaciones de manipulación de tablas, como **tablemix** y **tablecopy**.

46.2.4 AUTOR

Robin Whittle
Australia
Mayo 1997

47 CONTROL DE TABLAS DE FUNCIÓN: SELECCIÓN DE TABLAS

47.1 tablekt, tableikt

```
kr tablekt kndx, kfn[, ixmode[, ixoff[, iwrap]]]
ar tablekt xndx, kfn[, ixmode[, ixoff[, iwrap]]]
kr tableikt kndx, kfn[, ixmode[, ixoff[, iwrap]]]
ar tableikt xndx, kfn[, ixmode[, ixoff[, iwrap]]]
```

47.1.1 DESCRIPCIÓN

Controla (a frecuencia k-) los números de identificación de las tablas.

Los opcodes estándar de Csound **table** y **tablei**, cuando producen un resultado de tipo a- o k-, sólo pueden usar una variable de inicialización para seleccionar el número de la tabla. **tablekt** y **tableikt** aceptan tanto variables de control (k-) como de inicialización (i-). En todos los demás sentidos son opcodes idénticos.

47.1.2 INICIALIZACIÓN

indx - índice de la tabla, que debe ser un número positivo.

ifn - número de la tabla. Debe ser mayor o igual que 1. Los valores decimales se redondean a enteros. Si el número no apunta a una tabla válida, o la tabla no ha sido cargada aún (**GEN01**), se produce un error y el instrumento será desactivado.

ixmode – si es 0, los rangos de *xndx* e *ixoff* concuerdan con la longitud de la tabla; si no es 0, *xndx* e *ixoff* toman valores en el rango de 0 a 1. El valor por defecto es 0.

ixoff - si es 0, el índice final se controla directamente mediante *xndx*, es decir, la indexación empieza desde el comienzo de la tabla; si no es 0, la indexación comienza desde cualquier otra posición en la tabla. Los valores deben ser positivos y menores que la longitud de la tabla (cuando *ixmode* = 0) o menores que 1 (cuando *ixmode* es distinto de 0). El valor por defecto es 0.

iwrap – si es 0, trabaja en modo límite: cuando el índice total es menor que 0, entonces el índice final es 0. Un índice total mayor que la longitud de la tabla da un índice final igual a la longitud de dicha tabla (los índices totales que excedan por arriba el límite de la tabla se paran en el último elemento de ésta). Si *iwrap* es distinto de 0, trabaja en modo cíclico: el índice total es pasado a través de la tabla en ciclos de duración igual a la longitud de la tabla para que todos los índices totales caigan dentro de dicha tabla. Por ejemplo, en una tabla de longitud 8, si *xndx* toma el valor 5 e *ixoff* el valor 6, tendremos un índice total igual a 11, que, al pasarse cíclicamente a lo largo de la longitud de la tabla, dará lugar a un índice final de 3. El valor por defecto es 0.

47.1.3 EJECUCIÓN

kndx - índice de la tabla, que debe ser un número positivo.

andx - o bien concuerda con la longitud de la tabla (cuando *ixmode* es 0), o bien está en el rango de 0 a 1 (cuando *ixmode* es distinto a 0).

kfn - número de la tabla. Debe ser mayor o igual que 1. Los valores decimales son redondeados a enteros. Si el número no apunta a una tabla válida, o la tabla aún no ha sido cargada (**GEN01**), se produce un error y se desactiva el instrumento.

47.1.4 AUTOR

Robin Whittle
Australia
1997

48 CONTROL DE TABLAS DE FUNCIÓN: OPERACIONES DE LECTURA Y ESCRITURA

48.1 tableiw, tablew, tablewkt

tableiw	<i>isig</i> , <i>indx</i> , <i>ifn</i> [, <i>ixmode</i> [, <i>ixoff</i> [, <i>iwgmode</i>]]]
tablew	<i>ksig</i> , <i>kndx</i> , <i>ifn</i> [, <i>ixmode</i> [, <i>ixoff</i> [, <i>iwgmode</i>]]]
tablew	<i>asig</i> , <i>andx</i> , <i>ifn</i> [, <i>ixmode</i> [, <i>ixoff</i> [, <i>iwgmode</i>]]]
tablewkt	<i>ksig</i> , <i>kndx</i> , <i>kfn</i> [, <i>ixmode</i> [, <i>ixoff</i> [, <i>iwgmode</i>]]]
tablewkt	<i>asig</i> , <i>andx</i> , <i>kfn</i> [, <i>ixmode</i> [, <i>ixoff</i> [, <i>iwgmode</i>]]]

48.1.1 DESCRIPCIÓN

Estos opcodes operan sobre tablas de función existentes, cambiando sus contenidos. **tableiw** se usa cuando todas las entradas son variables o constantes de inicialización y sólo se desea ejecutarlo en la inicialización del instrumento. **tablew** se usa para escribir valores de control (k-) o de audio (a-), siendo especificado el número de la tabla en la inicialización. **tablewkt** hace lo mismo, pero usando una variable de control (k-) para designar el número de la tabla. Las combinaciones de tipos de variable válidas quedan especificadas por la primera letra de sus nombres.

48.1.2 INICIALIZACIÓN

isig, *ksig*, *asig*- valor a ser escrito en la tabla.

indx, *kndx*, *andx* - índice de la tabla, que puede ser tanto un número positivo dentro del rango de la longitud de la tabla (cuando *ixmode* = 0), como un número del 0 al 1 (cuando *ixmode* es distinto de 0).

ifn, *kfn* - número de la tabla. Deber ser un entero mayor o igual a 1. Los números decimales se redondean a enteros. Si el número de una tabla no apunta a una tabla válida o la tabla todavía no ha sido cargada (GEN01), se produce un error y el instrumento será desactivado.

ixmode - El valor por defecto es 0.

- cuando es 0, los rangos de *xndx* e *ixoff* concuerdan con los de la tabla.
- cuando es distinto a 0, *xndx* e *ixoff* deben estar entre 0 y 1.

ixoff - El valor por defecto es 0.

- cuando es 0, el índice final es controlado directamente por *xndx*, es decir, la indexación empieza desde el principio de la tabla.
- cuando es distinto a 0, la indexación empieza en cualquier otro elemento de la tabla. Los valores deben ser positivos y menores que la longitud de la tabla cuando *ixmode* es 0, o menor que 1 cuando *ixmode* es distinto a 0.

iwgmode - El valor por defecto es 0.

- 0 significa modo límite.
- 1 modo cíclico.
- 2 modo protegido.

48.1.3 EJECUCIÓN

48.1.3.1 Modo Límite (0)

Limita el índice final ($ndx + ixoff$) a un valor entre 0 y la posición del elemento límite añadido. Para tablas de longitud 5, por ejemplo, esto significa que se puede escribir en las posiciones de 0 a 3 y en el elemento límite añadido (posición 4). Un índice final negativo escribe en la posición 0. Un índice final igual a 4 escribe en la posición 4.

48.1.3.2 Modo cíclico (1)

El índice final se comporta cíclicamente entre las posiciones de 0 a E, donde E es menor en una unidad a la longitud de la tabla o a la potencia de 2 que es "la longitud de la tabla -1". Por ejemplo, en una tabla de 0 a 3, leída cíclicamente, un índice 6 apunta a la posición 2.

48.1.3.3 Modo Protegido (2)

El elemento límite añadido se escribe al mismo tiempo que el valor de la posición 0 y con el mismo valor. Esto facilita la escritura de tablas que están pensadas para ser leídas con interpolación, a fin de producir formas de onda cíclicas suaves. Además, antes de ser usado, el índice total es incrementado por la mitad del intervalo formado entre una posición de la tabla y la siguiente, y luego es redondeado a un entero que representa la dirección de la posición de la tabla.

Normalmente (cuando *igwmode* es 0 ó 1) para una tabla de longitud 5, cuyas posiciones de 0 a 3 forman el cuerpo de la tabla y la posición 4 es el elemento límite añadido, un índice total dentro del rango de 0 a 0.999 escribirá en la posición 0 (0.999 significa una cantidad justo por debajo de 1). Para valores entre 1.0 y 1.999 escribirá en la posición 1, etc. Un patrón similar opera para todos los índices de 0 a 4.999 (*igwmode* = 0) o a 3.999 (*igwmode* = 1). Con *igwmode* = 0 se nos permite escribir en las posiciones entre 0 y 4, siendo escrito el elemento límite (posición 4) con un valor diferente al de la posición 0.

En cambio, con una tabla de longitud 5 y con *iwgmode* = 2, cuando el índice total está en el rango de 0 a 0.499, se escribirá en las posiciones desde la 0 a la 4. Un índice en el rango de 0.5 a 1.499 escribirá en la posición 1 y así sucesivamente.

De esta manera, la operación de escritura se aproxima mucho a los resultados de una lectura con interpolación. El modo protegido sólo debe ser usado con tablas que tengan un elemento límite añadido. Este modo se consigue añadiendo 0.5 al índice final, redondeando al entero menor más próximo, pasando a través de la tabla en ciclos iguales a la longitud de la tabla menos 1 (que será una potencia de 2), escribiendo en dicha tabla (en las posiciones de 0 a 3 en nuestro ejemplo) y por último escribiendo en la posición del elemento límite si el índice es igual a 0.

tablew no tiene valor de salida. Los últimos tres parámetros son opcionales y tienen un valor por defecto de 0.

48.1.3.4 Precaución con números k- de tabla.

La siguiente nota puede aplicarse también a los generadores **tablekt** y **tableikt** que pueden admitir ahora un número de tabla que varíe en tiempo de ejecución.

En frecuencias de control o de audio, si tenemos un número de tabla que es menor que 1, que apunta a una tabla que no existe, o a una que tiene una longitud 0 (porque va a tomar sus valores luego desde un fichero) se produce un error y el instrumento se desactiva.

48.1.4 CAMBIO DE NOMBRE

En la versión 3.52 de Csound, el opcode **itablew** se cambió por **tableiw**.

48.1.4 AUTOR

Robin Whittle
Australia
Mayo 1997

48.2 tablegpw, tablemix, tablecopy, tableigpw, tableimix, tableicopy

tablegpw	kfn
tablemix	kdft, kdoff, klen, ks1ft, ks1off, ks1g, ks2ft, ks2off, ks2g
tablecopy	kdft, ksft
tableigpw	ifn
tableimix	idft, idoff, ilen, is1ft, is1off, is1g, is2ft, is2off, is2g
tableicopy	idft, isft

48.2.1 DESCRIPCIÓN

Estos opcodes permiten copiar, mezclar y comprobar tablas.

48.2.2 INICIALIZACIÓN

ifn, kfn - número de la tabla a utilizar.

ksft - número de la tabla fuente.

kdft - número de la tabla destino.

kdoff - offset (desplazamiento) a partir del cual empezar a escribir. Puede ser negativo.

klen - número de operaciones de escritura a realizar. Un valor negativo trabaja hacia atrás.

ks1ft, ks2ft - tablas fuente. Pueden ser las mismas que las tablas destino si se tiene cuidado con la dirección del flujo de los datos copiados.

ks1off, ks2off - Desplazamiento a partir del cual empezar a leer de las tablas fuente.

ks1g, ks2g - Valores de ganancia que se aplican al leer las tablas fuente. Los resultados se suman y esta suma se escribe en la tabla destino.

48.2.3 EJECUCIÓN

tablegpw

Se usa para escribir el elemento límite añadido de la tabla con el valor que hay en la posición 0. No hace nada si la tabla no existe,

Probablemente será útil después de haber manipulado una tabla con **tablemix** o **tablecopy**.

tablemix

Este generador mezcla valores de 2 tablas fuente, con valores de ganancia independientes para cada una, y guarda el resultado en la tabla destino. La escritura se realiza en las posiciones *klen*, normalmente avanzando a través de la tabla si *klen* es positivo. Si es negativo, entonces el orden de lectura y de escritura es de atrás hacia delante, es decir desde los índices más altos a los más bajos de la tabla. Esta opción bidireccional permite desplazar el contenido de una tabla a ambos lados, leyéndola y volviéndola a escribir con un offset distinto.

Si *klen* es 0, no se escribe nada en la tabla. Observa que el valor entero interno de *klen* se deriva de la función floor() del ANSI C, que devuelve el entero más negativo más cercano. Por tanto un valor fraccionario negativo de *klen* de, por ejemplo, -2.3 crearía una longitud interna de -3, y causaría que la copia empezara desde la posición del offset y continuara 2 posiciones a la izquierda.

El índice final de la tabla para su lectura y escritura se calcula según el offset inicial de cada tabla más el valor del índice, que empieza en 0 y luego se incrementa o decrementa en 1 conforme el proceso de mezcla va teniendo lugar.

Estos índices totales pueden llegar a ser valores muy altos, ya que no hay restricción alguna en el valor del offset o en el de *klen*. Sin embargo, a cada índice total de cada tabla se le aplica una operación lógica "Y" (AND) con una máscara que contiene un valor de longitud (por ejemplo 00000111, para una tabla de longitud 8), con el fin de calcular el índice final empleado para leer o escribir en la tabla. Así que ni la lectura ni la escritura pueden llevarse a cabo fuera de los límites de las tablas. Esto funciona igual que el modo cíclico de lectura y escritura de tablas. Estos opcodes no leen ni escriben el elemento límite añadido. Si una tabla ha sido reescrita con uno de ellos y se supone que debe haber un elemento límite añadido con el mismo valor que el de la posición 0, hay que invocar a **tablegpw** después.

Los índices y offsets se indican todos como posiciones de tabla, es decir no están normalizados entre 0 y 1. Así que para una tabla con una longitud de 256 elementos, *klen* debe tener un valor de 256 si queremos que la tabla sea leída o escrita entera. Las tablas no tienen porqué ser del mismo tamaño, ya que los bucles se producen individual e independientemente para cada una de ellas.

tablecopy

Es un opcode para copiar tablas de una forma rápida y sencilla. Coge la longitud de la tabla destino y empieza a leer desde el principio de la tabla fuente. En este proceso se puede leer varias veces la tabla fuente si es necesario. Una tabla fuente con una longitud 1 causará que todos los valores de la tabla destino tomen el valor del único elemento de la tabla fuente.

tablecopy no puede leer ni escribir el elemento límite de la tabla. Para ello hay que usar **table** con un *ndx* igual a la longitud de la tabla.

Para escribir en el elemento límite añadido el valor de la posición 0 se puede usar **tablegpw**. Esto se usa principalmente para modificar rápidamente tablas de función en aplicaciones en tiempo real.

48.2.4 CAMBIOS DE NOMBRE

A partir de la versión 3.52, los nombres de los opcodes **itablegpw**, **itablemix**, **itablecopy** y **itableng**, han sido cambiados por **tableigpw**, **tableimix**, **tableicopy** y **tableng**, respectivamente.

48.2.4 AUTOR

Robin Whittle
Australia
Mayo 1997

48.3 tablera, tablewa

<i>ar</i>	tablera	<i>kfn, kstart, koff</i>
<i>kstart</i>	tablewa	<i>kfn, asig, koff</i>

48.3.1 DESCRIPCIÓN

Estos opcodes escriben o leen posiciones sucesivas de una tabla a (o desde) una variable de tipo a-. Se necesita aclarar algunos conceptos antes de usarlas. Tienen por lo menos 2 aplicaciones principales y totalmente diferentes, que se discutirán más adelante.

48.3.2 INICIALIZACIÓN

ar - variable destino, de tipo a-, en la que se guarda el valor de las muestras de una tabla.

kfn - número de la tabla a leer o a escribir (tipo i- o k-).

kstart - dónde escribir o leer en una tabla.

asig - señal de audio a leer cuando se escribe en la tabla.

koff - offset (desplazamiento) de la tabla (valores de tipo i- o k-). El rango es ilimitado (ver explicación al final de la sección).

48.3.3 EJECUCIÓN

Se entiende que, en cada llamada, estos opcodes van en parejas o con varias unidades **tablera** precediendo a varias **tablewa** (ambas con la misma variable *kstart*).

Estos opcodes leen y escriben en posiciones secuenciales de una tabla a frecuencia de audio, escribiendo y leyendo cada ciclo los valores *ksmps*.

tablera empieza a leer desde la posición *kstart*. **tablewa** empieza a escribir también desde la posición *kstart* y luego incrementa en uno la posición a la que apunta *kstart* (Observa que para **tablewa**, *kstart* es tanto una variable de entrada como de salida). Si el índice de escritura alcanza el final de la tabla, el proceso de escritura finaliza y *kstart* es puesta a 0.

Por ejemplo, si la longitud de la tabla fuera 16 (posiciones de 0 a 15) y *ksmps* fuera 5, tendrían lugar los siguientes pasos en sucesivas ejecuciones del generador **tablewa**, asumiendo que *kstart* empezara en 0.

Ejecución número	<i>kstart</i> inicial	<i>kstart</i> final	posiciones escritas
1	0	5	0 1 2 3 4
2	5	10	5 6 7 8 9
3	10	15	10 11 12 13 14
4	15	0	15

Esto facilita el procesamiento de los datos de la tabla usando código de orquesta estándar (que use señales de tipo a-) entre los opcodes **tablera** y **tablewa**

Estos opcodes son una especie de parche, pero permiten a todos los operadores de CSound que trabajan con variables k- ser usados (con precaución) con variables a-. Algo que sólo sería posible de otra manera configurando **ksmps** = 1.

Algunas precauciones:

- El código que usa variables k- en el bucle de procesado está realmente ejecutándose a frecuencia de audio, así que funciones dependientes del tiempo como **port** y **oscil** trabajan más rápido de lo normal (ya que su código espera ejecutarse a frecuencia de control, no de audio).
- Esta técnica producirá resultados indeseables a menos que **ksmps** tenga un valor que no sobrepase la longitud de la tabla. Por ejemplo, una tabla de longitud 16 alojará muestras de la 1 a la 16, así que en este caso **ksmps** deberá tomar valores de 1 a 16.

Ambos opcodes generan un error y desactivan el instrumento si se selecciona una tabla con una longitud menor que **ksmps**. Así mismo, se produce un error si *kstart* es menor que 0 o mayor que la entrada más alta de la tabla (si *kstart* = longitud de la tabla).

- Se espera que *kstart* contenga valores enteros en el rango 0 a "longitud de la tabla -1". Valores fraccionales mayores no afectan a la ejecución pero tampoco logran nada útil.
- Estos opcodes usan interpolación y *kstart* y *koff* siempre están en el rango de 0 a "longitud de la tabla -1", a diferencia de otros opcodes que también trabajan con tablas (que usan un rango de 0 a 1). *koff* puede estar fuera de ese rango pero es truncado por la operación AND final.
- Estos opcodes funcionan permanentemente en modo cíclico. Cuando *koff* es 0, no es necesario que se produzca bucle alguno, ya que el índice *kstart*++ estará siempre dentro del rango normal de la tabla. Cuando *koff* sea distinto de 0, sí puede producirse que el bucle vuelva al principio (*wrapping*).
- El offset no afecta al número de ciclos de lectura/escritura ejecutados ni a los valores pasados a *kstart* por **tablewa**.
- Estos opcodes no pueden leer ni escribir el elemento límite añadido de la tabla. Usa **tablegpw** para hacerlo, después de haber modificado la tabla con **tablewa**.

48.3.4 EJEMPLOS

```
kstart = 0
lab1: atemp tablera ktabsource, kstart, 0 ; Lee 5 valores de la tabla y los pasa a
                                           ; una variable de audio.
atemp = log(atemp)                       ; Procesa los valores usando opcodes
                                           ; que usan variables a-.
kstart tablewa ktabdest, atemp, 0       ; lo vuelve a escribir en la tabla.
if ktemp 0 goto lab1                     ; bucle hasta que todas las posiciones
                                           ; de la tabla han sido procesadas.
```

El ejemplo de arriba muestra un bucle de procesado que se ejecuta cada ciclo de control, leyendo cada posición de la tabla *ktabsource*, y escribiendo el logaritmo de esos valores en las mismas posiciones de la tabla *ktabdest*.

Esto permite manipular de una vez tablas enteras (con código que use variables a-), partes de ellas (con offsets y bucles de control diferentes) o varias tablas distintas, volviendo a reescribir los valores a otra (o la misma) tabla. Esto es una especie de parche, pero es más rápido que hacerlo con los opcodes de procesado de tablas que trabajan con variables k-.

Otro posible uso es:

```

kzero = 0
kloop = 0
kzero tablewa 23, asignal, 0 ; muestras ksmps de tipo a- escritas
                                ; en las posiciones de 0 a (ksmps -1) de la tabla 23.
lab1: ktemp table kloop, 23 ; empieza un bucle que se repite ksmps veces,
                                ; en el que en cada ciclo se procesa
[ Código manipulador ] ; los 23 valores de la tabla con código
[ del valor de ktemp. ] ; que use variables k-
tablew ktemp, kloop, 23 ; Escribe el valor procesado en la tabla.
kloop = kloop + 1 ; Incrementa en 1 kloop, que es tanto
                                ; el índice de la tabla como el contador
if kloop < ksmps goto lab1 ; del bucle. Sigue en el bucle hasta que todos
                                ; los valores de la tabla han sido procesados.
asignal tablera 23, 0, 0 ; Copia los contenidos de la tabla
                                ; a una variable tipo a-.

```

koff - es un offset (desplazamiento) añadido a la suma de *kstart* con la variable que contiene el índice interno que se usa para recorrer la tabla. Al resultado se le aplica entonces una operación lógica AND con una máscara con la longitud de la tabla (por ejemplo 0000 0111 para una tabla de longitud 8, ó 9 si contamos el elemento límite añadido) y entonces se usa el índice final para leer o escribir en la tabla. *koff* puede tomar cualquier valor. Es convertido a un tipo long usando la función del ANSI C floor(), de tal manera que -4.3 se convierte en -5. Esto es precisamente lo que desearemos cuando estemos usando offsets con rango sobre y bajo 0.

Idealmente debería ser una variable opcional, con 0 como valor por defecto. Sin embargo, con el código de lectura de Csound, tal parámetro debe ejecutarse sólo en la inicialización y, como en este caso queremos que sea un valor -k (en lugar de i-), no podemos tener un valor por defecto.

49 MODIFICADORES DE SEÑAL: FILTROS ESTÁNDAR

49.1 port, portk, tone, tonek, atone, atonek, reson, resonk, areson, aresonk

kr	port	ksig, ihtim[, isig]
kr	portk	ksig, khtim[, isig]
kr	tonek	ksig, khp[, iskip]
kr	atonek	ksig, khp[, iskip]
kr	resonk	ksig, kcf, kbw[, iscl, iskip]
kr	aresonk	ksig, kcf, kbw[, iscl, iskip]
ar	tone	asig, khp[, iskip]
ar	atone	asig, khp[, iskip]
ar	reson	asig, kcf, kbw[, iscl, iskip]
ar	areson	asig, kcf, kbw[, iscl, iskip]

49.1.1 DESCRIPCIÓN

Una señal de control o de audio es modificada por un filtro recursivo pasa bajos o pasa banda con respuesta de frecuencia variable.

49.1.2 INICIALIZACIÓN

isig - valor inicial (es decir, previo) de la retroalimentación interna. El valor por defecto es 0.

iskip - disposición inicial del espacio de los datos internos. Como el filtrado incorpora un bucle de retroalimentación con las salidas anteriores, el estado inicial del espacio de almacenaje usado no es significativo. Un valor 0 limpiará el espacio; un valor distinto de 0 permitirá que la información previa se mantenga. El valor por defecto es 0.

iscl - factor de normalización codificado para los resonadores. Un valor 1 implica un factor de respuesta pico de 1, es decir, que todas las frecuencias que no sean *kcf* se atenúan de acuerdo con la curva de respuesta (normalizada). Un valor 2 eleva el factor de respuesta hasta que su valor RMS total sea 1 (dicha equalización adrede de la potencia de entrada y salida, asume que todas las frecuencias se encuentran físicamente presentes; por tanto se aplica sobre todo a ruido blanco). Un valor de 0 implica la no normalización de la señal, dejándola tal cual para un posterior ajuste (ver **balance**). El valor por defecto es 0.

49.1.3 EJECUCIÓN

port aplica un efecto de portamento a una señal de control escalonada. A cada nuevo valor, la señal *ksig* es filtrada para moverla hacia dicho valor, a una frecuencia determinada por *ihtim*. *ihtim* es el "tiempo medio" de la función (expresado en segundos), durante el cual la curva recorrerá la mitad de la distancia hacia el nuevo valor, así sucesivamente, teóricamente no alcanzando jamás su asíntota. Con **portk**, el tiempo medio puede ser variado por una señal de control (k-).

tone implementa un filtro pasa bajos recursivo de primer orden en el que la variable *khp* (en Hz) determina la potencia media en la curva de respuesta. La potencia media se define como "*potencia-pico / raíz de 2*".

reson es un filtro de segundo orden en el que *kcf* controla su frecuencia central, o posición (en Hz) del pico de su respuesta, y *kbw* controla su ancho de banda (la diferencia en Hz entre los puntos de potencia media superior e inferior).

atone y **areson** son filtros cuyas funciones de transferencia son complementarias a las de **tone** y **reson**. Así, **atone** es un filtro pasa altos y **areson** un filtro para banda cuyas funciones de transferencia (o curvas de respuesta) representan los aspectos "no filtrados" de sus complementarios. Observa, sin embargo, que la escala de potencia no está normalizada en **atone** y **areson**, sino que sigue siendo el complemento real de sus unidades correspondientes. Así, una señal de audio, filtrada por unidades **reson** y **areson** añadidas en paralelo, seguirá como si no hubiera sido modificada. Esta propiedad es particularmente útil para controlar la mezcla de distintas fuentes (ver **lpreson**). Se pueden obtener curvas de respuesta complejas, como las de picos múltiples (configuraciones en peine), usando un banco de filtros en serie (la respuesta resultante será el producto de las respuestas de sus componentes). En tales casos, la atenuación combinada puede ocasionar una seria pérdida de potencia en la señal, lo que puede ser arreglado con el uso del opcode **balance**.

49.2 tonex, atonex, resonx

ar	tonex	asig, khp[, inumlayer, iskip]
ar	atonex	asig, khp[, inumlayer, iskip]
ar	resonx	asig, kcf, kbw[, inumlayer, iscl, iskip]

49.2.1 DESCRIPCIÓN

tonex, **atonex** y **resonx** son filtros consistentes en más capas de filtros **tone**, **atone** y **reson**, con los mismos argumentos, conectados en serie. Usando una pila de un gran número de filtros se puede conseguir una curva de corte pronunciada. Es más rápido utilizar estos nuevos filtros que usar un número mayor de opcodes antiguos en una orquesta, porque con los primeros sólo se necesita una inicialización y un ciclo de control (k-) a la vez, y el bucle de audio cae enteramente dentro de la memoria cache del procesador.

49.2.2 INICIALIZACIÓN

inumlayer - número de elementos en la pila del filtro. El valor por defecto es 4.

iskip - disposición inicial del espacio de datos interno. Debido a que el filtrado incorpora un bucle de retroalimentación antes de producir la salida, el estado inicial del espacio de almacenamiento usado es significativo. Un valor de 0 limpiará el espacio. Un valor distinto a 0 permitirá que la información previa no se pierda. El valor por defecto es 0.

iscl - factor de escala para los resonadores. Un valor de 1 significa un factor de respuesta de pico de 1, es decir que se atenuarán todas las frecuencias distintas de *kcf* de acuerdo a la curva (normalizada) de respuesta. Un valor de 2, eleva el factor de respuesta para que su valor RMS total sea igual a 1. (Dicha ecualización de la potencia de entrada y de salida asume que todas las frecuencias están físicamente presentes; por lo tanto se aplica mayormente al ruido blanco). Un valor de 0 significa que no se escalará la señal, dejándola para ser modificada posteriormente (ver **balance**). El valor por defecto es 0.

49.2.3 EJECUCIÓN

asig - señal de entrada.

khp - punto medio de potencia de la curva de respuesta. El punto de potencia media se define como la potencia de pico dividida por la raíz de 2.

kcf - frecuencia central del filtro, o posición en Hz de la respuesta de pico.

kbw - ancho de banda del filtro (es decir, la diferencia en Hz entre los puntos de potencia media superior e inferior).

49.3 resonr, resonz

ar	resonr	asig, kcf, kbw[, iscl, iskip]
ar	resonz	asig, kcf, kbw[, iscl, iskip]

49.3.1 DESCRIPCIÓN

Implementaciones de un filtro pasa banda de segundo orden con dos polos y dos ceros, con una respuesta variable a la frecuencia.

49.3.2 INICIALIZACIÓN

Las variables de inicialización opcionales para **resonr** y **resonz** son idénticas a las variables de inicialización de **reson**.

iskip - disposición inicial del espacio de los datos internos. Como el filtrado incorpora un bucle de retroalimentación con las salidas anteriores, el estado inicial del espacio de almacenaje usado no es significativo. Un valor 0 limpiará el espacio; un valor distinto de 0 permitirá que la información previa se mantenga. El valor por defecto es 0.

iscl - factor de normalización codificado para los resonadores. Un valor 1 implica un factor de respuesta pico de 1, es decir, que todas las frecuencias que no sean *kcf* se atenúan de acuerdo con la curva de respuesta (normalizada). Un valor 2 eleva el factor de respuesta hasta que su valor RMS total sea 1 (dicha equalización adrede de la potencia de entrada y salida, asume que todas las frecuencias se encuentran físicamente presentes; por tanto se aplica sobre todo a ruido blanco). Un valor de 0 implica la no normalización de la señal, dejándola tal cual para un posterior ajuste (ver **balance**). El valor por defecto es 0.

49.3.3 EJECUCIÓN

resonr y **resonz** son variaciones del clásico resonador pasa banda de dos polos (**reson**). Ambos filtros tienen dos ceros en sus funciones de transferencia, además de los dos polos. **resonz** tiene sus ceros en $z=1$ y $z=-1$. **resonr** tiene sus ceros en $+\sqrt{R}$ y $-\sqrt{R}$, donde R es el radio de los polos en el plano complejo z . La adición de ceros a **resonr** y **resonz** proporciona una mejora en la selectividad de la respuesta a la amplitud de estos filtros en las frecuencias de corte cercanas a 0, al precio de un menor detalle en la selectividad de las frecuencias por encima del pico de corte.

resonr y **resonz** están muy cerca de conseguir una ganancia constante al ser barrida la frecuencia central, lo que proporciona un control de la curva de respuesta mucho más eficiente que el que se obtiene con los resonadores bipolares tradicionales como **reson**.

resonr y **resonz** producen un sonido considerablemente distinto al de **reson**, especialmente para frecuencias centrales más bajas. Experimentar es la mejor manera de determinar qué resonador se adapta mejor a una determinada aplicación.

asig - señal de entrada a ser filtrada.

kcf - frecuencia resonante o de corte del filtro, expresada en Hz.

kbw - ancho de banda del filtro (es decir, la diferencia en Hz entre los puntos de potencia media superior e inferior)

49.3.4 EJEMPLO

```
;Fichero orquesta para el barrido de un filtro resonante sobre una forma de onda
;diente de sierra. Las salidas de reson, resonr y resonz se escalan por los
;coeficientes que se indican en la partitura, para que cada filtro pueda ser oído en
;el mismo instrumento.
```

```
sr          =      44100
kr          =      4410
ksmpr       =      10
nchnls      =      1
```

```
instr 1
idur       =      p3
ibegfreq   =      p4 ; frecuencia inicial del barrido
iendfreq   =      p5 ; frecuencia final del barrido
ibw        =      p6 ; ancho de banda de los filtros en Hz
ifreq      =      p7 ; frecuencia de gbuzz que se va a filtrar
iamp       =      p8 ; amplitud por la que escalar la salida
ires       =      p9 ; coeficiente de reson en la salida
iresr      =      p10 ; coeficiente de resonr en la salida
iresz      =      p11 ; coeficiente de resonz en la salida
```

```
; envolvente de la frecuencia de corte del filtro
```

```
kfreq      linseg ibegfreq, idur * .5, iendfreq, idur * .5, ibegfreq
```

```
; envolvente de amplitud para evitar muestras fuera de rango.
```

```
kenv       linseg 0, .1, iamp, idur - .2, iamp, .1, 0
```

```
; número de armónicos para gbuzz escalados para evitar el aliasing.
```

```
iharms     =      (sr*.4)/ifreq
asig       gbuzz 1, ifreq, iharms, 1, .9, 1 ; onda diente de sierra
ain        =      kenv * asig ; salida escalada por amp
```

```
; envolvente
```

```
ares       reson  ain, kfreq, ibw, 1
aresr      resonr ain, kfreq, ibw, 1
aresz      resonz ain, kfreq, ibw, 1
out        ares * ires + aresr * iresr + aresz * iresz
endin
```

; Partitura

```
f1 0 8192 9 1 1 .25 ; tablas de cosenos para gbuzz
i1 0 10 1 3000 200 100 4000 1 0 0 ; salida de reson con un ancho de banda de 200
i1 10 10 1 3000 200 100 4000 0 1 0 ; salida de resonr con un ancho de banda de 200
i1 20 10 1 3000 200 100 4000 0 0 1 ; salida de resonz con un ancho de banda de 200
i1 30 10 1 3000 50 200 8000 1 0 0 ; salida de reson con un ancho de banda de 50
i1 40 10 1 3000 50 200 8000 0 1 0 ; salida de resonr con un ancho de banda de 50
i1 50 10 1 3000 50 200 8000 0 0 1 ; salida de resonz con un ancho de banda de 50
e
```

49.3.5 HISTORIAL TÉCNICO

resonr y **resonz** fueron originalmente descritos en un artículo de Julius O. Smith y James B. Angell (1). Ellos recomendaban emplear **resonz** (ceros en +1 y -1) cuando era prioritaria la velocidad de computación, ya que emplea una multiplicación menos por cada muestra, mientras que **resonr** (ceros en + y - la raíz cuadrada del radio polar R) era recomendado para situaciones en las que se requería un pico central con una ganancia constante.

Ken Steiglitz, en un artículo posterior (2), demostró que **resonz** tenía una ganancia constante en el pico real del filtro, al contrario que **resonr**, que mostraba una ganancia constante en el ángulo del polo. Steiglitz también recomendó **resonz** por tener cortes más agudos en la curva de ganancia en 0 y en la frecuencia Nyquist. En un reciente libro de Steiglitz (3) aparece una discusión técnica profunda sobre **reson** y **resonz**, mientras que el libro de Dodge y Jerse (4) ilustra las diferencias en las curvas de respuesta de **reson** y **resonz**.

49.3.6 REFERENCIAS

1. Smith, Julius O. and Angell, James B., "A Constant-Gain Resonator Tuned by a Single Coefficient," *Computer Music Journal*, vol. 6, no. 4, pp. 36-39, Winter 1982.
2. Steiglitz, Ken, "A Note on Constant-Gain Digital Resonators," *Computer Music Journal*, vol. 18, no. 4, pp. 8-10, Winter 1994.
3. Ken Steiglitz, *A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music*. Addison-Wesley Publishing Company, Menlo Park, CA, 1996.
4. Dodge, Charles and Jerse, Thomas A., *Computer Music: Synthesis, Composition, and Performance*. New York: Schirmer Books, 1997, 2nd edition, pp. 211-214.

49.3.7 AUTOR

Sean Costello
Seattle, Washington
1999
Nuevo en la versión 3.55

49.4 resony

ar **resony** asig, kbf, kbw, inum, ksep[, iscl, iskip]

49.4.1 DESCRIPCIÓN

Un banco de filtros pasa banda de segundo orden, conectados en paralelo.

49.4.2 INICIALIZACIÓN

inum - número de filtros.

iscl - factor de escala para los resonadores. Un valor de 1 proporciona un factor de respuesta pico de 1, es decir, todas las frecuencias que no sean *kcf* serán atenuadas de acuerdo a la curva de respuesta (normalizada). Un valor de 2 eleva el factor de respuesta para que su valor de potencia media global sea 1. (Esta ecualización adrede de la potencia de la entrada y la salida asume que todas las frecuencias están físicamente presentes; por tanto es especialmente útil con ruido blanco). Un valor de 0 significa que la señal no será escalada, dejando esa tarea a un ajuste posterior (por ejemplo con **balance**). El valor por defecto es 0.

iskip - disposición inicial del espacio de datos interno. Debido a que el filtrado incorpora un bucle de retroalimentación de la salida previa, el estado inicial del espacio de almacenamiento es significativo. Un valor de 0 limpiará el espacio. Un valor distinto de 0 permitirá que la información previa permanezca en el espacio de almacenamiento. El valor por defecto es 0.

49.4.3 EJECUCIÓN

asig - señal de audio de entrada.

kbf - frecuencia base, es decir, frecuencia central en Hz. del filtro más bajo.

kbw - ancho de banda en Hz.

ksep - separación, en octavas, de la frecuencia central de los filtros.

resony es un banco de filtros pasa banda de segundo orden conectados en paralelo (es decir, la señal resultante es una mezcla de la salida de cada filtro), con una separación de frecuencias, una frecuencia base y un ancho de banda variables a frecuencia de control. La frecuencia central de cada filtro depende de las variables *kbf* y *ksep*. El máximo número de filtros es 100.

49.4.4 EJEMPLOS

En este ejemplo la variable global *gk1* modifica *kbf*, *gk2* modifica *kbw*, *gk3* *inum*, *gk4* *ksep*, y *gk5* el volumen total.

```
instr 1
a1  soundin    "myfile.aif"
a2  resony     a1, gk1, gk2, i(gk3), gk4, 2
      out       a2 * gk5
```

49.4.5 AUTOR

Gabriel Maldonado

Italia

1999

Nuevo versión 3.56

49.5 lowres, lowresx

```
ar    lowres    asig, kcutoff, kresonance [,iskip]
ar    lowresx   asig, kcutoff, kresonance [, inumlayer, iskip]
```

49.5.1 DESCRIPCIÓN

lowres es un filtro resonador pasa-bajos. **lowresx** es lo mismo que un diseño de varias capas de **lowres**, con los mismos argumentos, conectados en serie.

49.5.2 INICIALIZACIÓN

inumlayer - número de elementos en la pila de **lowresx**. El valor por defecto es 4. No hay máximo.

iskip - disposición inicial del espacio de datos interno. Un valor 0 limpiará el espacio; un valor distinto de cero permitirá que se retenga la información previa. El valor por defecto es 0.

49.5.3 EJECUCIÓN

asig - señal de entrada.

kcutoff - frecuencia de corte del filtro.

kresonance - cantidad de resonancia.

lowres es un filtro resonador pasa-bajos derivado de una orquesta de Hans Mikelson. Esta implementación es mucho más rápida que la de la orquesta y permite una **kr** menor que **sr**. *kcutoff* no va en Hz y *kresonance* no va en dB, así que experimenta para encontrar los mejores resultados.

lowresx es un diseño con varias capas de **lowres**, con los mismos argumentos, conectados en serie. Usando una pila con un mayor número de filtros se puede conseguir una curva de corte mucho más aguda. Esto es más rápido que usar muchos **lowres** en la orquesta, porque sólo se necesita una inicialización y un ciclo de control (k-) cada vez, y el bucle de audio cae enteramente dentro de la memoria cache del procesador.

49.5.4 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1998

49.6 vlowres

ar **vlowres** *asig, kfco, kres, iord, ksep*

49.6.1 DESCRIPCIÓN

Banco de filtros en los que la frecuencia de corte de cada uno puede ser controlada por separado por el usuario.

49.6.2 INICIALIZACIÓN

iord - número total de filtros (de 1 a 10).

49.6.3 EJECUCIÓN

asig - señal de entrada.

kfco - frecuencia de corte (no va en Hz).

ksep - separación de la frecuencia de corte para cada filtro.

vlowres (filtro resonador pasa-bajos variable) permite una curva variable de respuesta en cada filtro resonante. Puede ser pensado como un banco de filtros resonantes pasa-bajos, cada uno con la misma resonancia, conectados en serie. La frecuencia de corte de cada filtro puede variar con los parámetros *kfco* y *ksep*.

49.7 lowpass2

ar lowpass2 asig, kcf, kq[, iskip]

49.7.1 DESCRIPCIÓN

Implementación de un filtro pasa bajos resonante de segundo orden.

49.7.2 INICIALIZACIÓN

iskip – disposición inicial del espacio de datos interno. Un valor 0 limpiaría el espacio. Un valor distinto de 0 retendría la información previa. El valor por defecto es 0.

49.7.3 EJECUCIÓN

asig - señal de entrada a filtrar.

kcf - frecuencia de corte (o de resonancia) del filtro, expresada en Hz.

kq - Q del filtro, definido, para filtros pasa banda, como el ancho de banda / frecuencia de corte. *Kq* debe estar entre 1 y 500.

Es un filtro pasa bajos IIR de segundo orden, que ajusta, a frecuencia de control, la frecuencia de corte, transformando la respuesta del filtro pasa bajos en una respuesta similar a la de un filtro pasa banda, pero con más energía en la zona baja de frecuencias. Esto corresponde a incrementar la magnitud y la “agudeza” del pico de resonancia. Para valores altos de *kq*, puede que se requiera el uso de una función de escala como **balance**. En la práctica, esto permite la simulación de los filtros controlados por voltaje de un sintetizador analógico, o la creación de una altura con amplitud constante durante el filtrado de ruido blanco.

49.7.4 EJEMPLO

; Fichero orquesta para el barrido de un filtro resonante sobre una onda diente de
; sierra.

```
sr          =          44100
kr          =          2205
ksmps      =          20
nchnls     =          1
```

```
instr 1
```

```
idur       =          p3
ifreq      =          p4
iamp       =          p5 * .5
iharms     =          sr*.4) / ifreq
```

; onda diente de sierra

```
asig      gbuzz      1, ifreq, iharms, 1, .9, 1
```

; envolvente para controlar la frecuencia de corte del filtro

```
kfreq     linseg    1, idur * 0.5, 5000, idur * 0.5, 1
afilt     lowpass2  asig, kfreq, 30
```

; envolvente de amplitud simple

```
kenv      linseg    0, .1, iamp, idur -.2, iamp, .1, 0
          out       asig * kenv
```

```
endin
```

; partitura

```
f1 0 8192 9 1 1 .25
i1 0 5 100 1000
i1 5 5 200 1000
e
```

49.7.5 AUTOR

Sean Costello
Seattle, Washington
Agosto, 1999
Nuevo en la Versión 4.0

49.8 biquad, rezy, moogvcf

ar	biquad	asig, kb0, kb1, kb2, ka0, ka1, ka2[, iskip]
ar	rezy	asig, xfco, xres[, imode]
ar	moogvcf	asig, xfco, xres[, iscale]

49.8.1 DESCRIPCIÓN

Implementación de un filtro de barrido de propósito general y dos filtros resonantes pasa-bajos de barrido.

49.8.2 INICIALIZACIÓN

iskip (opcional) – si no es cero, se omitirá la inicialización. El valor por defecto es 0. (Nuevo en la versión 3.50)

imode (opcional) –si es 0, **rezy** será pasa-bajos, si no, será pasa-altos. El valor por defecto es 0 (Nuevo en la versión 3.50)

iscale (opcional) – factor de escala interno. Usado si la señal *asig* no está en el rango entre +/-1. La entrada se divide primero por *iscale*, luego se multiplica la salida por *iscale*. El valor por defecto es 0 (Nuevo en la versión 3.50).

49.8.3 EJECUCIÓN

asig – señal de entrada.

xfco – frecuencia de corte en Hz del filtro. A partir de la versión 3.50 puede ser de tipo i-, k- o a-.

xres – cantidad de resonancia. Para **rezy**, los valores entre 1 y 100 son los típicos. La resonancia debe ser 1 o mayor. Para **moogvcf**, la oscilación propia ocurre cuando *xres* es aproximadamente 1. A partir de la versión 3.50 puede ser de tipo i-, k- o a-.

biquad es un filtro digital bicuadrático de propósito general de la forma:

$$a_0*y(n) + a_1*y[n-1] + a_2*y[n-2] = b_0*x[n] + b_1*x[n-1] + b_2*x[n-2]$$

Con la siguiente respuesta a la frecuencia:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b_0 + b_1*Z^{-1} + b_2*Z^{-2}}{a_0 + a_1*Z^{-1} + a_2*Z^{-2}}$$

Este tipo de filtro se encuentra a menudo en la literatura del procesamiento de señales digitales. Permite 6 coeficientes (de tipo k-) definidos por el usuario.

rezy es un filtro resonante pasa-bajos creado empíricamente por Hans Mikelson.

moogvcf es una emulación digital de la configuración en escalera de filtros de diodos del Moog. Esta emulación está basada más o menos en el artículo "Analyzing the Moog VCF with Considerations for Digital Implementation" de Stilson y Smith (del CCRMA). Esta versión fue originalmente codificada en Csound por Josep María Comajuncosas. Algunas modificaciones y la conversión a C fueron hechas por Hans Mikelson. Nota: el filtro requiere que la señal de entrada esté normalizada a 1.

49.8.4 EJEMPLOS

;ejemplo de **biquad**

```

kfcon      =          *3.14159265*kfco/sr
kalpha     =          -2*krez*cos(kfcon)*cos(kfcon)+krez*krez*cos(2*kfcon)
kbeta      =          *krez*sin(2*kfcon)-2*krez*cos(kfcon)*sin(kfcon)
kgama      =          +cos(kfcon)
km1        =          *kgama+kbeta*sin(kfcon)
km2        =          *kgama-kbeta*sin(kfcon)
kden       =          (km1*km1+km2*km2)
kb0        =          .5*(kalpha*kalpha+kbeta*kbeta)/kden
kb1        =          kb0
kb2        =          0
ka0        =          1
ka1        =          -2*krez*cos(kfcon)
ka2        =          krez*krez
ayn        biquad    axn, kb0, kb1, kb2, ka0, ka1, ka2
            outs    ayn*iamp/2, ayn*iamp/2

```

```

;      Sta   Dur   Amp   Pitch   Fco   Rez
i14   8.0    1.0   20000  6.00   1000  .8
i14   +     1.0   20000  6.03   2000  .95

```

;ejemplo de **rezzy**

```

kfco       expseg    100+.01*ifco, .2*idur, ifco+100, .5*idur, ifco*.1+100, //
            .3*idur, .001*ifco+100
apulse1    buzz      1,ifqc, sr/2/ifqc, 1 ; para evitar el aliasing
asaw       integ    apulse1
axn        =          asaw-.5
ayn        rezzy    axn, kfco, krez
            outs    ayn*iamp, ayn*iamp

```

```

;      Sta   Dur   Amp   Pitch   Fco   Rez
i10   0.0    1.0   20000  6.00   1000  2
i10   +     1.0   20000  6.03   2000  10

```

;ejemplo de **moogvcf**

```

apulse1    buzz      1,ifqc, sr/2/ifqc, 1 ; para evitar el aliasing
asaw       integ    apulse1
ax         =          asaw-.5
ayn        moogvcf  ax, kfco, krez
            outs    ayn*iamp, ayn*iamp

```

```

;      Sta   Dur   Amp   Pitch   Fco   Rez
i11   4.0    1.0   20000  6.00   1000  .4
i11   +     1.0   20000  6.03   2000  .7

```

49.8.5 AUTOR

Hans Mikelson
Octubre 1998

49.9 svfilter

alow, ahigh, aband **svfilter** asig, kcf, kq[, iscl]

49.9.1 DESCRIPCIÓN

Implementación de un filtro resonante de segundo orden, con salidas pasa bajos, pasa altos y pasa banda simultáneas.

49.9.2 INICIALIZACIÓN

iscl - factor de escala similar al de **reson**. Un valor distinto de 0 significa un factor de respuesta pico de 1, esto es, cualquier frecuencia distinta de *kcf* será atenuada de acuerdo con la curva de respuesta (normalizada). Un valor 0 significa que la señal no se escala, dejando esa tarea para un ajuste posterior (ver por ejemplo **balance**). El valor por defecto es 0.

49.9.3 EJECUCIÓN

svfilter es un filtro de segundo orden, variable en la declaración, que proporciona control (tipo k-) sobre la frecuencia de corte y Q. Al aumentar Q, un pico resonante se forma alrededor de la frecuencia de corte. **svfilter** tiene salidas pasa bajos, pasa altos y pasa banda simultáneas. Mezclando las salidas se pueden generar una gran variedad de curvas de respuesta a la frecuencia. Los filtros variables en la declaración, o filtros "multimodo", eran una característica común en los primeros sintetizadores analógicos, debido a que se podía obtener una gran variedad de sonidos gracias a la interacción de las distintas proporciones en la mezcla de las salidas, las resonancias y las frecuencias de corte. **svfilter** se adapta perfectamente a la emulación de sonidos analógicos, así como otras aplicaciones donde se requieren filtros resonantes.

asig - señal de entrada a ser filtrada.

kcf - frecuencia de corte o resonante del filtro, expresada en Hz.

kq - Q del filtro, definida (para los filtros pasa banda) como "ancho de banda / frecuencia de corte". *kq* debe estar en un rango de 1 a 500. Al incrementar *kq*, la resonancia del filtro aumenta, lo que corresponde a un aumento en la magnitud y en la "agudeza" del pico resonante. Cuando se usa **svfilter** sin escalar la señal (es decir, si *iscl* es 0 o está ausente), el volumen del pico resonante aumenta al aumentar Q. Para valores altos de Q, se recomienda que *iscl* tome un valor distinto a 0, o que se emplee una función de escala externa, como **balance**.

svfilter está basado en un algoritmo que aparece en el libro de Hal Chamberlin *Musical Applications of Microprocessors* (Hayden Books, 1985).

49.9.4 EJEMPLO

;Fichero orquesta para el barrido de un filtro resonante sobre una onda diente de
;sierra. Las salidas separadas del filtro se escalan por los valores pasados desde la
;partitura, y luego se mezclan.

```
sr          =      44100
kr          =      2205
ksmps      =      20
nchnls     =      1

instr 1
idur       =      p3
ifreq      =      p4
iamp       =      p5
ilowamp    =      p6      ; determina la cantidad de salida pasa bajos en la señal
ihighamp   =      p7      ; determina la cantidad de salida pasa altos en la señal
ibandamp   =      p8      ; determina la cantidad de salida pasa banda en la señal
iq         =      p9      ; valor de Q
iharms     =      (sr*.4) / ifreq
asig       gbuzz 1, ifreq, iharms, 1, .9, 1      ; onda diente de sierra
kfreq     linseg 1, idur * 0.5, 4000, idur * 0.5, 1 ; envolvente que controla
                                                ; la frecuencia de corte del filtro

alow, ahigh, aband svfilter asig, kfreq, iq
aout1     =      alow * ilowamp
aout2     =      ahigh * ihighamp
aout3     =      aband * ibandamp
asum      =      aout1 + aout2 + aout3
kenv      linseg 0, .1, iamp, idur -.2, iamp, .1, 0 ; envolvente simple de amplitud
out       out  asum * kenv

endin

; Partitura

f1 0 8192 9 1 1 .25
i1 0 5 100 1000 1 0 0 5      ; barrido pasa bajos
i1 5 5 200 1000 1 0 0 30    ; barrido pasa bajos, con un q mayor, una octava arriba
i1 10 5 100 1000 0 1 0 5    ; barrido pasa altos
i1 15 5 200 1000 0 1 0 30   ; barrido pasa altos, con un q mayor, una octava arriba
i1 20 5 100 1000 0 0 1 5    ; barrido pasa banda
i1 25 5 200 1000 0 0 1 30   ; barrido pasa banda, con un q mayor, una octava arriba
i1 30 5 200 2000 .4 .6 0    ; notch sweep - notch formed by combining
                                                ; highpass and lowpass outputs

e
```

49.9.5 AUTOR

Sean Costello
Seattle, Washington
1999

Nuevo en la versión 3.55

49.10 hilbert

`ar1, ar2 hilbert asig`

49.10.1 DESCRIPCIÓN

Una implementación IIR (Respuesta Infinita al Impulso) de un transformador de Hilbert.

49.10.2 EJECUCIÓN

asig - señal de entrada.

ar1 - salida coseno de *asig*.

ar2 - salida seno de *asig*.

hilbert es una implementación basada en una red de filtros IIR (Respuesta Infinita al Impulso) de banda ancha con una diferencia de fase de 90 grados cada uno. La entrada de **hilbert** es cualquier señal de audio, con un rango de frecuencias entre 15 Hz y 15 kHz.

Las salidas de **hilbert** tienen una respuesta a la frecuencia idéntica a la de la entrada (es decir, suenan igual), pero las dos salidas tienen una diferencia de fase constante de 90 grados, más o menos algún porcentaje de error a través del rango completo de frecuencias. Las salidas están en cuadratura. **hilbert** es útil en la implementación de muchas técnicas de procesamiento digital de señal que requieren una señal en cuadratura de fase. *ar1* corresponde a la salida coseno de **hilbert**, mientras que *ar2* corresponde a la salida seno. Las dos salidas tienen una diferencia de fase constante a través de todo el rango audible que corresponde a la relación de fase entre las ondas cosinusoidal y sinusoidal.

Internamente, **hilbert** está basado en dos filtros pasa-todo de sexto orden en paralelo. Cada filtro pasa-todo implementa una demora que aumenta con la frecuencia. La diferencia entre el retraso de fase de los filtros pasa-todo en paralelo en cualquier punto es aproximadamente 90 grados.

A diferencia del transformador de Hilbert basado en filtros FIR (Respuesta Finita al Impulso), la salida de **hilbert** no tiene una respuesta de fase lineal. Sin embargo, la estructura IIR usada en **hilbert** se calcula mucho más eficientemente y la respuesta de fase no lineal puede ser usada en la creación de interesantes efectos de audio, como en el segundo ejemplo de abajo.

49.10.3 EJEMPLOS

El primer ejemplo implementa un desplazamiento de frecuencia, o modulación de amplitud de sólo una banda lateral. El desplazamiento de frecuencia es similar a la modulación en anillo, excepto en que las bandas laterales superior e inferior están separadas en salidas individuales. Usando sólo una de las salidas, la señal de entrada puede ser "desafinada", donde los componentes armónicos de la señal están desalineados con respecto a los armónicos reales. Es decir, una señal con armónicos en 100, 200, 300, 400 y 500 Hz, desplazada 50 Hz hacia arriba, tendrá armónicos en 150, 250, 350, 450 y 550 Hz.

```
sr          =      44100
kr          =      4410
ksmps      =      10
nchnls     =      2

instr 1

idur       =      p3
ibegshift  =      p4          ; cantidad inicial de desplazamiento de frec.
                               ; puede ser positiva o negativa
iendshift  =      p5          ; cantidad final de desplazamiento de frec.
                               ; puede ser positiva o negativa
kfreq      =      linseg      ibegshift, idur, iendshift      ; Envolvente simple
                                               ; para determinar
                                               ; la cantidad de
                                               ; desplazamiento de frec.

ain        =      soundin     "supertest.wav"      ; usa tu propio sonido.
areal, aimag =      hilbert    ain                ; cuadratura de fase de salida
                                               ; derivada de la señal de entrada.

asin       =      oscili      1, kfreq, 1          ; Oscilador de cuadratura.
acos       =      oscili      1, kfreq, 1, .25
amod1      =      areal * acos                       ; indentidad trigonométrica.
                                               ; ver referencias para
                                               ; más detalles.

amod2      =      aimag * asin                       ; Tanto las frec. suma como resta
                                               ; pueden ser enviadas
                                               ; a la salida en seguida.

aupshift   =      (amod1 + amod2) * 0.7             ; aupshift corresponde a
                                               ; las frec. sumas, mientras que
adownshift = ( amod1 - amod2) * 0.7               ; adownshift corresponde a
                                               ; las frecuencias restas.
                                               ; Observa que cuando se
                                               ; suman las dos el resultado es
                                               ; idéntico a la salida de
                                               ; la modulación en anillo.

outs       =      aupshift, aupshift

endin

; a simple score

f1 0 16384 10 1      ; tabla sinusoidal para el oscilador de cuadratura
i1 0 29 0 200        ; empieza sin desplazamiento, termina con desplazamiento total
                       ; desplazamiento de frecuencia 200 Hz hacia arriba.
i1 30 29 0 -200     ; empieza sin desplazamiento, termina con desplazamiento total
                       ; desplazamiento de frecuencia 200 Hz hacia abajo.

e
```

El segundo ejemplo es una variación del primero, pero con la entrada retroalimentada por la salida. Con cantidades muy pequeñas de desplazamiento (es decir, entre 0 y +-6 Hz), el resultado es un sonido que ha sido descrito como desplazador de fase de Shepard. Aparecen cortes en el espectro, barridos constantemente en dirección opuesta a la del desplazamiento, produciendo un efecto de filtrado que recuerda a los glisandi sin fin de Risset.

```

sr           =      44100
kr           =      44100 ; kr DEBE ser igual a sr
                    ; para obtener un desplazador de Shepard
ksmps        =      1
nchnls       =      2

instr 2
afeedback      init      0      ; inicialización de la retroalimentación
idur           =          p3
ibegshift      =          p4      ; cantidad inicial de desplazamiento de frec.
                    ; puede ser positiva o negativa
iendshift      =          p5      ; cantidad final de desplazamiento de frec.
                    ; puede ser positiva o negativa
ifeed          =          p6      ; cantidad de retroalimentación. Cuanto mayor
                    ; el número, más pronunciado el efecto.
                    ; Experimenta para ver en que punto
                    ; comienza la oscilación.
                    ; (normalmente un valor de 1.4 es el máximo valor
                    ; de retroalimentación antes de la oscilación)

kfreq          linseg    ibegshift, idur, iendshift
ain            soundin   "supertest.wav"
areal, aimag   hilbert   ain + afeedback
asin          oscili    1, kfreq, 1
acos          oscili    1, kfreq, 1, .25
amod1         =          areal * acos
amod2         =          aimag * asin
aupshift      =          (amod1 + amod2) * 0.7
adownshift    =          (amod1 - amod2) * 0.7
afeedback     =          (amod1 - amod2) * .5 * ifeed ; feedback taken from
                    ; salida desplazada hacia abajo

outs         aupshift, aupshift

endin

; a simple score

f1 0 16384 10 1      ; tabla sinusoidal para el oscilador de cuadratura
i2 0 29 -.3 -.3 1.4 ; barrido hacia arriba, a una frecuencia de .3 veces/s
                    ; mucha retroalimentación
i2 30 30 .1 .1 1.4  ; barrido hacia abajo, a una frecuencia de .3 veces/s
                    ; mucha retroalimentación
i2 60 29 5 -5 1.4   ; el barrido va de .3 veces por segundo,
                    ; descendiendo en altura,
                    ; hasta .3 veces por segundo ascendiendo en altura
                    ; con una gran cantidad de retroalimentación

e

```

49.10.4 HISTORIAL TÉCNICO

El uso de redes de diferencia de fase en desplazadores de frecuencia fue introducido por Harald Bode (1). Bode y Bob Moog proporcionaron una excelente descripción de la implementación y el uso de un desplazador de frecuencia en el reino analógico (2). Esto sería una fuente inicial excelente para aquellos que desean explorar las posibilidades de la modulación de una única banda lateral. Bernie Hutchins proporciona más aplicaciones del desplazador de frecuencia, así como un análisis técnico detallado (3). Un artículo reciente de Scott Wardle (4) describe una implementación digital de un desplazador de frecuencia, así como algunas aplicaciones originales.

49.10.5 REFERENCIAS

1. H. Bode, "Solid State Audio Frequency Spectrum Shifter." AES Preprint No. 395 (1965).
2. H. Bode and R.A. Moog, "A High-Accuracy Frequency Shifter for Professional Audio Applications." *Journal of the Audio Engineering Society*, July/August 1972, vol. 20, no. 6, p. 453.
3. B. Hutchins. *Musical Engineer's Handbook* (Ithaca, NY: Electronotes, 1975), ch. 6a.
4. S. Wardle, "A Hilbert-Transformer Frequency Shifter for Audio." Disponible en línea en <http://www.iaa.upf.es/dafx98/papers/>.

49.10.6 AUTOR

Sean Costello
Seattle, Washington
1999

Nuevo en la versión 3.55

49.11 butterhp, butterlp, butterbp, butterbr

```
ar    butterhp    asig, kfreq [,iskip]
ar    butterlp    asig, kfreq [,iskip]
ar    butterbp    asig, kfreq, kband [,iskip]
ar    butterbr    asig, kfreq, kband [,iskip]
```

49.11.1 DESCRIPCIÓN

Implementaciones de filtros Butterworth de segundo orden pasa-altos, pasa-bajos, pasa-banda y para-banda, respectivamente. Nota: estos opcodes pueden escribirse también: **butlp, buthp, butbp, butbr**.

49.11.2 EJECUCIÓN

Estos opcodes son filtros Butterworth de segundo orden. Son sensiblemente más lentos que los filtros originales de Csound, pero ofrecen unas bandas de pasada (o parada) casi planas y una gran precisión en la atenuación para las bandas de corte.

asig - señal de entrada a filtrar.

kfreq - frecuencia central o de corte para cada uno de los filtros.

kband - ancho de banda de los filtros pasa-banda y para-banda.

iskip - si está presente y es distinto de 0, se omitirá la inicialización.

49.11.3 EJEMPLO

```
asig  rand          10000                ; señal de ruido blanco
alpf  butterlp     asig, 1000            ; corta las frecuencias mayores de 1KHz
ahpf  butterhp     asig, 500              ; pasa las frecuencias por encima de los 500Hz
abpf  butterbp     asig, 2000, 100        ; pasa sólo de 1950 a 2050 Hz
abrf  butterbr     asig, 4500, 200       ; corta sólo de 4400 a 4600 Hz
```

49.11.4 AUTOR

Paris Smaragdis
MIT, Cambridge
1995

49.12 filter2, zfilter2

ar	filter2	asig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
kr	filter2	ksig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
ar	zfilter2	asig, kdamp, kfreq, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN

49.12.1 DESCRIPCIÓN

Filtro de propósito general definido por el usuario con control polar variante en el tiempo. Los coeficientes del filtro se expresan en la siguiente ecuación:

$$(1) *y(n) = b0 * x[n] + b1 * x[n-1] + \dots + bM * x[n-M] - a1 * y[n-1] - \dots - aN * y[n-N]$$

la función del sistema se representa por:

$$H(Z) = \frac{B(Z)}{A(Z)} = \frac{b0 + b1 * Z^{-1} + \dots + bM * Z^{-M}}{1 + a1 * Z^{-1} + \dots + aN * Z^{-N}}$$

49.12.2 INICIALIZACIÓN

El la inicialización el número de ceros y polos del filtro se especifican junto con sus correspondientes coeficientes. Dichos coeficientes deben ser obtenidos usando una aplicación externa de diseño de filtros, como Matlab, y especificada directamente o cargada en una tabla con **GEN01**. Con **zfilter2**, las raíces de los polinomios característicos se calculan en la inicialización para que las operaciones de control de polos puedan ser implementadas sin problemas.

49.12.3 EJECUCIÓN.

Los opcodes **filter2** filtran la señal usando una rejilla de filtros digitales, sin capacidad de control sobre las variaciones en el tiempo. **filter2** usa operaciones adicionales de corte polar radial y deformación polar angular en el plano Z.

El corte polar incrementa la magnitud de los polos a lo largo de líneas radiales en el plano Z. Esto produce el mismo efecto que alterar la duración de los bucles en los filtros. La variable de control *kdamp* es un parámetro de amortiguación. Los valores positivos (de 0.01 a 0.99) aumentan la duración del bucle del filtro (valores altos de Q), mientras que los negativos (de -0.01 a -0.99) lo disminuyen (valores bajos de Q).

La deformación polar modifica la frecuencia de los polos moviéndolos a lo largo de trayectorias angulares en el plano Z.

Esta operación deja inalterada la curva de respuesta de la amplitud pero modifica la respuesta a la frecuencia según un factor constante (preservando ceros y polos). La variable de control *kfreq* determina el factor de "deformación" de la respuesta a la frecuencia. Los valores positivos (de 0.01 a 0.99) aumentan las frecuencias hacia los polos, mientras que los negativos (de -0.01 a -0.99) disminuyen las frecuencias hacia los ceros.

Debido a que **filter2** proporciona filtros recursivos de propósito general, puede usarse para diseñar un gran número de algoritmos de procesamiento general de señal (DSP). Por ejemplo, se puede diseñar una "guía de onda" digital para el modelado de un instrumento, usando un par de opcodes **delayr** y **delayw** junto a **filter2**.

49.12.4 EJEMPLO

Un filtro FIR (del inglés *Finite Impulse Response*) pasa bajos de primer orden, con respuesta lineal a la fase, que se aplica a una señal de control:

```
k1 filter2 ksig, 2, 0, 0.5, 0.5 ; filtro FIR sobre una señal de control
```

Un filtro IIR (del inglés *Infinite Impulse Response*) de segundo orden controlable por el usuario aplicado a una señal de audio:

```
a1 zfilter2 asig, kdamp, kfreq, 1, 2, 1, ia1, ia2 ; filtro IIR controlable  
; sobre una señal de audio
```

49.12.5 NOMBRES DESECHADOS

La versión de **filter2** que opera con señales de control (k-) se llamaba originalmente **kfilter2**. A partir de la versión 3.493 de Csound se desechó ese nombre. **filter2** debe ser usado ahora en su lugar. El opcode mismo decide si trabaja con señales de control o de audio, según su argumento de salida.

49.12.6 AUTOR

Michael A. Casey
M.I.T.
Cambridge, Mass.
1997

50 MODIFICADORES DE SEÑAL: FILTROS ESPECIALIZADOS

50.1 nlfilt

ar nlfilt ain, ka, kb, kd, kL, kC

50.1.1 DESCRIPCIÓN

Implementa el filtro definido por $Y\{n\} = a Y\{n-1\} + b Y\{n-2\} + d Y^{2\{n-L\} + X\{n\} - C$, descrito en "Dobson and Fitch (ICMC'96)"

50.1.2 EJEMPLO

i) efecto no lineal:

a = b = 0
d = 0.8, 0.9, 0.7
C = 0.4, 0.5, 0.6
L = 20

Lo que afecta al registro bajo principalmente, aunque los efectos son audibles en el rango entero. Puede ser útil para colorear sonidos percusivos y para enfatizar notas arbitrariamente.

ii) Pasa-bajos no lineal:

a = 0.4
b = 0.2
d = 0.7
C = 0.11
L = 20, ... 200

Hay problemas de inestabilidad con esta variante y el efecto es más pronunciado en el registro bajo. De todas formas es muy parecido a un filtro peine puro. Valores pequeños de L pueden añadir efectos de ataque al sonido.

iii) Pasa-altos no lineal:

a = 0.35
b = -0.3
d = 0.95
C = 0.2, ... 0.4
L = 200

iv) Pasa-altos no lineal:

$$a = 0.7$$

$$b = -0.2, \dots 0.5$$

$$d = 0.9$$

$$C = 0.12, \dots 0.24$$

$$L = 500, 10$$

Es probable que la versión pasa-altos sea un poco inestable. Añade brillo a los registros medio-altos. Con un valor de retardo largo de L se produce una especie de reverberación, mientras que valores pequeños parecerán producir algo parecido a regiones de formantes. Se producen cambios de color arbitrarios y resonancias al cambiar la altura. Funciona bien con notas individuales.

Precaución: los rangos "útiles" no han sido aún mapeados.

50.1.3 AUTOR

John ffitch

Universidad de Bath/Codemist Ltd.

Bath, Reino Unido

1997

50.2 pareq

ar **pareq** asig, kc, iv, iq, imode

50.2.1 DESCRIPCIÓN

Implementación de los filtros ecualizadores paramétricos de Zoelzer.

50.2.2 INICIALIZACIÓN

iv – cantidad de estímulo o corte. Los valores positivos se toman como estímulos, los negativos como cortes.

iq – factor Q del filtro (la raíz cuadrada de 0.5 da una resonancia nula)

imode – modo de operación.

0: Pico

1: Corte bajo.

2: Corte alto.

50.2.3 EJECUCIÓN

kc – frecuencia central en modo pico, lateral en modo corte.

asig – señal de entrada.

50.2.4 EJEMPLO

```
instr 15
ifc      =          p4          ; Central / lateral
iq       =          p5          ; la raíz cuadrada de 0.5 da una resonancia nula
iv       =          ampdb(p6)   ; Volumen de estímulo o corte
imode    =          p7          ; Modo 0=Pico, 1=corte bajo, 2=corte alto
kfc      linseg      ifc*2, p3, ifc/2
asig     rand       5000 ; fuente de números aleatorios para la comprobación
aout     pareq     asig, kfc, iv, iq, imode ; ecualización paramétrica
aout     outs      aout, aout ; devuelve la salida

endin
; Partitura
;    Sta    Dur    Fcenter    Q    Boost/Cut (dB)    Mode
i15    0    1    10000    .2    12    1
i15    +    .    5000    .2    12    1
i15    .    .    1000    .707    -12    2
i15    .    .    5000    .1    -12    0
e
```

50.2.5 AUTOR

Hans Mikelson
Diciembre, 1998
Nuevo en la Version 3.50

50.3 dcblock

ar dcblock asig[, ig]

50.3.1 DESCRIPCIÓN

Implementa un filtro en bloque de DCs

$$Y[i] = X[i] - X[i-1] + (igain * Y[i-1])$$

Basado en el trabajo de Perry Cook

50.3.2 INICIALIZACIÓN

igain - ganancia del filtro, 0.99 por defecto.

50.3.3 EJECUCIÓN

ain – señal de audio de entrada.

50.3.4 AUTOR

John ffitch
Universidad de Bath, Codemist Ltd.
Bath, Reino Unido
1998

51 MODIFICADORES DE SEÑAL: MODIFICADORES DE ENVOLVENTE

51.1 *linen*, *linenr*, *envlpx*, *envlpxr*

kr	linen	kamp, irise, idur, idec
ar	linen	xamp, irise, idur, idec
kr	linenr	kamp, irise, idec, iatdec
ar	linenr	xamp, irise, idec, iatdec
kr	envlpx	kamp, irise, idur, idec, ifn, iatss, iatdec[,ixmod]
ar	envlpx	xamp, irise, idur, idec, ifn, iatss, iatdec[,ixmod]
kr	envlpxr	kamp, irise, idec, ifn, iatss, iatdec[, ixmod[, irind]]
ar	envlpxr	xamp, irise, idec, ifn, iatss, iatdec[, ixmod[, irind]]

51.1.1 DESCRIPCIÓN

linen - aplica un patrón lineal de ataque y caída a la amplitud de cualquier señal de entrada.

envlpx - aplica un envolvente consistente en tres segmentos:

- 1) función de la curva de ataque almacenada
- 2) período exponencial pseudo-estable modificado
- 3) caída exponencial.

linenr, **envlpxr** - son como los anteriores, excepto en que sólo se entra en el segmento final cuando se percibe un mensaje MIDI de nota desactivada, momento en el cual la nota se prolonga según la duración del segmento de caída.

51.1.2 INICIALIZACIÓN

irise - duración en segundos del ataque. Un valor 0 o negativo significa que no hay curva de ataque.

idur - duración total en segundos. Un valor 0 o negativo causará la omisión de la inicialización.

idec - duración en segundos de la caída. 0 significa que no hay curva de caída. Un valor *idec* mayor que *idur* causará una caída truncada.

irind (opcional) - indicador de independencia. Si es dejado a 0, la duración de la caída (*idec*) influirá en la prolongación de la nota actual después del mensaje de desactivación. Si no es 0, la duración *idec* es independiente de la prolongación de la nota (ver abajo). El valor por defecto es 0.

ifn - número de la tabla de función de la curva de ataque, que debe tener un elemento límite añadido.

iatss - factor de atenuación, por el cual el último valor del ataque de **envlpx** es modificado durante el período pseudo estable de la nota. Un factor mayor que 1 causa una subida exponencial, y uno menor que 1 una caída exponencial. Un valor 1 proporciona un período verdaderamente estable, manteniendo fijo el último valor de la tabla de función del ataque. Observa que la amortiguación de *iatss* no es una constante

fija (como en un piano), sino que es sensible a la duración de la nota. Sin embargo, si *iatss* es negativo (o si el período estable es menor que 4 ciclos de control) se usará un factor de atenuación fijo igual al valor absoluto de *iatss* por segundo. Un valor de 0 es ilegal.

iatdec - factor de atenuación por el cual el último valor del período estable de la nota es exponencialmente reducido a lo largo de la duración de la caída. Este valor debe ser positivo y es normalmente del orden de 0.01. Un valor excesivamente grande o pequeño produce un corte fácilmente audible. Un valor 0 o negativo es ilegal.

ixmod (opcional, entre + - .9 aprox.) - modificador de la curva exponencial, que influye en la pendiente de su trayectoria durante el período estable. Valores menores que 0 causarán un ataque o caída "acelerados" hacia el objetivo (es decir, un efecto de *subito piano*). Valores mayores que 0 producirán un ataque o caída no tan pronunciados. El valor por defecto es 0 (curva exponencial sin modificar).

51.1.3 EJECUCIÓN

Se aplican modificaciones en el ataque de la nota en los primeros *irise* segundos, y en la caída desde *idur* - *idec*. Si estos períodos están separados en el tiempo, habrá un período estable durante el cual la amplitud *amp* se mantendrá con el mismo valor (**linen**) o será modificada por el primer patrón exponencial (**envlpx**). Si el ataque y la caída de **linen** se pisan, se producirán ambas modificaciones en el lapso de tiempo que dure el solapamiento. En **envlpx** ello causará una caída truncada. Si se sobrepasa la duración total *idur* en la ejecución, la caída final continuará en la misma dirección, cayendo a valores negativos en **linen** o tendiendo asintóticamente a 0 en **envlpx**.

linenr pertenece al grupo de opcodes en Csound que contienen un sensor de desactivación de nota y un prolongador temporal de caída. Cuando percibe bien un final de evento de la partitura, bien un mensaje MIDI de desactivación de nota, inmediatamente prolonga en el tiempo la ejecución del instrumento actual durante *idec* segundos, y luego aplica una caída exponencial tendiendo al valor del factor *iatdec*. Cuando se usan dos o más unidades en un instrumento, la nota se prolonga por el mayor valor de *idec*.

linenr, **envlpxr** son ejemplos de las unidades "r" especiales de Csound que contienen un sensor de desactivación de nota y un prolongador de la duración de caída. A menos que se independicen por medio de *irind*, donde cada uno percibe un final de evento en la partitura o un mensaje MIDI de desactivación de nota, prolongarán el tiempo de ejecución del instrumento actual durante *idec* segundos y luego aplicarán una caída (como se describió antes) desde dondequiera que se encuentren en la duración de la nota. Estas unidades "r" pueden ser también sensibles a las velocidades de desactivación de nota MIDI (ver **veloffs**). Si el indicador *irind* es distinto de 0, la duración total de la ejecución no se verá afectada por los datos de nota desactivada o velocidades de desactivación.

51.1.4 MÚLTIPLES UNIDADES "R"

Cuando se encuentran dos o más unidades "r" en el mismo instrumento, es normal dejar que sólo una de ellas influya en la duración total de la nota. Esta será, casi siempre, la unidad maestra de amplitud. Otras unidades, controlando, por ejemplo, el movimiento de un filtro, pueden ser sensibles a los comandos de desactivación mientras que no afecten a la duración total de la nota (haciéndolos independientes con un *irind* distinto de 0).

Dependiendo de su *idec* propio, las unidades "r" pueden o pueden no alcanzar sus destinos finales antes de que acabe el instrumento. Si lo hacen, simplemente mantienen su valor final hasta que el instrumento termina. Si dos o más unidades "r" son simultáneamente maestras, la prolongación de la nota se efectuará de acuerdo al valor mayor de *idec*.

52 MODIFICADORES DE SEÑAL: MODIFICADORES DE AMPLITUD

52.1 rms, gain, balance

```
kr    rms      asig[, ihp, iskip]
ar    gain     asig, krms[, ihp, iskip]
ar    balance  asig, acomp[, ihp, iskip]
```

52.1.1 DESCRIPCIÓN

La potencia media (rms, del inglés *root-mean-square*) de una señal puede ser consultada, indicada o ajustada según una señal de comparación.

52.1.2 INICIALIZACIÓN

ihp (opcional) - valor de potencia media (en Hz) de un filtro pasa-bajos interno especial. El valor por defecto es 10.

iskip (opcional) - disposición inicial del espacio para los datos internos (ver **reson**). El valor por defecto es 0.

52.1.3 EJECUCIÓN

rms devuelve valores de control *kr* que especifican en cada momento el valor de potencia media (rms) de la señal de entrada *asig*. Esta unidad no es un modificador de señal, pero sirve para calibrar la potencia de una señal.

gain proporciona un medio de modificar la amplitud de *asig* de tal manera que la salida *ar* tendrá una potencia media igual al valor señalado por *krms*. **rms** y **gain** usados conjuntamente (con el mismo valor *ihp*) producirán el mismo efecto que el opcode **balance**.

balance devuelve una versión de *asig* modificada en amplitud, de manera que su valor de potencia media sea igual al de una señal de comparación *acom*. Así una señal que ha sufrido pérdida de potencia (por ejemplo, al pasarla por un banco de filtros) puede ser restaurada a su potencia original al compararla, por ejemplo, con su propia fuente sin modificar. Debe ser observado que **gain** y **balance** sólo proporcionan modificaciones en la amplitud. Las señales de salida no se ven afectadas en ningún otro aspecto.

52.1.4 EJEMPLO

```
asrc  buzz      10000,440, sr/440, 1    ; tren de ondas pulso de banda limitada
a1    reson     asrc, 1000,100        ; enviado a traves de
a2    reson     a1,3000,500           ; 2 filtros
afin  balance   a2, asrc              ; y luego balanceado con su fuente original
```

52.2 dam

ar **dam** ain, kthreshold, ico_{mp1}, ico_{mp2}, rtime, ftime

52.2.1 DESCRIPCIÓN

Este opcode modifica dinámicamente un valor *gain* aplicado a una señal de entrada "*ain*", comparando su nivel de potencia con un umbral dado. La señal será comprimida o expandida con diferentes factores dependiendo de si está por encima o por debajo del umbral.

52.2.2 INICIALIZACIÓN

ico_{mp1} - coeficiente de compresión para la zona superior.

ico_{mp2} - coeficiente de compresión para la zona inferior.

rtime - duración de la subida de ganancia, es decir, la duración en la cual se permite el aumento del factor de ganancia de una unidad.

ftime - duración de la bajada de ganancia, es decir, la duración en la cual se permite el descenso del factor de ganancia de una unidad.

52.2.3 EJECUCIÓN

aout - señal de salida.

ain - señal de entrada.

kthreshold - nivel de sonido que actuará de umbral. Puede ser modificado en tiempo de ejecución (k-).

Nota sobre los factores de compresión: un coeficiente de compresión igual a 1 deja el sonido inalterado. Proporcionando al coeficiente valores más pequeños que 1, se comprimirá la señal (es decir, se reducirá su volumen), mientras que coeficientes mayores que 1 expandirán la señal (es decir, aumentarán su volumen).

52.2.4 AUTOR

Marc Resibois

Belgica

1997

53 MODIFICADORES DE SEÑAL: LIMITADORES DE SEÑAL

53.1 limit, mirror, wrap

ir	limit	isig, ilow, ihigh
kr	limit	ksig, klow, khigh
ar	limit	asig, klow, khigh
ir	wrap	isig, ilow, ihigh
kr	wrap	ksig, klow, khigh
ar	wrap	asig, klow, khigh
ir	mirror	isig, ilow, ihigh
kr	mirror	ksig, klow, khigh
ar	mirror	asig, klow, khigh

53.1.1 DESCRIPCIÓN

Modifica la señal de varias maneras.

53.1.2 INICIALIZACIÓN

isig – señal de entrada.

ilow - umbral inferior.

ihigh - umbral superior.

53.1.3 EJECUCIÓN

xsig - señal de entrada.

xlow - umbral inferior.

xhigh - umbral superior.

limit aplica los límites superior e inferior en la señal *xsig* que procesan. Si *xhigh* es menor que *xlow*, la salida será la media de ambas y no será afectada por *xsig*. **mirror** "refleja" la señal que excede los umbrales inferior y superior. **wrap** pliega la señal que excede los umbrales inferior y superior.

Estos opcodes son útiles en algunas situaciones, como por ejemplo en la indexación de tablas o para recortar o modelar señales de tipo i-, k-, o a-. **wrap** es también útil para usar cíclicamente los datos de una tabla cuando el índice máximo no es una potencia de 2 (ver **table** y **tablei**). Otro uso de **wrap** es la repetición cíclica de eventos, con cualquier longitud de ciclo.

53.1.4 AUTORES

Gabriel Maldonado (**wrap**, **mirror**)
Italia
Nuevo en la versión 3.49

Robin Whittle (**limit**)
Australia
Nuevo en la versión 3.46

54 MODIFICADORES DE SEÑAL: LÍNEAS DE RETARDO

54.1 delayr, delayw, delay, delayl

```
ar    delayr    idlt[, iskip]
      delayw    asig
ar    delay     asig, idlt[, iskip]
ar    delayl    asig[, iskip]
```

54.1.1 DESCRIPCIÓN

Permiten que una señal pueda ser leída desde o escrita en una línea de retardo, o simplemente pueda ser automáticamente retardada por un intervalo de tiempo determinado.

54.1.2 INICIALIZACIÓN

idlt - tiempo de retardo requerido en segundos. Puede ser tan grande como memoria disponible haya. El espacio requerido para *n* segundos de retardo es de $(4n * \mathbf{sr})$ bytes. Es asignado en la primera inicialización del instrumento, y desechada al final de una sección de la partitura.

iskip (opcional) - disposición inicial del espacio de los datos del bucle de retardo (ver **reson**). El valor por defecto es 0.

54.1.3 EJECUCIÓN

delayr lee una línea digital de retardo establecida automáticamente, en la que la señal guardada ha estado residente durante *idt* segundos. Esta unidad debe preceder y ser emparejada con una unidad **delayw**. Puede participar en el proceso cualquier otra sentencia de Csound.

delayw escribe la señal *asig* en la línea de retardo establecida por la unidad **delayr** precedente. Entendidas en parejas, estas dos unidades permiten la formación o modificación de bucles retroalimentados, etc. Sin embargo, hay un límite inferior en el valor de *idt*, que debe ser de al menos 1 ciclo de control de duración (o $1/kr$).

Es posible entrelazar pares de unidades **delayr/delayw**. Ahora también se puede empezar un nuevo par **delayr/delayw** antes de terminar un par precedente. Para tales pares, la primera unidad **delayr** va asociada con la primera unidad **delayw**, la segunda unidad **delayr** con la segunda unidad **delayw** y así sucesivamente. De esta forma es posible implementar retroalimentaciones cruzadas en parejas que serán completadas dentro del mismo ciclo de control. Ver ejemplo 2 (Esta opción fue añadida en la versión 3.57 por Jens Groh y John ffitich).

delay es una unidad compuesta de las dos unidades anteriores, ambas leyendo y escribiendo en su propio espacio de almacenamiento, pudiendo realizar así desplazamientos temporales de la señal, aunque la modificación de la retroalimentación no es posible. No hay período mínimo de retardo.

delay1 es una forma especial de retardo que sirve para retardar una señal de audio *asig* tan sólo una muestra. Funciona así de forma similar a "**delay asig, 1/sr**" pero es más eficiente tanto en tiempo de ejecución como en espacio consumido. Esta unidad es particularmente útil en la construcción de filtros no recursivos de propósito general.

54.1.4 EJEMPLOS

Ejemplo 1:

```
        tigoto      contin          ; excepto en una ligadura
a2      delay     a1, .05, 0       ; retrasa 50 msecs la señal
contin:
```

Ejemplo 2:

```
        ainput1 = .....
        ainput2 = .....

;Lee la señal retardada, primera aparición de delayr:
        adly1 delayr 0.11

; Lee la señal retardada, segunda aparición de delayr:
        adly2 delayr 0.07

;Realiza ciertas manipulaciones de pares cruzados:
        afdbk1 = 0.7 * adly1 + 0.7 * adly2 + ainput1
        afdbk2 = -0.7 * adly1 + 0.7 * adly2 + ainput2

;La señal de retroalimentación se asocia con la primera aparición de delayr:
        delayw afdbk1

; La señal de retroalimentación se asocia con la segunda aparición de delayr:
        delayw afdbk2
        outs
```

54.2 **deltap**, **deltapi**, **deltapn**, **deltap3**

ar	deltap	<i>kdlt</i>
ar	deltapi	<i>xdlt</i>
ar	deltapn	<i>xnumsamps</i>
ar	deltap3	<i>kdlt</i>

54.2.1 DESCRIPCIÓN

"Pincha" una línea de retardo a distintos intervalos de tiempo.

54.2.2 EJECUCIÓN

Estas unidades pueden introducirse entre un par **delayr/delayw**, extrayendo el audio retardado calculado por dichas unidades desde el instante *idlt* (expresado en segundos). Puede haber cualquier número de unidades **deltap** y/o **deltapi** entre un par de opcodes de lectura o escritura de líneas de retardo. Cada una capturará el audio retardado sin cambios en la amplitud original.

deltap extrae el sonido retardado leyendo directamente las muestras almacenadas por las unidades **delayr/delayw** en su espacio de trabajo. **deltapi**, en cambio, extrae el sonido interpolando los valores de muestras sucesivas. Con este procedimiento **deltapi** calcula el tiempo de retardo con más exactitud, pero también tardará el doble en ejecutarse.

Los argumentos *kdlt* y *xdlt* especifican el tiempo de retardo expresado en segundos. Cada uno de ellos puede tomar valores de entre 1 ciclo de control hasta el tiempo total de retardo del par de opcodes de lectura-escritura. Sin embargo, debido a que no hay ningún proceso interno de comprobación de estos márgenes, el usuario es totalmente responsable de que estos no se excedan. Ambos argumentos pueden ser constantes, variables, o una señal variante en el tiempo. El argumento *xdlt* de **deltapi** implica que se pueden implementar retardos variables a frecuencia de audio. **deltapn** es idéntico a **deltapi** excepto en que el tiempo de retardo se especifica en número de muestras, en vez de en segundos (Hans Mikelson). **deltap3** es experimental y usa interpolación cúbica. (Nuevo en la Versión 3.50)

Estas unidades pueden usarse para diseñar líneas de retardo múltiples o redes con retroalimentación múltiple. Pueden aplicarse a una frecuencia constante o variable, y son útiles para la construcción de efectos de coro, armonizadores, y desplazamientos Doppler. Las unidades que se aplican a una frecuencia constante (o las que varían lentamente) no necesitan usar el proceso de interpolación. Se pueden realizar bien con **deltap**. Sin embargo, las que cambian a una velocidad media o rápida, necesitarán la ayuda extra de **deltapi**.

Es posible entrelazar pares de unidades **delayr/delayw**. Para asociar un canal de retardo con una unidad **delayr** específica, no sólo tiene que ser colocado entre esa unidad **delayr** y la consiguiente **delayw**, sino que debe preceder también cualquier unidad **delayr** psoterior. Ver ejemplo 2 (Esta opción fue añadida en la versión 3.57 por Jens Groh y John ffitich).

N.B: los tiempos de retardo que varían a frecuencia de control no usan el proceso de interpolación interna, sino que proporcionan desplazamientos temporales "escalonados" de las muestras de audio. Esto no representa un problema para tiempos de retardo que varíen lentamente, pero para los que cambian a una velocidad media o rápida (es decir, a frecuencias de audio) se debe proporcionar un desplazamiento temporal de mayor resolución en el argumento de entrada (a frecuencia de audio).

54.2.3 EJEMPLO

Ejemplo 1:

```

asource      buzz          1, 440, 20, 1
atime        linseg       1, p3/2,.01, p3/2,1    ; traza la duración en segundos
ampfac       =             1/atime/atime      ; calcula un factor de amplitud
adump        delayr       1                ; ajusta la duración máxima
amove        deltapi      atime             ; retrasa el sonido fuente
              delayw      asource           ;
              out         amove * ampfac

```

Ejemplo 2:

```

ainput1 = .....
ainput2 = .....
kdlyt1 = .....
kdlyt2 = .....

;Lee la señal retardada, primera aparición de delayr:

adump delayr 4.0
adly1 deltap kdlyt1      ; asociado con la primera unidad delayr

; Lee la señal retardada, segunda aparición de delayr:

adump delayr 4.0
adly2 deltap kdlyt2      ; asociado con la segunda unidad delayr

;Realiza ciertas manipulaciones de pares cruzados:

afdbk1 = 0.7 * adly1 + 0.7 * adly2 + ainput1
afdbk2 = -0.7 * adly1 + 0.7 * adly2 + ainput2

;La señal de retroalimentación se asocia con la primera aparición de delayr:

delayw afdbk1

; La señal de retroalimentación se asocia con la segunda aparición de delayr:

delayw afdbk2
outs adly1, adly2

```

54.3 multitap

```
ar    multitap    asig, itime1, igain1, itime2, igain2 ...
```

54.3.1 DESCRIPCIÓN

Implementación de una línea de retardo múltiple.

54.3.2 INICIALIZACIÓN

Los argumentos *itime* e *igain* ajustan el tiempo de retardo y la ganancia de cada repetición.

La línea de retardo es alimentada por la señal *asig*.

54.3.3 EJEMPLO

```
a1    oscil        1000, 100, 1
a2    multitap    a1, 1.2, .5, 1.4, .2
      out         a2
```

Esto proporciona dos retardos: uno con tiempo de retardo de 1.2 y una ganancia de .5 y otro con un tiempo de retardo de 1.4 y una ganancia de .2.

54.3.4 AUTOR

Paris Smaragdis
MIT, Cambridge
1996

54.4 vdelay, vdelay3

```
ar    vdelay    asig, adel, imaxdel [, iskip]
ar    vdelay3   asig, adel, imaxdel [, iskip]
```

54.4.1 DESCRIPCIÓN

Aplican un retardo variable en el tiempo usando un proceso de interpolación, no muy diferente de la implementación de **deltapi**, pero más fácil de usar. **vdelay3** es experimental y es igual a **vdelay**, pero con interpolación cúbica (Nuevo en la versión 3.50)

54.4.2 INICIALIZACIÓN

imaxdel - valor máximo del retardo en milisegundos. Si *adel* recibe un valor mayor que *imaxdel* toma el valor de éste último, cosa que debería evitarse.

iskip - si está presente y no es 0, omite la inicialización.

54.4.3 EJECUCIÓN

Con estos opcodes es posible conseguir efectos Doppler, de coros o flangings.

asig - señal de entrada.

adel - valor actual del tiempo de retardo en milisegundos. Observa que las funciones lineales no tienen efectos de variación de altura.

Las variaciones rápidas del valor de *adel* pueden causar discontinuidades en la forma de onda que producen ruido.

54.4.4 EJEMPLO

```
f1 0 8192 10 1
ims = 100 ; tiempo máx. de retardo en segs.
a1 oscil 10000, 1737, 1 ; crea una señal
a2 oscil ims/2, 1/p3, 1 ; crea una señal de baja frecuencia (LFO)
a2 = a2 + ims/2 ; Desplaza la LFO para que sea positiva
a3 vdelay a1, a2, ims ; Usa la LFO para controlar la línea de retardo.
out a3
```

Dos cosas importantes: primero, el tiempo de retardo debe ser siempre positivo; segundo, incluso aunque el tiempo de retardo pueda ser controlado en frecuencia de control (k-), no se aconseja hacerlo, ya que variaciones temporales súbitas pueden ocasionar "clicks" en la señal.

54.4.5 AUTOR

Paris Smaragdis
MIT, Cambridge
1995

55 MODIFICADORES DE SEÑAL: REVERBERACIÓN

55.1 comb, alpass, reverb

ar	comb	asig, krvt, ilpt[, iskip]
ar	alpass	asig, krvt, ilpt[, iskip]
ar	reverb	asig, krvt[, iskip]

55.1.1 DESCRIPCIÓN

Reverberación de una señal de entrada, durante *krvt* segundos, con una respuesta de frecuencia "coloreada" (**comb**), plana (**alpass**) o la que proporciona una sala normal (**reverb**).

55.1.2 INICIALIZACIÓN

ilpt - duración del bucle en segundos, que determina la densidad del eco en la reverberación. Esto, a su vez, caracteriza el "color" del filtro peine (**comb**) cuya curva de respuesta a la frecuencia contendrá *ilpt* * *sr*/2 picos distribuidos regularmente entre 0 y *sr*/2 (es decir, la frecuencia Nyquist). La duración del bucle puede ser tan larga como lo permita la memoria disponible. El espacio requerido para un bucle de *n* segundos es de $4n * sr$ bytes. El espacio de retardo de los opcodes **comb** y **alpass** se asigna y se devuelve de la misma manera que en **delay**.

iskip (opcional) - disposición inicial del espacio de los datos del bucle de retardo (ver **reson**). El valor por defecto es 0.

55.1.3 EJECUCIÓN

Estos filtros repiten la entrada con una densidad de eco determinada por la duración del bucle *ilpt*. La velocidad de atenuación es independiente y viene determinada por *krvt*, o tiempo de reverberación (definido como el tiempo, en segundos, necesario para que una señal disminuya 60 dB con respecto a su amplitud original). La salida de un filtro peine aparecerá sólo después de *ilpt* segundos; la salida de **alpass** empezará inmediatamente.

Una unidad **reverb** estándar se compone de cuatro filtros peine (**comb**) en paralelo seguido de dos unidades **alpass** en serie. Las duraciones de los bucles están ajustadas para simular una óptima respuesta natural de sala. Los requerimientos internos de almacenamiento de la unidad son proporcionales únicamente a la frecuencia de muestreo. Las unidades **comb**, **alpass**, **delay**, **tone** y otras proporcionan los medios necesarios para experimentar con diseños de reverberación alternativos.

Debido a que la salida de la **reverb** estándar empieza sólo después de aproximadamente 1/20 segundos de retraso, a menudo con menos de tres cuartas partes de la potencia original, normalmente se opta por reproducir la señal fuente junto con la reverberada. Del mismo modo, debido a que el sonido reverberado persiste después de que el sonido fuente haya cesado, es normal poner la unidad **reverb** en un instrumento aparte al cual se pasan las señales de audio como **variables globales** y se deja el instrumento activo a lo largo de toda la pieza.

55.1.4 EJEMPLO

```
gal  init      0                ; inicializa un mezclador de audio
instr 1                ; instr (puede haber varias copias)
a1   oscili    8000, cspch(p5), 1 ; genera una señal fuente
      out      a1                ; reproduce el sonido tal cual
gal  =            gal + a1        ; y lo añade al mezclador
endin
instr 99                ; (el número más alto de instrumento se
ejecuta                ; el último)
a3   reverb    gal, 1.5          ; reverbera lo que haya en gal
      out      a3                ; y reproduce el resultado
gal  =            0                ; vacía el mezclador para la próxima señal
endin
```

55.2 reverb2, nreverb

```
ar    reverb2    asig, ktime, khdif [,iskip]
ar    nreverb   asig, ktime, khdif [,iskip]
```

55.2.1 DESCRIPCIÓN

Reverberación consistente en 6 filtros peine-pasa-bajos en paralelo alimentando una serie de 5 filtros pasa-altos. **nreverb** reemplaza a **reverb2** (versión 3.48), siendo ambos idénticos.

55.2.2 INICIALIZACIÓN

iskip - si está presente y es distinto de 0, omite la inicialización.

55.2.3 EJECUCIÓN

Se reverbera la señal de entrada *asig* durante *ktime* segundos. El parámetro *khdif* controla la cantidad de difusión de las altas frecuencias. Los valores de *khdif* deben estar entre 0 y 1. Si *khdif* es puesto a 0, todas las frecuencias decaen con la misma velocidad. Si *khdif* es 1, las altas frecuencias decaen más rápido que las bajas.

55.2.4 EJEMPLO

```
a1    oscil      10000, 100, 1
a2    reverb2   a1, 2.5, .3
out   out       a1 + a2 * .2
```

Esto provoca una reverberación de 2.5 segundos con una atenuación de las altas frecuencias más rápida que de las bajas.

55.2.5 AUTORES

Paris Smaragdis (**reverb2**)
MIT, Cambridge
1995

Richard Karpen (**nreverb**)
Seattle, Wash
1998

55.3 nestedap

ar **nestedap** asig, imode, imaxdel, idel1, igain1[, idel2, igain2[, idel3, igain3]]

55.3.1 DESCRIPCIÓN

Aplica tres diferentes filtros pasa-todo anidados, útil para diseñar reverberaciones.

55.3.2 INICIALIZACIÓN

imode –modo de operación del filtro:

1 = un sólo filtro pasa-todo

2 = un filtro pasa-todo anidado

3 = dos filtros pasa-todo anidados

idel1, *idel2*, *idel3* – tiempos de retardo de las etapas del filtro. Los tiempos de retardo van en segundos y deben ser mayores que 0. *idel1* debe ser mayor que la suma de *idel2* y *idel3*.

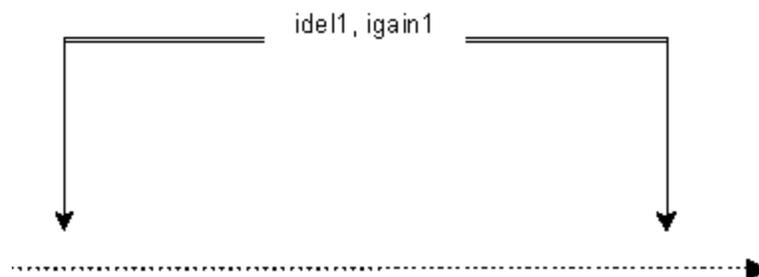
igain1, *igain2*, *igain3* – ganancia de las etapas del filtro.

imaxdel –será necesario si se implementan más adelante retardos variables a frecuencia de control. No se usa actualmente.

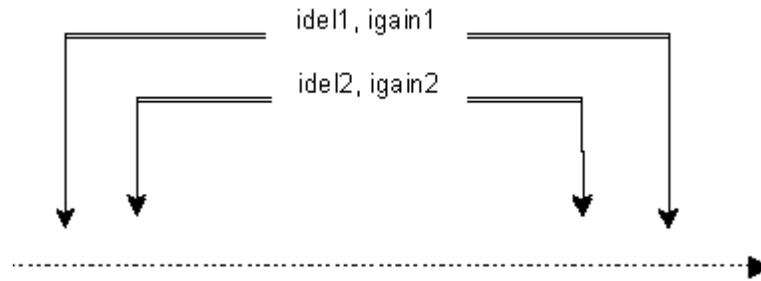
55.3.3 EJECUCIÓN

asig – señal de entrada.

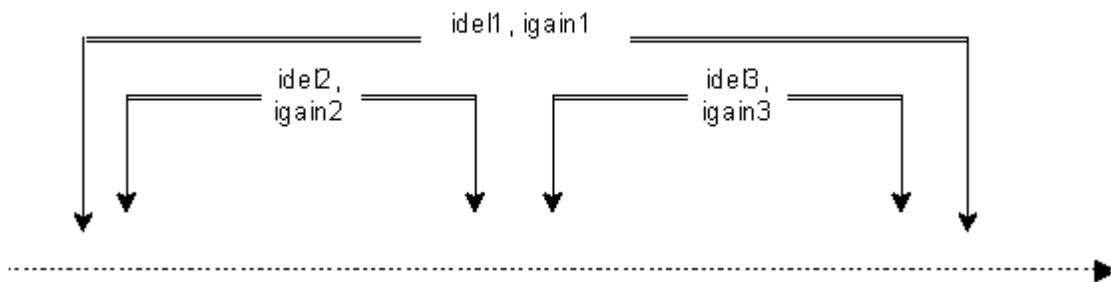
Si *imode* = 1, el filtro toma la forma.



Si $imode = 2$, el filtro toma la forma:



Si $imode = 3$, el filtro toma la forma:



55.3.4 EJEMPLO

```

instr 5
insnd          =          p4
gasig          = diskin  insnd, 1
endin

instr 10
imax          =          1
idel1         =          p4
igain1        =          p5
idel2         =          p6
igain2        =          p7
idel3         =          p8
igain3        =          p9
idel4         =          p10
igain4        =          p11
idel5         =          p12
igain5        =          p13
idel6         =          p14
igain6        =          p15
afdbk         = init    0
aout1         = nestedap gasig+afdbk*.4, 3, imax, idel1, igain1, idel2, \\
                 igain2, idel3, igain3
aout2         = nestedap aout1, 2, imax, idel4, igain4, idel5, igain5
aout          = nestedap aout2, 1, imax, idel6, igain6
afdbk         = butterlp aout, 1000
gasig         = outs   gasig+(aout+aout1)/2, gasig-(aout+aout1)/2
endin

```

;Partitura

f1 0 8192 10 1

; Diskin

; Sta Dur Soundin

i5 0 3 1

; Reverb

; St	Dur	Del1	Gn1	Del2	Gn2	Del3	Gn3	Del4	Gn4	Del5	Gn5	Del6	Gn6
i10	0 4	97	.11	23	.07	43	.09	72	.2	53	.2	119	.3

e

55.3.5 AUTOR

Hans Mikelson

Febrero 1999

Nuevo en la versión 3.53

56 MODIFICADORES DE SEÑAL: GUÍAS DE ONDA

56.1 wguide1, wguide2

```
ar wguide1 asig, xfreq, xcutoff, kfeedback
ar wguide2 asig, xfreq1, xreq2, xcutoff1, xcutoff2, kfeedback1,kfeedback2
```

56.1.1 DESCRIPCIÓN

Bloques simples de guías de ondas.

56.1.2 EJECUCIÓN

asig – ruido de excitación de entrada.

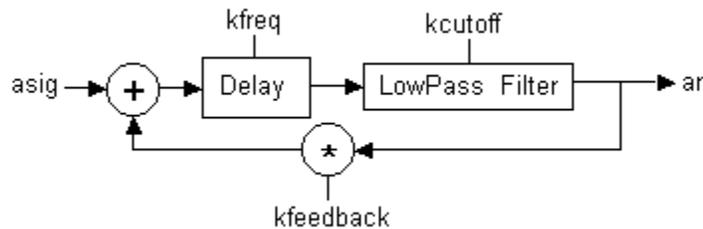
xfreq - frecuencia (es decir, la inversa del tiempo de retardo).

kcutoff - frecuencia de corte del filtro en Hz.

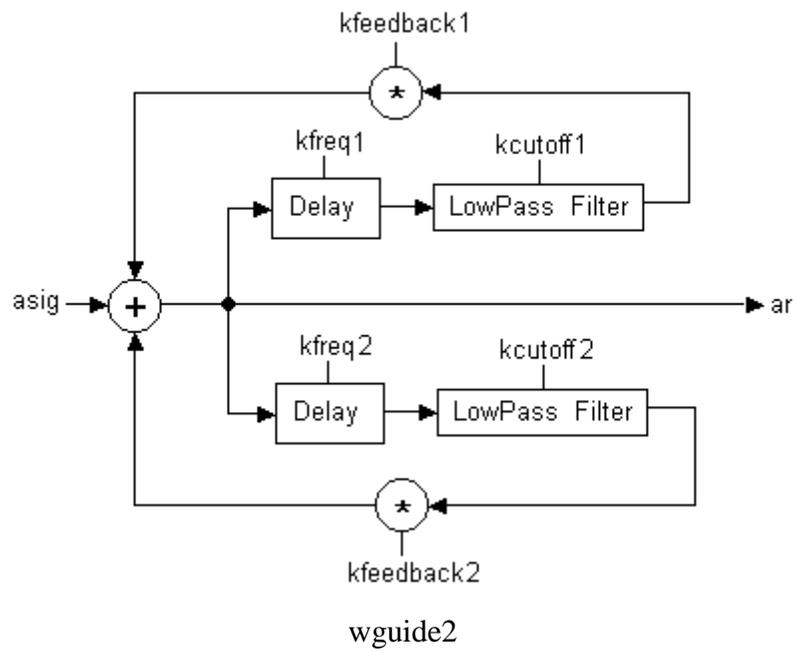
kfeedback - factor de retroalimentación.

wguide1 es el modelo de guía de onda más elemental, consistente en una línea de retardo y un filtro pasa-bajos de primer orden.

wguide2 es un modelo de una lámina golpeada, consistente en dos líneas paralelas de retardo y dos filtros pasa-bajos de primer orden. Las dos líneas de retroalimentación se mezclan y se envían a la línea de retardo una y otra vez en cada ciclo. Implementando algoritmos de guía de onda como opcodes, en vez de como instrumentos en la orquesta, se consigue que el usuario pueda especificar una **kr** distinta de la **sr**, permitiendo una mejor ejecución, particularmente en tiempo real.



wguide1



56.1.3 AUTOR

John ffitch
 Universidad de Bath, Codemist Ltd.
 Bath, Reino Unido
 1998

56.2 streson

ar **streson** asig, kfr, ifdbgain

Una señal de audio es modificada por un efecto de cuerda resonante, con una frecuencia fundamental variable.

56.2.1 INICIALIZACIÓN

ifdbgain - ganancia de retroalimentación, entre 0 y 1, de la línea de retardo interna. Un valor cercano a 1 crea una caída más lenta y una resonancia más pronunciada. Los valores pequeños pueden dejar a la señal inalterada. Dependiendo de la frecuencia del filtro, los valores típicos son mayores que 0.9.

56.2.2 EJECUCIÓN

streson pasa la señal de entrada *asig* a través de una red compuesta por filtros peine, pasa-altos y pasa-bajos, similar a la que se usa en algunas versiones de algoritmo de Karplus-Strong, creando el efecto de la resonancia de una cuerda. La frecuencia fundamental de la cuerda se controla por la variable de control *kfr*. Este opcode puede ser usado para simular resonancias simpáticas de una señal de entrada.

streson es una adaptación del objeto `StringFlt` de la Librería de Sonidos "SndObj" desarrollada por el autor.

56.2.3 AUTOR

Victor Lazzarini
Departamento de Música
Universidad Nacional de Irlanda, Maynooth
Maynooth, Co. Kildare
1998 (Nuevo en la versión 3.494)

57 MODIFICADORES DE SEÑAL: EFECTOS ESPECIALES

57.1 harmon

```
ar    harmon    asig, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, imode, \\
      iminfrq, iprd
```

57.1.1 DESCRIPCIÓN

Analiza un sonido de entrada y genera voces armonizantes en sincronía.

57.1.2 INICIALIZACIÓN

imode - modo de interpretar los valores de entrada *kgenfreq1*, *kgenfreq2* para la generación de las frecuencias armónicas. Un valor 0 indica que los valores de entrada deben ser tomados como cocientes (en Hz) respecto a la señal de audio analizada. Una valor 1 indica que los valores de entrada deben ser las frecuencias reales (absolutas) requeridas (en Hz).

iminfrq - es la frecuencia más baja esperada (en Hz) de la entrada de audio. Este parámetro determina la cantidad de la entrada que se guarda para el análisis, e indica un límite más bajo en el rastreador interno de alturas.

iprd - período del análisis (en segundos). Debido a que el análisis interno de alturas puede llevar mucho tiempo, la entrada es normalmente analizada sólo cada un cierto intervalo (de 20 a 50 milisegundos).

57.1.3 EJECUCIÓN

Esta unidad es un armonizador, capaz de proveer hasta 2 voces adicionales con la misma amplitud y espectro que la entrada. El análisis de la señal de entrada se basa en dos cosas: una frecuencia estimada de entrada *kestfrq* (en Hz) y un margen máximo fraccionario *kmaxvar* que sirve para delimitar la zona en la que se buscará dicha frecuencia estimada. Una vez que la frecuencia de entrada sea determinada, la forma del pulso más reciente se usa para generar las demás voces a la frecuencia deseada.

Las tres entradas de frecuencia pueden derivarse de un fichero de partitura o de un fichero MIDI. La primera de las tres es la altura esperada, con un parámetro que sirve de margen para permitir capturar inflexiones o pequeñas desafinaciones. Si la frecuencia esperada es 0, el armonizador no producirá otras voces. Los valores segundo y tercero controlan las frecuencias de salida. Si alguno de ellos es 0, el armonizador devolverá sólo la frecuencia distinta de 0. Si ambos son 0, el armonizador no producirá más voces. Cuando la frecuencia requerida es mayor que la de la señal de entrada, el proceso requiere tiempo adicional de computación, debido a los pulsos de salida que se pisan. Esto está actualmente limitado, por razones de eficiencia, a que sólo una voz pueda ser al mismo tiempo más alta que la entrada.

Esta unidad es útil para proporcionar efectos de coros de fondo o para corregir la altura de una voz de entrada defectuosa. Esencialmente, no hay retardo entre la señal de entrada y de salida. Esta incluye sólo las partes generadas (es decir, no incluye la voz original de entrada).

57.1.4 EJEMPLO

```
asig1      in                ; la entrada es una señal en directo
kcps1      cpsmidib          ; define la altura a buscar
asig2      harmon           asig1, kcps1, .3, kcps1, kcps1 * 1.25, 1, 110, .04 ; añade
una 3 mayor
          out                asig2                ; salida sólo con las voces añadidas
```

57.1.5 AUTOR

Barry Vercoe
M.I.T., Cambridge, Mass
1997

57.2 flanger

ar flanger asig, adel, kfeedback[, imaxd]

57.2.1 DESCRIPCIÓN

Aplica un flanger controlado por el usuario.

57.2.2 INICIALIZACIÓN

imaxd - retardo máximo en segundos (necesario para la asignación inicial de memoria).

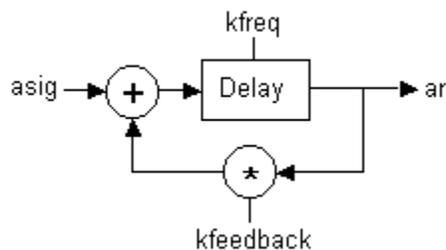
57.2.3 EJECUCIÓN

asig - señal de entrada.

adel - retardo en segundos.

kfeedback - cantidad de retroalimentación (en tareas normales no debería exceder un valor de 1, incluso si son válidos valores mayores).

Esta unidad es útil para generar coros y flangers. El retardo debe ser variado a una frecuencia de audio conectando la salida de un oscilador a *adel*. También la retroalimentación puede ser variada, a frecuencia de control. Este opcode se usa para permitir una **kr** diferente a la **sr** (de otra manera el retardo no podría ser menor que **ksmps**) lo que mejora la ejecución en tiempo real. Esta unidad es muy similar a **wguide1**, con la diferencia de que **flanger** no incorpora el filtro pasa-bajos.



57.2.4 AUTOR

Gabriel Maldonado

Italia

Octubre 1997

57.3 distort1

```
ar    distort1    asig[, kpregain[, kpostgain[, kshape1[, kshape2]]]]
```

57.3.1 DESCRIPCIÓN

Implementación de una distorsión de tangente hiperbólica modificada. **distort1** puede ser usado para generar distorsiones que modelen ondas sonoras, basadas en una modificación de la función de la tangente hiperbólica.

$$aout = \frac{\exp(asig * (pregain + shapel)) - \exp(asig*(pregain+shape2))}{\exp(asig*pregain) + \exp(-asig*pregain)}$$

57.3.2 EJECUCIÓN

asig – señal de entrada.

kpregain – determina la cantidad de ganancia aplicada a la señal antes del modelado de la onda. Un valor de 1 provoca una ligera distorsión.

kpostgain – determina la cantidad de ganancia aplicada a la señal después del modelado de la onda.

kshape1 – determina la forma de la parte positiva de la curva. Un valor de 0 proporciona un corte plano, mientras que valores positivos proporcionan curvas en pendiente.

kshape2 – determina la forma de la parte negativa de la curva.

57.3.3 EJEMPLO

```
gadist    init    0
instr 1
iamp      =      p4
ifqc      =      cspch(p5)
asig      pluck   iamp, ifqc, ifqc, 0, 1
gadist    =      gadist + asig
endin
instr 50
kpre      init    p4
kpost     init    p5
kshap1    init    p6
kshap2    init    p7
aout      distort1 gadist, kpre, kpost, kshap1, kshap2
outs      =      aout, aout
gadist    =      0
endin
```

```
;      Sta   Dur   Amp   Pitch
i1     0.0   3.0  10000 6.00
i1     0.5   2.5  10000 7.00
i1     1.0   2.0  10000 7.07
i1     1.5   1.5  10000 8.00

;      Sta   Dur   PreGain PostGain  Shape1 Shape2
i50    0     3     2         1         0         0
e
```

57.3.4 AUTOR

Hans Mikelson
Diciembre 1998
Nuevo en la Versión 3.50

57.4 phaser1, phaser2

ar	phaser1	asig, kfreq, iord, kfeedback[, iskip]
ar	phaser2	asig, kfreq, kq, iord, imode, ksep, kfeedback

57.4.1 DESCRIPCIÓN

Implementación de *iord* filtros pasa todo dispuestos en serie de primer orden (**phaser1**) y de segundo orden (**phaser2**).

57.4.2 INICIALIZACIÓN

iord – el número de etapas pasa todo en serie. Para **phaser1**, estas son filtros de primer orden, y *iord* puede estar en el rango de 1 a 4999. Para **phaser2**, estas son filtros de segundo orden, y *iord* puede estar en el rango de 1 a 2499. Con ordenes más altos, el tiempo de ejecución aumenta.

iskip – usado para controlar la disposición inicial del espacio de datos interno. Debido a que el filtrado incorpora un bucle de retroalimentación de la salida previa, el estado inicial del espacio de almacenamiento usado es significativo. Un valor 0, limpiará el espacio. Un valor distinto de 0 retendrá la información previa. El valor por defecto es 0.

imode – usado en el calculo de las frecuencias de corte.

57.4.3 EJECUCIÓN

kfreq – frecuencia (en Hz) del (de los) filtro(s). Para **phaser1** ésta es la frecuencia a la que cada filtro de la serie desplaza su entrada 90 grados. Para **phaser2**, es la frecuencia central del corte del primer filtro pasa todo de la serie. Esta frecuencia se usa como frecuencia base de la cual se derivarán todas las demás frecuencias de corte.

kq – Q de cada corte. Los valores altos de Q dan lugar a cortes más estrechos. Un valor de Q entre .5 y 1 da lugar al efecto de desplazamiento de fase más pronunciado, pero es posible usar valores incluso más altos para conseguir efectos especiales.

kfeedback – cantidad de la salida que se retroalimenta en la entrada de la cadena pasa todo. Con cantidades más grandes de retroalimentación, más cortes prominentes aparecen en el espectro de la salida. *Kfeedback* debe estar entre -1 y +1 para proporcionar estabilidad.

ksep –factor de escala usado, en conjunto con *imode*, para determinar las frecuencias de los cortes adicionales en el espectro de salida.

phaser1 implementa *iord* secciones pasa todo de primer orden, conectadas serialmente y compartiendo el mismo coeficiente. Cada sección pasa todo puede ser representada por la siguiente ecuación diferencial:

$$y(n) = C * x(n) + x(n-1) - C * y(n-1)$$


```

iamp      =      p4 * .05
iorder    =      p5                ; número de etapas de primer orden en
                                        ; la red de phaser1
                                        ; divide iorder por la mitad para conseguir
                                        ; el número de cortes
ifreq     =      p6                ; frecuencia de modulación de
                                        ; phaser1
ifeed     =      p7                ; cantidad de retroalimentación de phaser1
kamp      linseg 0, .2, iamp, idur - .2, iamp, .2, 0
iharms    =      (sr*.4) / 100
asig      gbuzz 1, 100, iharms, 1, .95, 2 ; onda diente de sierra
                                        ; oscilador de modulación para phaser1

kfreq     oscili 5500, ifreq, 1
kmod      =      kfreq + 5600
aphs      phaser1 asig, kmod, iorder, ifeed
out      (asig + apha) * iamp
endin

instr    2                ; demostración del desplazamiento de fase
                                        ; habilidades de phaser2. Entrada
                                        ; mezclada con la salida de phaser2 para
                                        ; generar cortes. Demuestra
                                        ; la interacción de imode y
                                        ; ksep.

idur      =      p3
iamp      =      p4 * .04
iorder    =      p5                ; número de etapas de segundo orden en
                                        ; la red de phaser2
                                        ; no se usa
ifreq     =      p6                ; cantidad de realimentación para phaser2
ifeed     =      p7                ; modo para la escala de frecuencia
imode     =      p8                ; usado con imode para determinar
isep      =      p9                ; las frecuencias de corte

kamp      linseg 0, .2, iamp, idur - .2, iamp, .2, 0
iharms    =      (sr*.4) / 100
asig      gbuzz 1, 100, iharms, 1, .95, 2 ; onda diente de sierra
                                        ; función de caída exponencial,
                                        ; para controlar las frecuencias de corte

kline     expseg 1, idur, .005
aphs      phaser2 asig, kline * 2000, .5, iorder, imode, isep, ifeed
out      (asig + apha) * iamp
endin

; partitura
f1 0 16384 9 .5 -1 0 ; medio seno invertido
; para modular la frecuencia de phaser1
f2 0 8192 9 1 1 .25 ; onda coseno para gbuzz
; phaser1
i1 0 5 7000 4 .2 .9
i1 6 5 7000 6 .2 .9
i1 12 5 7000 8 .2 .9
i1 18 5 7000 16 .2 .9
i1 24 5 7000 32 .2 .9
i1 30 5 7000 64 .2 .9

```

```
; phaser2, imode=1
    i2 37 10 7000 8 .2 .9 1 .33
    i2 48 10 7000 8 .2 .9 1 2
; phaser2, imode=2
    i2 60 10 7000 8 .2 .9 2 .33
    i2 72 10 7000 8 .2 .9 2 2
e
```

57.4.5 HISTORIAL TÉCNICO

Se puede encontrar una descripción general de las diferencias entre los efectos de un “flanger” y un “phaser” en Hartmann [1]. En Biegel [2] puede verse una implementación de filtros pasa todo de primer orden conectados en serie, donde la transformación z bilineal es usada para determinar la frecuencia del desplazamiento de fase de cada etapa. Cronin [3] presenta una implementación similar para una red de desplazamiento de fase de cuatro fases. Chamberlin [4] y Smith [5] tratan ambos de secciones pasa todo de segundo orden usadas para controlar más eficientemente la profundidad, la anchura y la frecuencia del corte,.

57.4.6 REFERENCIAS

1. Hartmann, W.M. “Flanging and Phasers.” *Journal of the Audio Engineering Society*, Vol. 26, No. 6, pp. 439-443, Junio 1978.
2. Beigel, Michael I. “A Digital ‘Phase Shifter’ for Musical Applications, Using the Bell Labs (Alles-Fischer) Digital Filter Module.” *Journal of the Audio Engineering Society*, Vol. 27, No. 9, pp. 673-676, Septiembre 1979.
3. Cronin, Dennis. “Examining Audio DSP Algorithms.” *Dr. Dobb's Journal*, Julio 1994, p. 78-83.
4. Chamberlin, Hal. *Musical Applications of Microprocessors*. Second edition. Indianapolis, Indiana: Hayden Books, 1985.
5. Smith, Julius O. “An Allpass Approach to Digital Phasing and Flanging.” *Proceedings of the 1984 ICMC*, p. 103-108.

57.4.7 AUTOR

Sean Costello
Seattle, Washington
1999
Nuevo en la Versión 4.0

58 MODIFICADORES DE SEÑAL: CONVOLUCIÓN Y MORPHING

58.1 convolve

```
ar1[,ar2][,ar3][,ar4] convolve ain, ifilcod, ichannel
```

58.1.1 DESCRIPCIÓN

La salida es la convolución de la señal *ain* con la respuesta al impulso contenida en *ifilcod*. Si se especifica más de una señal de salida, el proceso de convolución de cada una de ellas se efectuará con la misma respuesta al impulso. Observa que es mucho más eficiente usar este opcode cuando se procesa una entrada monoaural para crear una salida estéreo o cuadrafónica.

58.1.2 INICIALIZACIÓN

ifilcod - entero o cadena de caracteres que indica un fichero de datos de respuesta a un impulso. Un entero indica la extensión de un fichero *convolve.m*; una cadena de caracteres (entre comillas) proporciona un nombre de fichero, opcionalmente con su ruta completa. Si no se proporciona ésta, el fichero se buscará primero en el directorio actual, luego en el especificado por la variable de entorno SADIR (si está definida). Los datos del fichero contienen la transformada de Fourier de la respuesta a un impulso. El uso de la memoria depende del tamaño del fichero de datos, que es leído y mantenido en memoria completamente durante la ejecución, aunque compartido por varias llamadas.

58.1.3 EJECUCIÓN

convolve implementa un proceso de convolución rápida. La salida de este opcode es retrasada con respecto a la entrada. Se deben usar las siguientes fórmulas para calcular dicho retraso:

```
Para (1/kr) <= IRdur:  
  Retraso = ceil(IRdur * kr) / kr  
Para (1/kr) > IRdur:  
  Retraso = IRdur * ceil(1/(kr*IRdur))
```

donde:

kr = frecuencia de control de Csound

IRdur = duración, en segundos, de la respuesta al impulso

ceil(n) = entero más pequeño mayor que *n*

Se debe tener cuidado con el retardo inicial, si lo hay, de la respuesta al impulso.

Por ejemplo, si se crea la respuesta a un impulso desde una grabación, el fichero de sonido puede no incluir el retardo inicial. Así que debe asegurarse de que el fichero de sonido tenga la correcta cantidad de ceros al principio, o mejor, compensar este retraso en la orquesta (este último método es más eficiente). Para ello, se ha de restar el retardo inicial del resultado calculado usando las fórmulas anteriores.

Para las aplicaciones típicas, tal como reverbs, el retardo estará entre 0.5 y 1.5 segundos, o incluso más largo. Esto hace que la actual implementación no pueda ser usada en aplicaciones en tiempo real. Sin embargo, puede usarse para filtrado en tiempo real, si el número de capturas es suficientemente pequeño.

El autor tiene el intento de crear otro opcode de alto nivel que mezcle las señales con y sin modificación, usando la correcta cantidad de retardo automáticamente.

58.1.4 EJEMPLO

- Crea un fichero de respuesta a un impulso en el dominio de la frecuencia:

```
c:\ CSound -Ucvanal l1_44.wav l1_44.cv
```

- Determina la duración de la respuesta al impulso. Para mayor exactitud, determina el número frames de muestras del fichero de respuesta, y luego calcula la duración con:

```
duración = (no. de cuadros de muestra)/(frecuencia de muestreo del fichero de sonido)
```

Esto es debido al hecho de que la utilidad SNDINFO sólo proporciona la duración de los 10ms más próximos. Si tienes una herramienta que proporcione la duración con la requerida exactitud, usa dicho valor directamente.

```
c:\ sndinfo l1_44.wav
longitud = 60822 muestras, frecuencia de muestreo (hz) = 44100
Duración = 60822/44100 = 1.379s.
```

- Determina el retardo inicial, si hay, de la respuesta al impulso. Si no se le a quitado a ésta el retardo inicial, puedes saltarte este paso. Si ha sido eliminado, entonces la única manera que tienes de conocer el retardo inicial es si esa información se proporciona por separado. Para este ejemplo, asumamos que el retardo inicial es 60 ms.

- Determina el retardo requerido a ser aplicado a la señal original, para alinearla con la señal a la que se aplica el proceso de convolución.

```
Si kr = 441:
  1/kr = 0.0023, que es <= IRdur (1.379s), entonces:
    Retardo1 = ceil(IRdur * kr) / kr
              = ceil(608.14) / 441
              = 609/441
              = 1.38s
Informando sobre el retardo inicial:
  Retardo2 = 0.06s
Retardo total = retardo1 - retardo2
              = 1.38 - 0.06
              = 1.32s
```

- Crear fichero .orc, por ejemplo:

```
; Simple demostración del operador convolve para aplizar una reverb
sr          =      44100
kr          =      441
ksmps       =      100
nchnls      =       2

instr 1
imix = 0.22 ; mezcla de las señales con y sin modificación. Variar a voluntad
; NB: reverberaciones pequeñas a menudo requieren un mayor
; porcentaje de señal modificada para que suenen interesantes.
; reverberaciones grandes parecen requerir menos. Experimenta! La mezcla de señales
; modificada y no modificada es muy importante. Un cambio pequeño puede suponer
; una gran diferencia.
ivol = 0.9 ; Nivel de volumen total de la reverb. Puede necesitar algún ajuste
; cuando la mezcla varía, para evitar cortes.
idel = 1.32 ; retardo requerido para alinear la señal no modificada
; con la modificada por convolve
; Puede ser calculado dentro de la orquesta
; si se desea
adry      soundin    "anechoic.wav"      ; audio de entrada (sin modificar)
awet1,awet2 convolve  adry,"l1_44.cv"     ; audio stereo modificado por convolve
adrydel   delay     (1-imix)*adry,idel ; retrasa la señal sin modificar
; para alinearla con la señal modificada por convolve. Aplica ajustes
; de nivel aquí también
          outs       ivol*(adrydel+imix*awet1),ivol*(adrydel+imix*awet2)
; mezcla las señales modificada y no modificada y reproduce la salida.
endin
```

58.1.5 AUTOR

Greg Sullivan
1996

58.2 cross2

```
ar cross2 ain1, ain2, isize, ioverlap, iwin, kbias
```

58.2.1 DESCRIPCIÓN

Síntesis cruzada con FFT (Transformada rápida de Fourier).

58.2.2 INICIALIZACIÓN

isize - tamaño de la FFT a ejecutar. Cuanto mayor sea el tamaño, mejor será la respuesta a la frecuencia, aunque la respuesta temporal es un poco peor.

ioverlap - factor de solapamiento del algoritmo FFT. Debe ser una potencia de 2. Los valores que dan mejores resultados son 2 y 4. Un factor de solapamiento grande tarda más en compilarse.

iwin - tabla de función que contiene la ventana que se va a usar en el análisis.

58.2.3 EJECUCIÓN

ain1 - sonido fuente. Para mejores resultados conviene que tenga muchas frecuencias altas.

ain2 - sonido modulador. Debe tener una respuesta a la frecuencia variable (como en la voz) para conseguir buenos resultados.

kbias - cantidad del efecto de síntesis cruzada. 1 es lo normal, 0 la total ausencia de síntesis cruzada.

58.2.4 EJEMPLOS

```
a1  oscil      10000, 1, 1
a2  rand       10000
a3  cross2    a2, a1, 2048, 4, 2, 1
out out      a3
```

Si la tabla de función es un sonido vocal, esto producirá un ruido blanco "parlante". La tabla de función2 debe ser la función de la ventana (**GEN20**).

58.2.5 AUTOR

Paris Smaragdis
MIT, Cambridge
1997

59 MODIFICADORES DE SEÑAL: PANORAMIZACIÓN Y ESPACIALIZACIÓN

59.1 pan

`a1, a2, a3, a4 pan asig, kx, ky, ifn[, imode[, ioffset]]`

59.1.1 DESCRIPCIÓN

Distribuye una señal de audio entre cuatro canales, con control sobre la localización exacta.

59.1.2 INICIALIZACIÓN

ifn - número de la tabla de función de un patrón que describe el aumento de amplitud en el canal de un altavoz al acercársele el sonido de un altavoz adyacente. Requiere un elemento límite añadido.

imode (opcional) - modo de los valores de posición *kx* y *ky*. 0 significa modo de indexación normal, 1 indica que las entradas se normalizan en el intervalo de 0 a 1. El valor por defecto es 0.

ioffset (opcional) - indicador de offset para *kx* y *ky*. Un valor 0 deduce que el origen es el canal 3 (altavoz trasero izquierdo); un valor 1 pide un desplazamiento del eje hacia centro cuadrafónico. El valor por defecto es 0.

59.1.3 EJECUCIÓN

pan toma la señal de entrada *asig* y la distribuye por los cuatro canales de salida (esencialmente altavoces cuadrafónicos) de acuerdo con el valor de los controles *kx* y *ky*. Para entradas normalizadas (modo 1) y sin offset, las cuatro posiciones de salida van en el siguiente orden: altavoz izquierdo delantero en (0,1), derecho delantero en (1,1), izquierdo trasero en el origen (0,0) y derecho trasero en (1,0). En la notación (*kx*, *ky*), las coordenadas *kx* y *ky*, cada una de ellas en el rango de 0 a 1, controlan de esta manera la localización del sonido en cualquier dirección "izquierda-derecha, adelante-atrás".

El movimiento de la señal entre altavoces se efectúa por la variación de amplitud que le llega a cada uno de ellos, controlada por la tabla de función almacenada *ifn*. Al ir *kx* desde 0 a hasta 1, la fuerza de las señales a mano derecha crecerán desde el valor más a la izquierda de la tabla hasta el valor más a la derecha, mientras que la fuerza de las señales a mano izquierda irán desde el valor más a la derecha de la tabla hasta el valor más a la izquierda. Para una panoramización lineal simple, la tabla debe contener una función lineal de 0 a 1. Una panoramización más correcta que mantenga la potencia constante podría obtenerse guardando en la tabla el primer cuadrante de una función sinusoidal. Debido a que el proceso de panoramización escalará y truncará los valores de *kx* y *ky* al realizar su búsqueda en la tabla, se debería usar una tabla medianamente grande (de por ejemplo 8193).

Los valores k_x y k_y no están restringidos al rango de 0 a 1. Un movimiento circular pasando a través de los cuatro altavoces debería tener un diámetro igual a "raíz de 2", y puede ser definido por un círculo de radio "R= raíz de 1/2", con centro en (.5, .5) (es decir, offset =1). Los valores no escalados operan de manera análoga. Los sonidos pueden ser así posicionados en cualquier lugar del plano polar o cartesiano. Los puntos que caigan fuera del cuadrado formado por los altavoces serán proyectados correctamente sobre el perímetro del cuadrado para un oyente situado justo en el centro.

59.1.4 EJEMPLO

```
instr 1
k1      phasor 1/p3          ; fracción de un círculo
k2      tablei k1, 1, 1     ; seno del ángulo (sinusoide en f1)
k3      tablei k1, 1, 1, .25, 1 ; coseno del ángulo(seno del offset
1/4 círculo)
a1      oscili 10000,440, 1  ; señal de audio...
a1,a2,a3,a4  pan a1, k2/2, k3/2, 2, 1, 1
                                ; enviada en un círculo
                                ;(f2=primer cuadrante de
                                ; la función seno)

outq    a1, a2, a3, a4

endin
```

59.2 locsig, locsend

a1, a2	locsig	asig, kdegree, kdistance, kreverbsend
a1, a2, a3, a4	locsig	asig, kdegree, kdistance, kreverbsend
a1, a2	locsend	
a1, a2, a3, a4	locsend	

59.2.1 DESCRIPCIÓN

locsig toma una señal de entrada y la distribuye entre 2 o 4 canales, usando valores en grados para calcular el balance entre canales adyacentes. También toma un parámetro para la distancia (usado para atenuar señales que deberían sonar como si estuvieran más lejos que el altavoz mismo), y otro para la cantidad de señal que se enviará a los reverberadores. Esta unidad esta basada en el ejemplo del libro Computer Music, página 320, de Charles Dodge/Thomas Jerse.

locsend depende de la existencia de una unidad **locsig** previamente definida. El número de señales de salida debe coincidir con el número de la unidad **locsig** anterior. Las señales de salida de **locsend** se derivan de los valores dados para la distancia y la reverberación en **locsig** y están listos para ser enviados a las unidades de reverberación locales o globales (ver el ejemplo más abajo). La cantidad de reverberación y balance entre los 2 o 4 canales se calcula de la misma manera que se describe en el libro mencionado (un libro esencial!).

59.2.2 EJECUCIÓN

kdegree - valor entre 0 y 360 para la localización de la señal en un espacio de 2 o 4 canales configurados como: a1=0, a2=90, a3=180, a4=270 (por ejemplo *kdegree* = 45 balancearía la señal equitativamente entre a1 y a2). **locsig** pasa *kdegree* a funciones seno y coseno para calcular el balance de la señal (por ejemplo: asig=1, kdegree=45, a1=a2=.707).

kdistance - valor (mayor o igual que 1) usado para atenuar la señal y calcular el nivel de reverberación con el fin de simular las indicaciones de distancia. Al hacer *kdistance* mayor, el sonido se hará más suave y de alguna manera más reverberante (asumiendo que se use **locsend** en este caso).

kreverbsend - porcentaje de la señal directa que será procesada junto con los valores de distancia y localización para obtener la cantidad de señal que será enviada a la unidad de reverberación, por ejemplo **reverb** o **reverb2**.

59.2.3 EJEMPLO

```
;asig = alguna señal de audio
kdegree      line      0, p3, 360
kdistance    line      1, p3, 10
a1, a2, a3, a4  locsig  asig, kdegree, kdistance, .1
ar1, ar2, ar3, ar4  locsend
ga1          =          ga1+ar1
ga2          =          ga2+ar2
ga3          =          ga3+ar3
ga4          =          ga4+ar4
              outq      a1, a2, a3, a4
endin
```

```

instr 99                                     ; instrumento reverb
a1      reverb2      ga1, 2.5, .5
a2      reverb2      ga2, 2.5, .5
a3      reverb2      ga3, 2.5, .5
a4      reverb2      ga4, 2.5, .5
        outq         a1, a2, a3, a4
ga1     =            0
ga2     =            0
ga3     =            0
ga4     =            0

```

En este ejemplo la señal *asig* se envía alrededor de un círculo completo una vez durante la duración de una nota, mientras que al mismo tiempo se aleja más y más de la posición del oyente.

locsig envía la cantidad apropiada de señal a **locsend** internamente. Las salidas de **locsend** se suman a los acumuladores globales de Csound y las señales globales se usan como entradas de las unidades de reverberación en un instrumento separado.

locsig es útil para la panoramización de sonido en estéreo o cuadrafonía, así como para colocar sonidos en un punto fijo entre altavoces. Abajo hay un ejemplo de la localización fija de sonidos en un campo estéreo.

```

instr 1
a1, a2   locsig      asig, p4, p5, .1
ar1, ar2 locsend
ga1      =          ga1+ar1
ga2      =          ga2+ar2
        outs        a1, a2

endin
instr 99                                     ; reverb....
endin

```

Unas notas:

```

; coloca el sonido en el altavoz izquierdo y cerca
il 0 1 0 1
; coloca el sonido en el altavoz derecho y lejos
il 1 1 90 25
; coloca el sonido en el medio de los altavoces y a media distancia
il 2 1 45 12
e

```

El siguiente ejemplo muestra un uso simple e intuitivo del valor de la distancia para simular el efecto Doppler. El mismo valor que se usa para escalar la frecuencia es el que se usa como parámetro de distancia en **locsig**.

```

kdistance   line      1, p3, 10
kfreq       =        (ifreq * 340) / (340 + kdistance)
asig        oscili    iamp, kfreq, 1
kdegree     line      0, p3, 360
a1, a2, a3, a4 locsig  asig, kdegree, kdistance, .1
ar1, ar2, ar3, ar4 locsend

```

59.2.4 AUTOR

Richard Karpen

Seattle, Wash

1998 (Nuevo en la versión 3.48)

59.3 space, spsend, spdist

a1, a2, a3, a4	space	asig, ifn, ktime, kreverbsend [,kx, ky]
a1, a2, a3, a4	spsend	
k1	spdist	ifn, ktime, [,kx, ky]

59.3.1 DESCRIPCIÓN

space toma una señal de entrada y la distribuye entre 4 canales usando coordenadas cartesianas xy para calcular el balance de las salidas. Las coordenadas xy pueden ser definidas en un fichero de texto separado y accedidas a través de una función de la partitura usando **GEN28**, o pueden ser especificadas usando los argumentos opcionales kx y ky . Hay algunas ventajas en usar el primer método:

1. se puede usar una interfaz gráfica para dibujar y editar trayectorias en el plano cartesiano;
2. el formato del fichero está en la forma "tiempo1 X1 Y1 tiempo2 X2 Y2 tiempo3 X3 Y3", permitiendo al usuario definir una trayectoria propia en el tiempo.

space permite al usuario especificar un puntero temporal (como los usados en **pvoc**, **lpread** y otras unidades) para conseguir un control preciso sobre la velocidad final del movimiento.

spsend depende de la existencia de una unidad **space** previamente definida. Las señales de salida de **spsend** se derivan de los valores dados a XY y a reverb en la unidad **space** y están listas para ser enviadas a una unidad local o global de reverberación (ver el ejemplo de abajo).

spdist usa los mismos datos xy que **space**, también desde un fichero de texto usando **GEN28** o directamente desde los argumentos kx y ky de la unidad. El propósito de esta unidad es hacer disponible los valores de distancia que se calculan a partir de las coordenadas xy . En el caso de **space**, los valores xy se usan para determinar una distancia que es usada para atenuar la señal y prepararla para su uso con **spsend**. Pero es también útil para tener esos valores de distancia disponibles para escalar la frecuencia de la señal antes de enviarla a la unidad **space**.

59.3.2 EJECUCIÓN

La configuración de las coordenadas xy en **space** coloca la señal en los altavoces (an) de la siguiente manera: a1 es (-1,1); a2 es (1, 1); a3 es (-1, -1); a4 es (1, -1). Esto asume que un altavoz configurado como a1 está delante a la izquierda, a2 está delante en la derecha, a3 atrás a la izquierda y a4 está atrás a la derecha. Los valores mayores que 1 darán lugar a sonidos atenuados como si estuvieran a distancia. **space** considera que los altavoces están a una distancia 1. Pueden usarse valores más pequeños que 1 para coordenadas xy , pero **space** no amplificará la señal en ese caso. Sin embargo, balanceará la señal para que pueda sonar como si estuviera entre el espacio de los cuatro altavoces. $x=0, y=1$, colocarán la señal igualmente balanceada entre los canales delanteros izquierdo y derecho; $x=y=0$ colocarán la señal en medio de los cuatro altavoces, etc. Aunque debe haber 4 señales de salida en **space**, éste puede ser usado en una orquesta con sólo 2 canales. Si las coordenadas se mantienen para que y sea siempre mayor o igual que 1, se puede mover o fijar sonidos en un campo estéreo sin problemas.

ifn - número de la función almacenada creada con **GEN28**. Este generador de función lee un fichero de texto que contiene las series de tres valores que representan las coordenadas xy y una etiqueta de tiempo

que indica el instante en el que será colocada la señal en esa posición en el espacio. El fichero debe ser de esta forma:

```
0 -1 1
1 1 1
2 4 4
2.1 -4 -4
3 10 -10
5 -40 0
```

si este fichero se llamara "move" entonces la llamada de **Gen28** en la partitura sería;

```
f1 0 0 "move"
```

GEN28 toma un valor de 0 como tamaño y asigna automáticamente la memoria necesaria, creando valores de hasta 10 milisegundos de resolución. Así que en este caso habrá 500 valores, creados por interpolación, de x_1 a x_2 , de x_2 a x_3 etc. (lo mismo con las ordenadas), a partir de los valores que proporciona la tabla de función. En el ejemplo de arriba, el sonido empezará en la parte delantera izquierda, durante un segundo se moverá la parte delantera derecha, durante otro segundo se alejará pero todavía en la parte delantera izquierda y luego, en sólo una décima de segundo, se moverá a la parte trasera izquierda, sonando un poco a distancia. Finalmente durante los últimos .9 segundos el sonido se moverá a la parte trasera derecha, a media distancia, viniendo por fin a descansar entre los canales izquierdos, bastante alejado. Debido a que los valores de la tabla son accedidos a través del uso del puntero temporal de la unidad **space**, el tiempo real puede ser obtenido siguiendo los tiempos que indica exactamente el fichero. También puede ser modificado si hacemos que el puntero se mueva más lento o más rápido a lo largo de la misma trayectoria (es decir, cambiando la velocidad del movimiento pero no la trayectoria). Si tienes acceso a una interfaz gráfica que te permita dibujar y editar los ficheros, no hay necesidad de teclear manualmente los valores en un fichero de texto. Pero no importa como se realice el fichero, servirá siempre que se ajuste al formato especificado arriba.

IMPORTANTE: si *ifn* es 0, **space** tomará los valores de las coordenadas xy de los parámetros kx y ky .

vertime - índice de la tabla que contiene las coordenadas xy . Si se usa como:

```
vertime          line          0, 5, 5
a1, a2, a3, a4   space        asig, 1, vtime, ...
```

con el fichero "move" descrito arriba, la velocidad del movimiento de la señal será exactamente la especificada en el fichero. Sin embargo, en:

```
vertime line 0, 10, 5
```

la señal se moverá a la mitad de la velocidad especificada. O, en el caso de:

```
vertime line 5, 15, 0
```

la señal se moverá en dirección contraria a la especificada y tres veces más lento. Finalmente:

```
vertime line 2, 10, 3
```

hará que la señal se mueva sólo desde el lugar especificado en la línea 3 del fichero de texto hasta el lugar especificado en la línea 5, y el proceso llevará 10 segundos.

*kreverb*send - es el porcentaje de la señal directa que será procesado junto con los valores de distancia y de coordenada para calcular la cantidad de señal que será enviada a las unidades de reverberación como **reverb**, o **reverb2**

kx, *ky* - cuando *ifn* es 0, **space** y **spdist** usarán estos valores como coordenadas xy para posicionar la señal. Son opcionales y son ambos 0 por defecto.

59.3.3 EJEMPLO

```

;asig es alguna señal de audio
ktime          line 0, p3, p10
a1, a2, a3, a4  space asig,1, ktime, .1
ar1, ar2, ar3, ar4  spsend
ga1            =    ga1+ar1
ga2            =    ga2+ar2
ga3            =    ga3+ar3
ga4            =    ga4+ar4
outq a1, a2, a3, a4
endin
instr 99                ; instrumento reverb
a1  reverb2 ga1, 2.5, .5
a2  reverb2 ga2, 2.5, .5
a3  reverb2 ga3, 2.5, .5
a4  reverb2 ga4, 2.5, .5
outq a1, a2, a3, a4
ga1 = 0
ga2 = 0
ga3 = 0
ga4 = 0
```

En el ejemplo de arriba, la señal *asig* es desplazada según los datos de la función 1 indexada por *ktime*. **space** envía la cantidad apropiada de señal a **spsend** directamente. Las salidas de **spsend** se suman a los acumuladores globales de Csound y las señales globales se usan como entrada a las unidades de reverberación en un instrumento separado.

space puede ser útil para la panoramización estéreo o cuadrafónica, así como para fijar sonidos en cualquier lugar entre los altavoces. Abajo hay un ejemplo de la localización fija de sonidos en un campo estéreo, usando los valores xy de la partitura en lugar de una tabla de función.

```

instr 1
...
a1, a2, a3, a4  space      asig, 0, 0, .1, p4, p5
ar1, ar2, ar3, ar4  spsend
ga1  =            ga1+ar1
ga2  =            ga2+ar2
                outs a1, a2
endin
instr 99 ; reverb...
....
endin
```

Unas cuantas notas: p4 y p5 son las coordenadas x e y:

```
; coloca el sonido a la izquierda y delante
  il 0 1 -1 1
; coloca el sonido a la derecha y lejos
  il 1 1 45 45
; coloca el sonido entre los altavoces a una distancia media.
  il 2 1 0 12
e
```

El siguiente ejemplo muestra un uso intuitivo y simple de los valores de distancia devueltos por **spdist** para simular el efecto Doppler:

```

ktime          line      0, p3, 10
kdist          spdist   1, ktime
kfreq         =         (ifreq * 340) / (340 + kdist)
asig          oscili   iamp, kfreq, 1
a1, a2, a3, a4 space   asig, 1, ktime, .1
ar1, ar2, ar3, ar4 spsend
```

La misma función y valores de tiempo se usan para **spdist** y **space**. Esto asegura que los valores de distancia usados internamente en la unidad **space** serán los mismos devueltos por **spdist** para dar la impresión del efecto Doppler.

59.3.4 AUTOR

Richard Karpen
Seattle, Wash
1998 (Nuevo en la versión 3.48)

59.4 hrtfer

```
aLeft, aRight    hrtfer    asig, kAz, kElev, "HRTFcompact"
```

59.4.1 DESCRIPCIÓN

La salida es audio tridimensional estéreo.

59.4.2 INICIALIZACIÓN

kAz - valor azimutal en grados. Los valores positivos representan las posiciones a la derecha, los negativos a la izquierda.

kElev - valor de elevación en grados. Los valores positivos representan posiciones por encima de la horizontal, los negativos por debajo.

En el presente, el único fichero que puede ser usado con **hrtfer** es HRTFcompact. Debe ser pasado al opcode como último argumento entre comillas como se muestra arriba.

HRTFcompact puede ser obtenido en:

```
ftp://ftp.maths.bath.ac.uk/pub/dream/utilities/Analysis/HRTFcompact
```

59.4.3 EJECUCIÓN

Estos opcodes colocan una señal mono en un espacio virtual tridimensional alrededor del oyente, realizando un proceso de convolución de la entrada con los datos HRTF apropiados, especificados por los valores de azimut y de elevación del opcode. **hrtfer** permite que estos valores sean de tipo -k, lo que permite una espacialización dinámica variable a frecuencia de control. **hrtfer** sólo puede colocar su entrada en una posición requerida, porque el HRTF se carga en la inicialización (recuerda que actualmente Csound tiene un límite de 20 ficheros en memoria, de otra manera causa un error de segmentación). La salida necesitará ser escalada bien con **balance**, bien multiplicándola por alguna constante.

Nota - la frecuencia de muestreo de la orquesta debe ser 44.1 kHz. Esto es así porque los HRTFs fueron medidos a dicha frecuencia de muestreo. Para ser usados a una frecuencia diferente, los HRTFs deben ser resampleados a la frecuencia deseada.

59.4.4 EJEMPLO

```
kaz          linseg    0, p3, -360           ; mueve el sonido en círculo
kel          linseg   -40, p3, 45           ; alrededor del oyente, cambiando
                                           ; la elevación en su giro

asrc         soundin  "soundin.1"
aleft,aright hrtfer  asrc, kaz, kel, "HRTFcompact"
aleftscale  =         aleft * 200
arightscale =         aright * 200
outs       aleftscale, arightscale
```

59.4.5 AUTORES

Eli Breder & David MacIntyre
Montreal
1996

60 MODIFICADORES DE SEÑAL: OPERADORES A NIVEL DE MUESTRA

60.1 samphold, downsamp, upsamp, interp, integ, diff

kr	downsamp	asig[, iwlen]
ar	upsamp	ksig
ar	interp	ksig[, iskip]
kr	integ	ksig[, iskip]
ar	integ	asig[, iskip]
kr	diff	ksig[, iskip]
ar	diff	asig[, iskip]
kr	samphold	xsig, kgate[, ival, ivstor]
ar	samphold	asig, xgate[, ival, ivstor]

60.1.1 DESCRIPCIÓN

Modifica una señal utilizando técnicas de submuestreo o sobremuestro, integración y diferenciación.

60.1.2 INICIALIZACIÓN

iwlen (opcional) - longitud, en muestras, de la ventana sobre la cual se promedia la señal de audio para obtener un valor submuestreado. La longitud máxima es *ksmps*; 0 y 1 implican que no hay proceso de promedio. El valor por defecto es 0.

iskip (opcional) - disposición inicial del espacio interno reservado (ver **reson**). El valor por defecto es 0.

ival, *ivstor* (opcional) - controla la disposición inicial del espacio interno reservado. Si *ivstor* es 0, el valor "mantenido" interno se iguala al valor de *ival*, en cualquier otro caso retiene su valor previo. Los valores por defecto son 0 y 0 (es decir, inicializado a 0)

60.1.3 EJECUCIÓN

downsamp submuestra una señal de audio convirtiéndola en una señal de control. Produce un valor de control *kval* por cada ciclo de control de audio. La ventana opcional invoca un proceso de promedio para evitar el *foldover* (muestras fuera de rango).

upsamp e **interp** convierten una señal de control en una señal de audio. El primer opcode lo realiza mediante la simple repetición de un valor de control *kval*, mientras que el segundo usa un proceso de interpolación lineal entre valores de control (*kvals*) sucesivos. **upsamp** es sensiblemente más eficiente que la asignación "asig=ksig".

integ y **diff** ejecutan procesos de integración y diferenciación de una señal de control o audio de entrada. Cada opcode es el opuesto del otro, y aplicando ambos sucesivamente se reconstruye la señal original. Ya que estas unidades son casos especiales de filtros pasa-bajos y pasa-altos, producen una salida escalada (y con desplazamiento de fase) que es dependiente de la frecuencia. Así, **diff** de una función seno produce

una función coseno con una amplitud " $2 * \text{seno}(\pi * \text{cps} / \text{sr})$ " con respecto a la original (para cada parcial); **integ** afectará de manera inversa a las magnitudes de sus componentes de entrada. Con esta comprensión, estas unidades pueden proporcionar muchas ideas útiles para modificar la señal.

samphold ejecuta un proceso de "congelación de muestra" en su señal de entrada, de acuerdo con el valor de *gate*. Si *gate* = 0, el último valor de salida se repite. El valor de control *gate* puede ser una constante, una señal de control o incluso una de audio.

60.1.4 EJEMPLO

```
asrc  buzz          10000,440,20, 1      ; tren de ondas pulso de banda limitada
adif  diff         asrc                ; enfatiza los altos
anew  balance     adif, asrc          ; pero retiene la potencia
agate reson       asrc,0,440          ; usa un filtro pasa bajos del original
asamp samphold    anew, agate         ; como entrada de la nueva señal de audio
aout  tone        asamp,100          ; suaviza los bordes asperos.
```

60.2 ntrpol

```
ir    ntrpol    isig1, isig2, ipoint [, imin, imax]
kr    ntrpol    ksig1, ksig2, kpoint [, imin, imax]
ar    ntrpol    asig1, asig2, kpoint [, imin, imax]
```

60.2.1 DESCRIPCIÓN

Calcula la media ponderada (esto es, la interpolación lineal) de dos señales de entrada.

60.2.2 INICIALIZACIÓN

imin - valor mínimo de *xpoint* (opcional, valor por defecto 0).

imax - valor superior de *xpoint* (opcional, valor por defecto 1).

60.2.3 EJECUCIÓN

xsig1, *xsig2* - señales de entrada.

xpoint - punto de interpolación entre los dos valores.

ntrpol calcula la interpolación lineal entre dos valores de entrada. *xpoint* es la distancia, desde el primer valor, del punto de evaluación. Con los valores por defecto de *imin* y *imax* (0 y 1) un valor de 0 indica que no hay separación desde primer valor, y que la distancia desde el segundo es máxima. Con un valor de 0.5, **ntrpol** devolverá el valor medio de los dos valores, indicando el punto medio exacto entre las señales *xsig* y *xsig2*. Un valor 1 indica que hay una distancia máxima desde el primer valor, y que no existe separación desde el segundo. El rango de *xpoint* puede ser definido también con *imin* y *imax* para hacer su uso más fácil.

Estos opcodes son útiles para interpolar los valores de dos señales.

60.3 fold

```
ar    fold  asig, kincr
```

60.3.1 DESCRIPCIÓN

Añade un efecto de *foldover* artificial a una señal de audio.

60.3.2 EJECUCIÓN

asig - señal de entrada.

kincr - cantidad de *foldover* expresada como un múltiplo de la frecuencia de sampleo. Debe ser ≥ 1 .

fold es un opcode que crea un efecto de *foldover* artificial. Por ejemplo, cuando *kincr* es igual a 1, con $sr=44100$, no se añade *foldover*. Cuando *kincr* es puesto a 2, el *foldover* será equivalente a mubmuestrear la señal a 22050 Hz, cuando es puesto a 4, a 11025 Hz, etc. Se pueden usar valores fraccionarios de *kincr*, lo que permite una variación continua de la cantidad de *foldover*. Esto puede ser usado para un amplio espectro de efectos especiales.

60.3.3 EJEMPLO

```
instr 1
kfreq line 1,p3,200
a1  oscili 10000, 100, 1
k1  init 8.5
a1  fold a1, kfreq
    out a1
endin
```

60.3.4 AUTOR

Gabriel Maldonado

Italia

1999

Nuevo en la versión 3.56

61 EL SISTEMA DE CABLEADO ZAK

Los opcodes zak se usan para crear un sistema para "conectar" señales de tipo i-, k- y a-. El sistema zak puede verse como un array global de variables. Estos opcodes son útiles para llevar a cabo conexiones flexibles de un instrumento a otro. El sistema es similar a la matriz de conexión de entradas-salidas en una mesa de mezclas o a una matriz de modulación en un sintetizador. Es útil también para cuando se necesita un array de variables.

El sistema zak es inicializado por el opcode **zakinit**, que es colocado normalmente justo después de las demás inicializaciones: **sr**, **kr**, **ksmps**, **nchnls**. El opcode **zakinit** define dos áreas de memoria, una para conexiones de señales de inicialización (i-) y de control (k-) y otra para la conexión de señales de audio (a-). El opcode **zakinit** sólo puede ser llamado una vez. Una vez que el espacio de zak es inicializado, se pueden usar los demás opcodes zak para leer y escribir en el espacio de memoria de zak, así como ejecutar otras varias tareas.

61.1 zakinit

`zakinit` `isizea, isizek`

61.1.1 DESCRIPCIÓN

Establece el espacio zak. Debe ser llamado sólo una vez.

61.1.2 INICIALIZACIÓN

isizea - número de posiciones para las variables de tipo a- para las conexiones de audio. Cada posición es en realidad un array de longitud **ksmps**.

isizek - número de posiciones reservadas para los floats (valores en coma flotante) en el espacio zk. Estos floats pueden ser escritos y leídos a frecuencias de inicialización (i-) o control (k-).

61.1.3 EJECUCIÓN

Se asigna siempre al menos una posición tanto para el espacio za como para el espacio zk. Puede haber miles o decenas de miles de posiciones para los espacios za y zk pero la mayoría de las piezas probablemente sólo necesitarán unas pocas docenas para la conexión de sus señales. Dichas posiciones de conexión son referidas por su número en los demás opcodes del sistema zak.

Para ejecutar **zakinit** sólo una vez, se coloca fuera de la definición de cualquier instrumento, en la cabecera del fichero orquesta, después de **sr**, **kr**, **ksmps**, y **nchnls**.

61.1.4 EJEMPLO

```
zakinit 10 30
```

reserva memoria para las posiciones desde la 0 hasta la 30 en el espacio zk y para las posiciones desde la 0 hasta la 10 en el espacio za. Con **ksmps**=8, por ejemplo, esto tomaría 31 floats para zk y 80 floats para za.

61.1.5 AUTOR

Robin Whittle
Australia
Mayo 1997

61.2 **ziw, zkw, zaw, ziwm, zkwm, zawm**

```
ziw   isig, indx
zkw   ksig, kndx
zaw   asig, kndx
ziwm  isig, indx [, imix]
zkwm  ksig, kndx [, imix]
zawm  asig, kndx [, imix]
```

61.2.1 DESCRIPCIÓN

Escribe en una posición del espacio zk, tanto a frecuencia de inicialización como de control, o en una posición del espacio za a frecuencia de audio. La escritura puede realizarse con o sin mezcla.

61.2.2 INICIALIZACIÓN

indx –apunta a la posición en el espacio zk en la que escribir.

isig – inicializa el valor de la posición del espacio zk.

61.2.3 EJECUCIÓN

kndx –apunta a la posición en el espacio zk o za en la que escribir.

ksig – valor a escribir en la posición del espacio zk.

asig – valor a escribir en la posición del espacio za.

ziw escribe *isig* en la variable zk especificada por *indx*.

zkw escribe *ksig* en la variable zk especificada por *kndx*.

zaw escribe *asig* en la variable za especificada por *kndx*.

Estos opcodes son rápidos, y comprueban siempre que el índice está dentro del rango del espacio zk o za. Si no lo está, se devuelve un mensaje de error junto con un valor 0, y no tiene lugar ninguna operación de escritura.

ziwm, **zkwm**, y **zawm** son opcodes de mezcla, es decir, añaden la señal al valor actual de la variable. Si no se especifica *imix*, el proceso de mezcla tiene lugar por defecto, pero si *imix* es especificado previamente, un valor *imix*=0 sobrescribirá los valores tal y como lo hacen **ziw**, **zkw**, y **zaw**. Cualquier otro valor de *imix* indicará que se lleve a cabo la mezcla.

Atención: cuando se usen los opcodes **ziwm**, **zkwm**, y **zawm**, se debe tener cuidado con que las variables con las que se mezclen los nuevos valores sean puestas a 0 al final (o al principio) de cada ciclo de control o de audio. Si se les siguiera añadiendo señales sus valores podrían llegar a tomar proporciones astronómicas.

Un posible enfoque sería establecer en la mezcla ciertos rangos para las variables zk o za y luego usar **zkcl** o **zacl** para borrar dichos rangos.

61.2.4 EJEMPLOS

```
instr 1
    zkw    kzoom, p8           ; p8 en la paritura determinará dónde
                                ; se escribe kzoom en el espacio zk.
endin

instr 2
    zkw    kzoom, 7           ; escribe kzoom siempre en la posición 7.
endin

instr 3
kxxx phasor 1
kdest =    40 + kxxx * 16     ; escribirá azoom en las posiciones de la 40 a la 55
    zaw    azoom, kdest      ; en una pasada de un ciclo de un segundo
endin
```

61.2.5 AUTOR

Robin Whittle
Australia
Mayo 1997

61.3 zir, zkr, zar, zarg

ir	zir	indx
kr	zkr	kndx
ar	zar	kndx
ar	zarg	kndx, kgain

61.3.1 DESCRIPCIÓN

Lee una posición en el espacio zk a frecuencias de inicialización o control, o una posición en el espacio za a frecuencia de audio.

61.3.2 INICIALIZACIÓN

kndx – apunta a la posición zk o za a ser leída.

kgain –multiplicador para la señal de audio.

61.3.3 EJECUCIÓN

zir lee la señal en la posición *indx* del espacio zk.

zkr lee el array de floats en la posición *kndx* del espacio zk.

zar lee el array de floats en la posición *kndx* del espacio za, es decir el número **ksmps** de valores float de audio a ser procesados en cada ciclo de control.

zarg es similar a **zar** pero multiplica la señal de audio (a-) por un valor de control (k-) *kgain*.

61.3.4 AUTOR

Robin Whittle
Australia
Mayo 1997

61.4 zkmod, zamod, zkcl, zacl

kr	zkmod	ksig, kzkmod
ar	zamod	asig, kzamod
	zkcl	kfirst, klast
	zacl	kfirst, klast

61.4.1 DESCRIPCIÓN

Borra y modula los espacios za y zk.

61.4.2 EJECUCIÓN

ksig –señal de entrada.

kzkmod –controla qué variable zk se usará para la modulación. Un valor positivo indica modulación aditiva, un valor negativo indica una modulación multiplicativa. Un valor 0 indica que no hay cambio en *ksig*. *kzkmod* puede ser parámetro de inicialización (i-) o control (k-).

kfirst –primera posición zk o za en el rango a borrar.

klast – última posición zk o za en el rango a borrar.

zkmod facilita la modulación de una señal por otra, donde la señal modulante procede de una variable zk. Pueden especificarse modulaciones tanto aditivas como multiplicativas.

zamod modula una señal de audio con una segunda, que procede de una variable za. La posición de la variable modulante se controla mediante el parámetro *kzamod* que puede ser tanto de inicialización (i-) como de control (k-). Es una versión audio de **zkmod**.

zkcl borra una o más variables en el espacio zk. Es útil para aquellas variables que se usan como acumuladores en la mezcla de señales de control cada ciclo, pero que deben ser borradas antes de la siguiente serie de cálculos.

zacl borra una o más variables en el espacio za. Es útil para aquellas variables que son usadas como acumuladores en la mezcla de señales de audio cada ciclo, pero que deben ser borradas antes de la siguiente serie de cálculos.

61.4.3 EJEMPLO

```
k1    zkmod      ksig, 23    ; añade a ksig el valor de la posición 23
a1    zamod      asig, -402   ; multiplica asig por el valor de la posición 402
```

61.4.4 AUTOR

Robin Whittle
Australia
Mayo 1997

62 OPERACIONES USANDO TIPOS DE DATOS ESPECTRALES

Estas unidades generan y procesan tipos de datos no estándar, como señales de control y audio subsampleadas en el dominio temporal, así como sus representaciones en el dominio de la frecuencia (espectro).

Los nuevos tipos de datos (**-d**, **-w**) son autodefinibles, y su contenido no puede ser procesado por ningún otro opcode de Csound. Estas unidades son experimentales, y pueden estar sujetas a cambios en próximas versiones. Además, vendrán seguidas por otras nuevas dentro de algún tiempo.

62.1 specaddm, specdiff, specscal, spechist, specfilt

```
wsig  specaddm  wsig1, wsig2[, imul2]
wsig  specdiff  wsignin
wsig  specscal  wsignin, ifscale, ifthresh
wsig  spechist  wsignin
wsig  specfilt  wsignin, ifhtim
```

62.1.1 INICIALIZACIÓN

imul2 (opcional) - si no es 0, escala las magnitudes de *wsig2* antes de sumar. El valor por defecto es 0.

62.1.2 EJECUCIÓN

specaddm - realiza una suma ponderada de dos espectros de entrada. Para cada canal de los dos espectros de entrada, las dos magnitudes se combinan y se escriben en la salida según la fórmula: "magout = mag1in + mag2in * *imul2*". La operación se realiza cuando se detecta que la entrada *wsig1* va a ser nueva. Esta unidad verificará (en la inicialización) la compatibilidad de los dos espectros (igual tamaño, período y tipos de magnitudes).

specdiff - encuentra los valores de diferencia positivos entre dos cuadros consecutivos del espectro. A cada nuevo cuadro (frame) de *wsignin*, cada magnitud se compara con su predecesora, y las variaciones positivas se escriben en el espectro de salida. Esta unidad es útil como detector inicial de energía.

specscal - escala un bloque de datos espectrales de entrada mediante envolventes espectrales. Las tablas de función *ifthresh* y *ifscale* son muestreadas inicialmente a través del rango de frecuencias (logarítmico) del espectro de entrada. Entonces, cada vez que se percibe un nuevo espectro de entrada, los valores muestreados se usan para escalar cada canal de amplitud de la siguiente manera: si *ifthresh* es distinto de 0, cada magnitud se reduce de acuerdo a su correspondiente valor en la tabla (no menor que 0); luego cada magnitud se reescala por el valor *ifscale* correspondiente y el espectro resultante se escribe en *wsig*.

spechist - acumula los valores de cuadros (frames) espectrales sucesivos. A cada nuevo frame de *wsignin*, las acumulaciones actualizadas en cada pista de magnitud se escriben en el espectro de salida. Esta unidad proporciona así un histograma de la distribución espectral variable en el tiempo.

specfilt - filtra cada canal de un espectro de entrada. A cada nuevo cuadro (frame) de *wsignin*, cada magnitud es pasada a un filtro pasa-bajos recursivo de primer orden, cuya constante de tiempo medio ha sido inicialmente configurada muestreando la tabla de función *ifhtim* a través del rango de frecuencia (logarítmico) del espectro de entrada. Esta unidad aplica eficientemente un *factor de persistencia* a los datos de cada canal espectral, y es útil para simular la integración de energía que ocurre durante el proceso de la percepción auditiva. Puede también ser usada como un histograma de la distribución espectral comprimido en el tiempo.

62.1.3 EJEMPLO

```
wsig2      specdiff  wsig1      ; detecta los inicios
wsig3      specfilt  wsig2, 2   ; absorbe lentamente
specdisp   wsig2     , .1      ; y muestra ambos espectros
specdisp   wsig3     , .1
```

62.2 specptrk

```
koct,      specptrk  wsig, kvar, ilo, ihi, istr, idbthresh, inptls, //  
kamp
```

62.2.1 INICIALIZACIÓN

ilo, ihi, istr – condicionadores de rangos de altura (bajo, alto e inicial) expresados en formato de octava decimal.

idbthresh – umbral de energía (en dB) para que tenga lugar el rastreo de altura. Una vez iniciado, el rastreo será continuo hasta que la energía caiga por debajo de la mitad del umbral (6 dB abajo), donde por tanto las salidas *koct* y *kamp* serán 0 hasta que el umbral se sobrepase de nuevo. *idbthresh* es un valor guía. En la inicialización es convertido primero al modo *idbout* del espectro fuente (el punto 6 dB abajo pasa a ser .5, .25 o 1/ raíz de 2 para los modos 0,2 y 3 respectivamente). Los valores son también escalados posteriormente para permitir la suma ponderada de parciales usada durante la correlación. El umbral real es calculado usando el valor *kamp* interno que es visible como segundo parámetro de salida.

inptls, irolloff – el número de armónicos usados como plantilla de correlación en la detección de altura basada espectralmente, y la caída de amplitud para la serie expresada en la forma “fracción por octava” (lineal, así que no debe caerse a valores negativos). Debido a que los parciales y la fracción de caída pueden afectar al rastreo de altura, será útil experimentar un poco: prueba 4 o 5 parciales con .6 de caída como valores iniciales; luego eleva el valor a 10 o 12 parciales con una caída de .75 si lo usas con timbres complejos como el del fagot (con una fundamental débil). El tiempo de computación depende del número de parciales buscados. El máximo es 16.

iodd (opcional) – si no es 0, se emplean sólo los armónicos impares de la serie anterior (por ejemplo, *inptls* = 4 emplearía los parciales 1,3,5 y 7). Esto mejora el rastreo de algunos instrumentos como el clarinete. El valor por defecto es 0 (se emplean todos los parciales).

iconfs (opcional) – número de confirmaciones requeridas por el rastreador de altura para saltar una octava, expresado según fracciones de una octava (esto es, el valor 12 implicaría que un cambio de semitono necesitaría una confirmación (dos aciertos) en el *iprd* de **spectrum**). Este parámetro suaviza los fallos en el análisis de altura tales como errores de octava. Un valor de 0 significa que no sería necesaria ninguna confirmación. El valor por defecto es 10.

interp (opcional) – si no es 0, interpola cada señal de salida (*koct, kamp*) entre frames de la *wsig* entrante. El valor por defecto es 0 (repite los valores de la señal entre frames)

ifprd (opcional) – si no es 0, representa en pantalla el espectro calculado internamente de las fundamentales candidatas. El valor por defecto es 0 (sin representación).

iwtftg (opcional) - indicador de espera. Si no es 0, mantiene cada presentación en pantalla hasta que el usuario lo decida. El valor por defecto es 0 (sin espera).

62.3 specsum, specdisp

```
ksum          specsum      wsig[, interp]  
              specdisp    wsig, iprd[, iwtflg]
```

62.3.1 INICIALIZACIÓN

interp (opcional) - si no es 0, interpola la señal de salida (*koct* o *ksum*). El valor por defecto es 0 (repite el valor de la señal entre los cambios).

iwtflg (opcional) - indicador de espera. Si no es 0, mantiene cada valor mostrado hasta que el usuario decida. El valor por defecto es 0 (sin espera).

62.3.2 EJECUCIÓN

specsum - suma las magnitudes a través de todos los canales del espectro. A cada nuevo frame de *wsig*, las magnitudes se suman y se entregan como una señal *ksum* escalar. Entre frames sucesivos, la salida es bien repetida, bien interpolada a frecuencia de control. Esta unidad produce una señal de control (-k) suma de las magnitudes presentes en los datos espectrales y es, por tanto, una medida variable dependiente de su fuerza total, instante a instante.

specdisp - muestra los valores de las magnitudes del espectro *wsig* cada *iprd* segundos (redondeado a algún valor entero del *iprd* que da lugar a *wsig*).

62.3.3 EJEMPLO

```
ksum          specsum      wsig, 1                ; suma los cuadros espectrales y  
suaviza  
if          ksum < 2000 kgoto zero ; si hay suficiente amplitud  
koct         specptrk    wsig                ; reastrea la altura de la señal  
              kgoto      contin  
zero: koct   =           0                ; si no devuelve 0  
contin:
```

62.4 spectrum

`wsig spectrum` `xsig, iprd, iocts, ifrqs, iq[,ihann, idbout, idsprd, idsinrs]`

62.4.1 DESCRIPCIÓN

Realiza un DFT, distribuido exponencialmente y de Q constante, a través de una señal de control o de audio submuestreada.

62.4.2 INICIALIZACIÓN

ihann (opcional) - aplica una ventana Hamming o Hanning a la entrada. El valor por defecto es 0 (ventana Hamming)

idbout (opcional) - versión codificada de la salida del DFT:

- 0 = magnitud simple,
- 1 = dB,
- 2 = magnitud al cuadrado,
- 3 = raíz dcuadrada de la magnitud.

El valor por defecto es 0 (magnitud simple).

idispd (opcional) - si no es 0, muestra el buffer de submuestreo calculado cada *idispd* segundos. El valor por defecto es 0 (no muestra nada).

idsines (opcional) - si no es 0, muestra las ondas sinusoidales de las ventanas Hamming o Hanning usadas en el filtrado DFT. El valor por defecto es 0 (no muestra las ondas).

62.4.3 EJECUCIÓN

Esta primera unidad llama a la señal *asig* o *ksig* en *iocts* ocasiones sucesivas para submuestrearla en octavas y luego conservar un buffer de valores submuestreados dentro de cada una de las octavas (opcionalmente mostrado como un buffer compuesto cada *idispd* segundos). Luego, cada *iprd* segundos, las muestras conservadas se pasan a través de un banco de filtros (*ifrqs* filtros paralelos espaciados exponencialmente, con una frecuencia/ancho de banda Q de *iq*) y las magnitudes de salida son opcionalmente convertidas (*idbout*) para producir un espectro de banda limitada que puedan leer otras unidades.

Este proceso demanda una computación intensa y el tiempo requerido será directamente proporcional a *iocts*, *ifrqs*, *iq*, e inversamente proporcional a *iprd*. Una configuración *ifrqs* = 12, *iq* = 10, *idbout* = 3, and *iprd* = .02 será adecuada en la mayoría de los casos, pero se anima al usuario a experimentar con dichos valores. *ifrqs* actualmente tiene un máximo de 120 divisiones por octava. Para las entradas de audio, las pistas de frecuencia se afinan para coincidir con el la 440. Esta unidad produce un bloque de datos espectrales *wsig* autodefinidos, cuyas características (*iprd*, *iocts*, *ifrqs*, *idbout*) se pasan, vía el bloque de datos mismo, a todas las *wsig* derivadas. Puede haber cualquier número de unidades de espectro en un instrumento de la orquesta, pero todos los nombres de *wsig* deben ser únicos.

62.4.4 EJEMPLO

```
asig in ; consigue audio externo
wsig spectrum asig,.02,6,12,33,0,1,1 ; submuestra en 6 octavas y calcula
; un dft de 72 puntos (Q 33, salida en dB)
; cada 20 msecs.
```

63 ENTRADA Y SALIDA DE LA SEÑAL:

ENTRADA DE LA SEÑAL

63.1 in, ins, inq, inh, ino, soundin, diskin

a1	in	
a1, a2	ins	
a1, a2, a3, a4	inq	
a1, a2, a3, a4, a5, a6	inh	
a1, a2, a3, a4, a5, a6, a7, a8	ino	
a1	soundin	ifilcod[, iskptim[, iformat]]
a1, a2	soundin	ifilcod[, iskptim[, iformat]]
a1, a2, a3, a4	soundin	ifilcod[, iskptim[, iformat]]
a1[, a2[, a3, a4]]	diskin	ifilcod, kpitch[, iskiptim // [, iwraparound[, iformat]]]

63.1.1 DESCRIPCIÓN

Estas unidades leen datos de audio desde un dispositivo o canal externo.

63.1.2 INICIALIZACIÓN

ifilcod - entero o cadena de caracteres que indica el nombre del fichero de sonido fuente. Un entero indica el fichero *soundin.filcod*; una cadena de caracteres (entre comillas, con espacios permitidos) proporciona el nombre del fichero mismo, opcionalmente con la ruta completa. Si no se especifica dicha ruta, el fichero indicado se busca primero en el directorio actual, luego en el que indica la variable SSDIR (si se encuentra definida) y luego en el indicado por SFDIR. Ver también **GEN01**.

iskptim (opcional) - duración en segundos a omitir del sonido de entrada. El valor por defecto es 0.

iformat (opcional) - especifica el formato de los datos de audio.

- 1 = 8-bit signed char (high-order 8 bits of a 16-bit integer),
- 2 = 8-bit A-law bytes,
- 3 = 8-bit U-law bytes,
- 4 = 16-bit short integers,
- 5 = 32-bit long integers,
- 6 = 32-bit floats.

Si *iformat* = 0 el formato se toma de la cabecera del fichero y, si no hay cabecera, del indicador en la línea de comando de Csound.

iwraparound - 1 = activado, 0 = desactivado (el proceso se repite cíclicamente al llegar al final del fichero en cualquier dirección).

kpitch - puede ser cualquier número real. Un valor negativo indica reproducción hacia atrás. El número proporcionado es una razón de alturas, donde:

```
1 = altura normal,  
2 = octava alta,  
3 = duodécima alta, etc.;  
.5 = octava baja,  
.25 = 2 octavas abajo, etc;  
-1 = altura normal hacia atrás,  
-2 = octava alta hacia atrás, etc..
```

63.1.3 EJECUCIÓN

in, ins, inq, inh, ino - copia los valores actuales del buffer estándar de entrada de audio. Si la línea de comandos tiene un indicador **-i** activado, el sonido se pasa continuamente del flujo de entrada de audio (por ejemplo, *stain* o un fichero de sonido) a un buffer interno. Cualquier número de estas unidades pueden leer libremente de este buffer.

soundin es funcionalmente un generador de audio que deriva su señal de un fichero preexistente. El número de canales leídos es controlado por el número de células resultantes, a1, a2, etc., que deben coincidir con las del fichero de entrada. Una unidad **soundin** abre dicho fichero siempre que el instrumento "anfitrión" se inicializa, luego se cierra de nuevo cada vez que el instrumento se desactiva. Puede haber cualquier número de unidades **soundin** dentro de un instrumento u orquesta. También 2 o más de ellas pueden leer simultáneamente del mismo fichero externo.

diskin es idéntico a **soundin**, excepto en que puede alterar la altura del sonido que esta leyendo.

63.1.4 AUTORES

Barry Vercoe
MIT

Matt Ingots/Mike Berry
Mills College, 1993-1997

64 ENTRADA Y SALIDA DE LA SEÑAL:

ENTRADA DE LA SEÑAL

64.1 **soundout, soundouts, out, outs1, outs2, outs, outq1, outq2, outq3, outq4, outq, outh, outo**

soundout	asig, ifilcod[, iskptim]
soundouts	asig, ifilcod[, iskptim]
out	asig
outs1	asig
outs2	asig
outs	asig1, asig2
outq1	asig
outq2	asig
outq3	asig
outq4	asig
outq	asig1, asig2, asig3, asig4
outh	asig1, asig2, asig3, asig4, asig5, asig6
outo	asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8

64.1.1 DESCRIPCIÓN

Estas unidades leen/escriben los datos de audio desde/a un dispositivo o canal externo.

64.1.2 INICIALIZACIÓN

ifilcod - entero o cadena de caracteres que indica el nombre del fichero de sonido fuente. Un entero indica el fichero *soundin.filcod*; una cadena de caracteres (entre comillas, con espacios permitidos) proporciona el nombre del fichero mismo, opcionalmente con la ruta completa. Si no se especifica dicha ruta, el fichero indicado se busca primero en el directorio actual, luego en el que indica la variable SSDIR (si se encuentra definida) y luego en el indicado por SFDIR. Ver también **GEN01**.

iskptim (opcional) - duración en segundos a omitir del sonido de entrada. El valor por defecto es 0.

64.1.3 EJECUCIÓN

out, outs, outq, outh, outo - envía muestras de audio a un buffer acumulativo de salida (creado al principio de la ejecución) que sirve para recoger las salidas de todos los instrumentos activos antes de que el sonido sea escrito en el disco. Puede haber cualquier número de estas unidades de salida en un instrumento. El tipo (mono, estéreo, quad, hex, or oct) debe coincidir con **nchnls**, pero a partir de la versión 3.50, el compilador intentará cambiar un opcode incorrecto para conformarlo a la sentencia **nchnls**. Las unidades pueden ser elegidas para dirigir sonido a cualquier canal particular: así **outs1** lo envía al canal estéreo 1, **outq3** al canal cuadrafónico 3, etc...

soundout y **soundouts** escriben la salida de audio en un fichero del disco. **soundouts** aún no ha sido implementado.

64.1.4 AUTORES

Barry Vercoe
MIT

Matt Ingots/Mike Berry
Mills College, 1993-1997

65 ENTRADA Y SALIDA DE LA SEÑAL: E/S DE FICHEROS

65.1 `dumpk`, `dumpk2`, `dumpk3`, `dumpk4`, `readk`, `readk2`, `readk3`, `readk4`

	<code>dumpk</code>	<code>ksig, ifilename, iformat, iprd</code>
	<code>dumpk2</code>	<code>ksig1, ksig2, ifilename, iformat, iprd</code>
	<code>dumpk3</code>	<code>ksig1, ksig2, ksig3, ifilename, iformat, iprd</code>
	<code>dumpk4</code>	<code>ksig1, ksig2, ksig3, ksig4, ifilename, iformat, iprd</code>
<code>k1</code>	<code>readk</code>	<code>ifilename, iformat, iprd[, ipol]</code>
<code>k1,k2</code>	<code>readk2</code>	<code>ifilename, iformat, iprd[, ipol]</code>
<code>k1,k2,k3</code>	<code>readk3</code>	<code>ifilename, iformat, iprd[, ipol]</code>
<code>k1,k2,k3,k4</code>	<code>readk4</code>	<code>ifilename, iformat, iprd[, ipol]</code>

65.1.1 DESCRIPCIÓN

Escribe periódicamente señales de control de la orquesta en un fichero externo con un formato específico.

65.1.2 INICIALIZACIÓN

ifilename - cadena de caracteres (entre comillas, espacios permitidos) que indica el nombre de un fichero externo. Puede ser tanto un nombre con la ruta completa, como simplemente el nombre del fichero a crear en el directorio actual.

iformat - formato de salida:

- 1 = 8-bit signed char (high order 8 bits of a 16-bit integer)
- 4 = 16-bit short integers,
- 5 = 32bit long integers,
- 6 = 32-bit floats, 7=ASCII long integers,
- 8 = ASCII floats (2 lugares decimales).

Observa que no se permiten salidas A-law o U-law y que todos los formatos excepto los dos últimos son binarios. El fichero de salida no contendrá información de cabecera.

iprd - período de la salida *ksig* en segundos, redondeado al período de control de la orquesta más cercano. Un valor de 0 indica un 1 ciclo de control (el mínimo permitido), que creará un fichero de salida muestreado a la frecuencia de control de la orquesta.

ipol (opcional) - Si es distinto de 0, y *iprd* implica más de un período de control, interpolará las señales *k*-entre lecturas sucesivas del fichero externo. El valor por defecto es 0 (repite cada señal entre lecturas). Actualmente no disponible.

65.1.3 EJECUCIÓN

Estas unidades permiten guardar en un fichero externo hasta cuatro señales de control generadas. Dicho fichero no contiene información propia de cabecera, pero tendrá la forma de una serie muestreada a intervalos regulares, por lo que será apropiado para posteriores análisis o entradas. Puede haber cualquier número de unidades **readk** en un mismo instrumento u orquesta, y todos pueden leer el mismo o diferentes ficheros. Puede haber cualquier número de unidades **dumpk** en un instrumento u orquesta, pero cada uno debe escribir en un fichero diferente.

65.1.4 HISTORIA DE LOS OPCODES

Los opcodes **dumpk** se llamaban originalmente **kdump**. A partir de la versión 3.493 se desechó ese nombre. **dumpk** debe ser usado ahora en lugar de **kdump**. Los opcodes **readk** fueron originalmente llamados **kread**, pero no fueron implementados hasta la versión 3.52. Sin embargo, el argumento opcional de **readk**, *ipol* es aún ignorado. Se espera poder corregir esta situación en una posterior versión.

65.1.5 EJEMPLO

```
knum      =          knum+1          ; a cada período de control
ktemp     tempest   krms, .02, .1, 3, 2, 800, .005, 0, 60, 4, .1, .995
                                     ;estima el tempo
koct      specptrk  wsig, 6, .9, 0    ;y la altura
          dumpk3   knum, ktemp, cpsoct(koctr), "what happened when", 8 0
                                     ; y los graba
```

65.2 fout, foutk, fouti, foutir, fiopen

	fout	"ifilename", iformat, aout1[, aout2, aout3,...,aoutN]
	foutk	"ifilename", iformat, kout1[, kout2, kout3,...,koutN]
	fouti	ihandle, iformat, iflag, iout1[, iout2, iout3,...,ioutN]
	foutir	ihandle, iformat, iflag, iout1[, iout2, iout3,...,ioutN]
ihandle	fiopen	"ifilename", imode

65.2.1 DESCRIPCIÓN

fout, **foutk**, **fouti** y **foutir** envían a la salida *n* señales de audio (a-), control (k-) o inicialización (i-) a un fichero especificado de *n* canales. **fiopen** puede ser usado para abrir un fichero en uno de los modos especificados.

65.2.2 INICIALIZACIÓN

ifilename - nombre de fichero, dado como una cadena entre comillas.

iformat - indicador para escoger el formato de salida.

1. para **fout** y **foutk** sólo:

- 0 - muestras de 32 bits en coma flotante sin cabecera (fichero PCM binario multicanal).
- 1 - enteros de 16 bits sin cabecera (fichero PCM binario multicanal).
- 2 - enteros de 16 bits con cabecera de tipo WAV (fichero WAV de mono o estéreo de Microsoft).

2. para **fouti** y **foutir** sólo:

- 0 - valores en coma flotante en formato de texto floating point in text format.
- 1 - valores de 32 bits en coma flotante en formato binario.

iflag - escoge el modo de escritura al fichero ASCII (válido solo en modo ASCII; en modo binario *iflag* no tiene sentido, pero debe estar presente de todas formas). *iflag* puede ser un valor escogido entre los siguientes:

- 0 - línea de texto sin prefijo de instrumento.
- 1 - línea de texto con prefijo de instrumento (ver abajo).
- 2 - pone a 0 el tiempo de los prefijos de instrumento (a usar sólo en casos particulares. Ver abajo).

iout,..., *ioutN* - valores a escribir en el fichero.

imode - escoge el modo de abrir el fichero. *imode* puede ser un valor escogido entre los siguientes:

- 0 - abre un fichero de texto para escritura.
- 1 - abre un fichero de texto para lectura.
- 2 - abre un fichero binario para escritura.
- 3 - abre un fichero binario para lectura.

65.2.3 EJECUCIÓN

aout1,... *aoutN* - señales a ser escritas en el fichero.

kout1,...*koutN* - señales a ser escritas en el fichero.

fout escribe muestras de señales de audio en un fichero con cualquier número de canales. El número de canal depende del número de variables *aoutN* (es decir, una señal mono tendrá sólo un argumento de tipo a-, una señal estéreo tendrá dos argumentos de tipo a-, etc.). El número máximo de canales está fijado en 64. Puede haber varios opcodes **fout** en el mismo instrumento, apuntando a ficheros diferentes.

Observa que, a diferencia de **out**, **outs** y **outq**, **fout** no pone a 0 la variable de audio, así que debes hacerlo tú después de llamar a **fout**, si se va a usar polifonía. Puedes usar los opcodes **incr** y **clear** para dicha tarea. **foutk** opera en la misma manera que **fout**, pero con señales de control. *iformat* sólo puede tomar dos valores: 0 ó 1.

fouti y **foutir** escriben valores de tipo i- en un fichero. El uso principal de estos opcodes es generar un fichero de partitura durante una sesión en tiempo real. Con este propósito, el usuario debe poner *iformat* a 0 (salida como fichero de texto) e *iflag* a 1, lo que permite la salida de un prefijo consistente en las cadenas *inum*, *actiontime*, y *duration*, antes de los valores de los argumentos *iout1*...*ioutN*. Los argumentos en el prefijo se refieren al número de instrumento, instante de activación y duración de la nota actual.

La diferencia entre **fouti** y **foutir** es que, en el caso de **fouti**, cuando *iflag* es puesto a 1, la duración del primer opcode es indefinida (así que se reemplaza por un punto). Mientras que **foutir** es definido al final de cada nota, de manera que la correspondiente línea de texto sólo es escrita al final de la nota actual (para que pueda ser posible reconocer su duración). El fichero correspondiente está referenciado por el valor *ihandle* generado por el opcode **fiopen** (ver abajo). Así que **fouti** y **foutir** pueden ser usados para generar una orquesta de Csound mientras tiene lugar una interpretación en tiempo real.

fiopen abre un fichero para que sea usado por la familia de opcodes **fout**. Debe estar definido en la sección de cabecera, fuera de cualquier instrumento. Devuelve un número, *ihandle*, que apunta inequívocamente al fichero abierto.

Observa que **fout** y **foutk** pueden usar tanto una cadena de caracteres que defina la ruta y el nombre de un fichero, como un número *ihandle* generado por **fiopen**. Mientras que con **fouti** y **foutir** el fichero destino sólo puede ser especificado por medio de un número *ihandle*.

65.2.4 AUTOR

Gabriel Maldonado

Italia

1999

Nuevo en la versión 3.56

65.3 fin, fink, fini

```
fin    "ifilename", iskipframes, iformat, ain1[, ain2, ain3,...,ainN]  
fink  "ifilename", iskipframes, iformat, kin1[, kin2, kin3,...,kinN]  
fini  "ifilename", iskipframes, iformat, in1[, in2, in3,...,inN]
```

65.3.1 DESCRIPCIÓN

Lee señales de un fichero (a frecuencias de audio, control o inicialización).

65.3.2 INICIALIZACIÓN

ifilename - fichero de entrada (puede ser una cadena o un número *ihandle* generado por **fiopen**).

iskipframes - número de *frames* a saltarse al comienzo (cada *frame* contiene una muestra de cada canal).

iformat - número especificando el formato del fichero de entrada.

para **fin** y **fink**:

0 - 32 bits en coma flotante sin cabecera .

1 - enteros de 16 bits sin cabecera.

para **fini**:

0 - valores en coma flotante en formato de texto (con bucle; ver abajo).

1 - valores en coma flotante en formato de texto (sin bucle; ver abajo).

2 - valores de 32 bits en coma flotante en formato binario (sin bucle).

65.3.3 EJECUCIÓN

fin es el complemento a **fout**. Lee un fichero multicanal para generar señales de audio. Por el momento no se soporta ninguna cabecera en el formato del fichero. El usuario debe asegurarse de que el número de canales del fichero de entrada es el mismo que el número de argumentos *ainX*. **fink** es igual a **fin**, pero opera sólo a frecuencia de control.

fini es el complemento a **fouti** y **foutir**. Lee los valores cada vez que la nota correspondiente es activada. Cuando *iformat* es puesto a 0, si se alcanza el final del fichero, el puntero puesto a 0, comenzando de nuevo la lectura desde el principio. Cuando *iformat* es puesto a 1 o 2, no se permiten bucles, así que al final del fichero, las variables correspondientes serán puestas a 0.

65.3.4 AUTOR

Gabriel Maldonado

Italia

1999

Nuevo en la versión 3.56

65.4 vincr, clear

```
vincr      asig, aincr
clear     avar1[,avar2, avar3,...,avarN]
```

65.4.1 DESCRIPCIÓN

vincr añade la variable de audio de otra señal, es decir acumula la salida. **clear** pone a 0 una serie de señales de audio.

65.4.2 EJECUCIÓN

asig - variable de audio a la que añadir la señal *aincr*.

aincr - señal que se añade a *asig*.

avar1 [,*avar2*, *avar3*,...,*avarN*] - señales a ser puestas a 0.

vincr (acumulador de variables) y **clear** están pensados para ser usados juntos. **vincr** guarda el resultado de la suma de dos variables de audio en la primera de las variables (que se entiende como un acumulador en polifonía). La variable acumulador puede ser usada para enviar señales a la salida por medio del opcode **fout**. Después de la operación de escritura en el disco, la variable acumulador debe ser puesta a 0 por medio del opcode **clear** (o de otra manera "explotará").

65.4.3 AUTOR

Gabriel Maldonado
Italia
1999
Nuevo en la versión 3.56

66 ENTRADA Y SALIDA DE LA SEÑAL: CONSULTAS A FICHEROS DE SONIDO

66.1 filelen, filesr, filenchnls, filepeak

```
ir    filelen          "ifilcod"  
ir    filesr           "ifilcod"  
ir    filenchnls      "ifilcod"  
ir    filepeak        "ifilcod"[, ichnl]
```

66.1.1 DESCRIPCIÓN

Obtiene información de un fichero de sonido.

66.1.2 INICIALIZACIÓN

ifilcod – fichero a ser examinado.

ichnl – canal a ser usado para calcular el valor de pico. El valor por defecto es 0.

ichnl = 0 devuelve el valor de pico de todos los canales

ichnl > 0 devuelve el valor de pico del canal *ichnl*

66.1.3 EJECUCIÓN

filelen devuelve la longitud en segundos del fichero de sonido *ifilcod*. **filesr** devuelve la frecuencia de muestreo del fichero *ifilcod*. **filenchnls** devuelve el número de canales del fichero *ifilcod*. **filepeak** devuelve el valor de pico absoluto del fichero de sonido *ifilcod*. Actualmente **filepeak** sólo soporta ficheros AIFF-C.

66.1.4 AUTOR

Matt Ingalls

Julio, 1999

Nuevo en la versión 3.57

66.2 print, display, dispfft

```
print      iarg[, iarg,...]
display    xsig, iprd[, inprds[, iwtflg]]
dispfft    xsig, iprd, iwsiz[, iwtyp[, idbouti[, iwtflg]]]
```

66.2.1 DESCRIPCIÓN

Estas unidades imprimirán valores de inicialización de la orquesta, o producirán una representación gráfica de señales de control o de audio. Usan ventanas X11 si están habilitadas (o si el indicador -g está activo), si no la representación será un gráfico usando caracteres ASCII.

66.2.2 INICIALIZACIÓN

iprd - período de la representación en segundos.

iwsiz - tamaño de la ventana de entrada en muestras. Una ventana de *iwsiz* puntos producirá una transformada de Fourier de *iwsiz*/2 puntos, distribuida linealmente a lo largo del espectro desde 0 a $\pi/2$. *iwsiz* debe ser una potencia de 2, con un valor mínimo de 16 y uno máximo de 4096. Las ventanas pueden solaparse.

iwtyp (opcional) - tipo de ventana. 0 = rectangular, 1 = Hanning. El valor por defecto es 0 (rectangular).

idbout (opcional) - unidades de salida para los coeficientes de Fourier. 0 = magnitud, 1 = dB. El valor por defecto es 0 (magnitudes).

iwtflg (opcional) - indicador de espera. Si no es 0, cada representación en pantalla es congelada hasta que el usuario decida. El valor por defecto es 0 (sin espera).

66.2.3 EJECUCIÓN

print - imprime el valor actual de los argumentos (o expresiones) de inicialización *iarg* a cada pasada de inicialización del instrumento.

display - representa gráficamente una señal de audio o control *xsig* cada *iprd* segundos, en la forma amplitud vs tiempo.

dispfft - representa gráficamente la transformada de Fourier de una señal de audio o de control (*asig* o *ksig*) cada *iprd* segundos usando el método de la transformada rápida de Fourier.

66.2.4 EJEMPLO

```
k1    envlpx    1, .03, p3, .05, 1, .5, .01    ; genera el envolvente de una nota
      display   k1, p3                        ; y lo representa en pantalla
```

66.3 printk, printks

```
printk      ispace, kval [, itime]
printks     "txtstring", itime, kval1, kval2, kval3, kval4
```

66.3.1 DESCRIPCIÓN

Estos opcodes están pensados para facilitar la depuración del código de la orquesta.

66.3.2 INICIALIZACIÓN

ispace - cuántos espacios hay que insertar antes de imprimir (máximo 130).

itime - cuánto tiempo en segundos debe transcurrir entre impresiones (por defecto un segundo).

txtstring - texto a imprimir primero. Puede ser de hasta 130 caracteres y estar entre comillas.

66.3.3 EJECUCIÓN

kvalx - valores de control a imprimir. Estos valores se especifican en "*txtstring*" con el especificador de valores estándar del lenguaje C %f, en el orden dado. Dale un valor 0 a aquellos que no se vayan a usar.

printk imprime un valor -k cada ciclo de control, cada segundo, o a un intervalo de tiempo especificado. Primero se imprime el número del instrumento, luego el tiempo absoluto en segundos, luego un número especificado de espacios y luego el valor de control *kval*. El número variable de espacios permite distribuir los diferentes valores en la pantalla, facilitando su lectura.

printks imprime números y texto, con hasta 4 valores que pueden ser de tipo -i o -k. Es muy flexible y usado junto con los opcodes de posicionamiento de cursor, puede ser usado para escribir valores en posiciones de la pantalla según va teniendo lugar el procesado de Csound.

Un modo especial de operación permite a **printks** convertir el parámetro de entrada *kval1* a un valor en el rango de 0 a 255 y usarlo como primer carácter a imprimir. Esto permite a Csound enviar caracteres arbitrarios a la consola. Para conseguir esto, haz que el primer carácter de la cadena sea # y luego, si lo deseas, continua con texto normal y otros especificadores de formato. Pueden usarse hasta tres especificadores de formato más, accediendo cada uno de ellos a *kval2*, *kval3* y *kval4* respectivamente.

Ambos opcodes pueden ejecutarse cada ciclo de control del instrumento en donde se hallen. Para hacerlo, *itime* debe tomar un valor de 0.

Cuando *itime* no es 0, el opcode imprime en el primer ciclo de control en que es llamado, y luego cada vez que el período de *itime* ha transcurrido. Los ciclos empiezan desde que el opcode es inicializado, normalmente en la inicialización del instrumento.

66.3.4 FORMATO DE SALIDA DE LA IMPRESIÓN

Pueden usarse los caracteres de control estándar del lenguaje C, pero deben ser precedidos de un carácter \ adicional:

```
\\n or \\N Nueva línea  
\\t or \\T Tabulación
```

El formato estándar %f del lenguaje C se usa para imprimir *kval1*, *kval2*, *kval3*, y *kval4*. Por ejemplo:

```
%f imprime con máxima precisión 123.456789  
%6.2f imprime 1234.56  
%5.0p imprime 12345
```

66.3.5 EJEMPLOS

Lo siguiente:

```
printfs \"Volume = %6.2f Freq = %8.3f\\n\\n\", 0.1, kval, kfreq, 0, 0
```

imprimiría:

```
Volume = 1234.56 Freq = 12345.678
```

Lo siguiente:

```
printfs \"#x\\y = %6.2\\n\\n\", 0.1, kxy, 0, 0, 0
```

imprimiría una tabulación seguida de:

```
x\\y = 1234.56
```

66.3.6 AUTOR

Robin Whittle
Australia
Mayo 1997

66.4 printk2

```
printk2 kvar [, numspaces]
```

66.4.1 INICIALIZACIÓN

numspaces - número de espacios a imprimir antes del valor de *kvar*.

66.4.2 EJECUCIÓN

kvar - señal a imprimir.

Derivado del **printk** de Robin Whittle, **printk2** imprime un nuevo valor de *kvar* cada vez que ésta cambia. Es útil para dirigir los cambios de los controladores MIDI cuando se usan deslizadores.

ATENCIÓN: no uses este opcode con señales normales de control que varíen constantemente porque puede bloquear el ordenador, al ser la frecuencia de impresión demasiado rápida.

66.4.3 AUTOR

Gabriel Maldonado

Italia

1998 (Nuevo en la versión 3.48)

67 LA PARTITURA NUMÉRICA ESTÁNDAR

67.1 PREPROCESADO DE PARTITURAS ESTANDAR

Una partitura (colección de sentencias de partitura) se divide en secciones ordenadas en el tiempo por las **sentencias s** (de sección). Antes de ser leída por la orquesta, la partitura es preprocesada sección a sección. Cada una de ellas es procesada normalmente por tres rutinas: **Carry**, **Tempo**, y **Sort**.

67.1.1 CARRY

Dentro de un grupo de **sentencias i** consecutivas cuyos números enteros p1 se correspondan, cualquier campo p vacío tomará su valor del mismo campo p de la sentencia precedente. Un campo p vacío puede ser indicado por un punto (.) delimitado por espacios. No se requiere ningún punto después del último campo p no vacío. La salida del preprocesado de Carry mostrará los valores explícitos. El procesado de Carry no se ve afectado por comentarios o líneas en blanco. Se desactiva sólo por una **sentencia** que no sea **i** o por una **sentencia i** con un valor de p1 distinto.

Otra característica adicional está disponible sólo para el campo p2. El símbolo + en p2 hará que éste tome el valor de la suma de los campos p2 y p3 de la **sentencia i** precedente. Esto permite que el instante de activación de las notas sea automáticamente calculado a partir de la suma de la duración y el instante de comienzo de las notas precedentes. El símbolo + puede ser procesado con Carry también. Es legal sólo en campos p2. Por ejemplo, las sentencias:

```
i1 0 .5 100  
i . +  
i
```

darán lugar a:

```
i1 0 .5 100  
i1 .5 .5 100  
i1 1 .5 100
```

El procesado de Carry puede ser usado libremente. Su uso, especialmente en partituras largas, puede reducir mucho el tiempo de tecleo y simplificar cambios posteriores.

67.1.2 TEMPO

Esta operación modifica el tempo de una sección de partitura según la **sentencia t**. La operación de tempo convierte p2 (y, para las **sentencias i**, p3) de su formato original a pulsos en segundos reales, ya que esa es la unidad de tiempo requerida por los opcodes de la orquesta. Después de la modificación de **Tempo**, los archivos de partitura tendrán un formato legible por la orquesta siguiendo la forma: **i p1 p2pulsos p2segundos p3pulsos p3segundos p4 p5....**

67.1.3 SORT

Esta rutina ordena todas sentencias que implican una acción en el tiempo según el orden cronológico especificado por su valor p2. También ordena eventos coincidentes en orden de precedencia. Siempre que una **sentencia f** o **i** tienen el mismo valor de p2, la primera que aparezca será precedente. Siempre que dos o más **sentencias i** tienen el mismo valor de p2, son ordenadas ascendentemente según el valor de p1. Si estas tienen a su vez el mismo valor p1, se ordenarán ascendentemente de acuerdo al valor de p3. La ordenación de la partitura se realiza sección a sección (ver **sentencias s**). La ordenación automática implica que las sentencias de la partitura pueden aparecer en cualquier orden dentro de una sección.

67.1.4 N.B.

Las operaciones **Carry**, **Tempo** y **Sort** se combinan en una única pasada de 3 fases al fichero de partitura para producir un nuevo fichero en el formato necesario para que la orquesta pueda leerlo (ver el ejemplo de **Tempo**). El procesado puede ser invocado explícitamente por el comando **Ssort** o implícitamente por Csound, que procesa la partitura automáticamente antes de llamar a la orquesta. Los ficheros fuente normales y los que puede leer la orquesta están ambos en formato ASCII y pueden ser revisados o modificados por editores de texto estándar. Pueden usarse rutinas diseñadas por el usuario para modificar ficheros de partitura antes o después del procesado descrito anteriormente, con tal de que no se viole el formato legible por la orquesta. Secciones de diferentes formatos pueden ser secuencialmente procesadas y aquellas de igual formato pueden ser mezcladas para su ordenación automática.

67.2 SÍMBOLOS DE SIGUIENTE-P Y PREVIO-P

Al final de cualquier operación **Carry**, **Tempo**, o **Sort** hay tres acciones adicionales que se llevan a cabo durante la escritura del fichero de salida: **siguiente-p (np)**, **previo-p (pp)** y **ramping (<)**.

Los campos de p de las **sentencias i** que contienen los símbolos **np_x** o **pp_x** (donde *x* es un entero) serán reemplazados por el valor de campo p apropiado que se encuentre en la siguiente o anterior, respectivamente, **sentencia i** que tenga el mismo valor de p1. Por ejemplo, el símbolo **np7** será reemplazado por el valor encontrado en el campo p7 de la próxima nota que tocará el instrumento. Los símbolos **np** y **pp** son recursivos y pueden apuntar a otros símbolos iguales. Sin embargo, estas referencias deberán terminar siempre en un número real o en un símbolo **ramp**. Se deberán evitar referencias en bucles cerrados. Los símbolos **np** y **pp** pueden ser procesados por Carry. Las referencias de **np** y **pp** no pueden cruzar el límite de una sección. A cualquier referencia, hacia arriba o hacia abajo, a una sentencia de nota no existente se le dará un valor 0.

Por ejemplo: las sentencias

```
i1 0 1 10 np4 pp5
i1 1 1 20
i1 1 1 30
```

darán lugar a

```
i1 0 1 10 20 0
i1 1 1 20 30 20
i1 2 1 30 0 30
```

Los símbolos **np** y **pp** pueden proporcionar a un instrumento un "conocimiento" contextual de la partitura, permitiendo glissando o crescendos, por ejemplo, hacia la altura o la dinámica de algún evento futuro (que puede o puede no ser adyacente). Observa que mientras la opción **Carry** aplicará los símbolos **np** y **pp** a las sentencias desordenadas, la operación que interpreta dichos símbolos actúa siempre sobre una versión de la partitura completamente ordenada y con el tempo final.

67.3 RAMPING

Los campos p de las **sentencias i** que contengan el símbolo < serán reemplazados por los valores derivados de la interpolación lineal de una "rampa" en el tiempo. Estas rampas vienen definidas en cada extremo por el primer número encontrado en el mismo campo p de cualquier nota siguiente o precedente del mismo instrumento. Por ejemplo las sentencias:

```
i1 0 1 100
i1 1 1 <
i1 2 1 <
i1 3 1 400
i1 4 1 <
i1 5 1 0
```

darán lugar a

```
i1 0 1 100
i1 1 1 200
i1 2 1 300
i1 3 1 400
i1 4 1 200
i1 5 1 0
```

Las rampas no pueden cruzar el límite de una sección ni pueden tener como extremos símbolos **np** o **pp** (aunque pueden ser referenciadas por ellos). Estos símbolos **ramp** son ilegales en p1, p2 y p3. Pueden ser procesados por Carry. Observa, sin embargo, que mientras la opción **Carry** aplicará los símbolos **ramp** a las sentencias desordenadas, la operación que interpreta dichos símbolos actúa siempre sobre una versión con el tempo final y completamente ordenada de la partitura. De hecho la interpolación lineal se realiza en la partitura ordenada y con su tempo final, así que una rampa que abarque un grupo de notas en acelerando permanecerá lineal con respecto a sus tiempos cronológicos reales.

A partir de la versión 3.52, se han ido añadiendo nuevos tipos de rampas. Usando los símbolos (o) se obtendrá una interpolación exponencial, similar a la de **expon**. Los símbolos { y }, que definían en la versión 3.49 este tipo de interpolación exponencial, han sido desechados. Usando el símbolo ~ se conseguirá una distribución aleatoria uniforme entre el primer y último valor de la rampa. Usa estas funciones con las mismas reglas que las de la interpolación lineal descrita arriba.

67.4 MACROS DE LA PARTITURA

```
#define NOMBRE # texto a reemplazar #  
#define NOMBRE(a' b' c') # texto a reemplazar #  
$NOMBRE.  
#undef NOMBRE
```

67.4.1 DESCRIPCIÓN

Las macros son sustituciones textuales que toman lugar en el momento en que la orquesta está siendo leída. El sistema de macros en Csound es muy simple y usa dos caracteres especiales para indicar su presencia, los signos # y \$. Esto permite una más fácil escritura de las partituras, y proporciona una alternativa elemental a los sistemas de generación de partituras complejos. El sistema de macros es similar pero completamente independiente del sistema de macros del lenguaje de la orquesta.

#define NOMBRE –define una macro simple. El nombre de la macro debe empezar con una letra y puede consistir en cualquier combinación de letras y números. Se distinguen mayúsculas y minúsculas. El nombre de las variables debe ser fijo. Se puede obtener más flexibilidad usando una macro con argumentos, tal y como se describe abajo.

#define NOMBRE (a' b' c') - define una macro con argumentos. Esto puede ser usado en situaciones más complejas. El nombre de una macro debe empezar con una letra y consistir en cualquier combinación de letras y números. Dentro del texto a reemplazar, los argumentos pueden ser sustituidos por \$A. De hecho, la implementación define los argumentos como macros simples. Puede haber hasta 5 argumentos, y los nombres pueden consistir en cualquier combinación de letras. Recuerda que se distingue entre mayúsculas y minúsculas.

\$NOMBRE. - llama a una macro previamente definida. Para usar una macro, su nombre es usado tras el carácter \$. Este nombre termina con el último carácter que no sea ni una letra ni un número. Si es preciso que el nombre no termine con un espacio, un punto, que será ignorado, puede ser usado para terminar dicho nombre. Por ejemplo, la cadena \$NOMBRE. será reemplazada por el texto definido en la macro. Por supuesto, este texto puede llamar a otras macros.

#undef NOMBRE – borra el nombre de una macro. Si una macro no va a ser usada en adelante puede ser borrada con **#undef NOMBRE**.

67.4.2 INICIALIZACIÓN

replacement text # - El texto que reemplazará cada aparición del nombre de la macro es cualquier cadena de caracteres (excepto #) y puede extenderse en más de una línea. Este texto está colocado entre los signos #, que aseguran que no se tomarán accidentalmente otros caracteres fuera de estos límites.

67.4.3 EJECUCIÓN

Se necesita algún cuidado con las macros textuales, ya que a veces juegan malas pasadas. Ellas no distinguen significado alguno, así que los espacios son significativos, por lo cual, en la definición, el texto que reemplazará a la macro va entre # #, a diferencia del lenguaje C. Usadas con cuidado, las macros son un concepto potente, pero no se puede abusar de ellas.

67.4.4 OTRO USO PARA LAS MACROS

Cuando escribimos partituras complejas a veces es demasiado fácil olvidar a que instrumento se refiere un determinado número. Se pueden usar macros para dar nombres a los instrumentos. Por ejemplo:

```
#define Flute #i1#
#define Whoop #i2#
$Flute. 0 10 4000 440
$Whoop.5 1
```

67.4.5 EJEMPLOS

67.4.5.1 Macro Simple

una nota tiene una serie de campos-p que se repiten:

```
#define ARGS # 1.01 2.33 138#
i1 0 1 8.00 1000 $ARGS
i1 0 1 8.01 1500 $ARGS
i1 0 1 8.02 1200 $ARGS
i1 0 1 8.03 1000 $ARGS
```

Esto dará lugar antes del proceso de ordenación a:

```
i1 0 1 8.00 1000 1.01 2.33 138
i1 0 1 8.01 1500 1.01 2.33 138
i1 0 1 8.02 1200 1.01 2.33 138
i1 0 1 8.03 1000 1.01 2.33 138
```

Esto puede ahorrar tiempo de tecleo y hace que la revisión del texto sea mucho más fácil. Si hubiera dos series de campos-p, una podría contener una segunda macro (no hay un límite real en el número de macros que uno puede definir).

```
#define ARGS1 # 1.01 2.33 138#
#define ARGS2 # 1.41 10.33 1.00#
i1 0 1 8.00 1000 $ARGS1
i1 0 1 8.01 1500 $ARGS2
i1 0 1 8.02 1200 $ARGS1
i1 0 1 8.03 1000 $ARGS2
```

67.4.5.2 Macros con argumentos

```
#define ARG(A) # 2.345 1.03 $A 234.9#
i1 0 1 8.00 1000 $ARG(2.0)
i1 + 1 8.01 1200 $ARG(3.0)
```

da lugar a:

```
i1 0 1 8.00 1000 2.345 1.03 2.0 234.9
i1 + 1 8.01 1200 2.345 1.03 3.0 234.9
```

67.4.6 AUTOR

John ffitch, Universidad de Bath/Codemist Ltd.
Bath, Reino Unido. Abril, 1998 (Nuevo en la versión 3.48)

67.5 PARTITURAS CON VARIOS FICHEROS

A veces es conveniente tener la partitura dividida en varios ficheros, por ejemplo para tener cada instrumento en un fichero separado. La nueva opción `#include`, que es una parte del sistema de macros, soporta esta manera de configurar la partitura. Una línea con el siguiente texto:

```
#include "nombrefichero"
```

donde el carácter `:` puede ser reemplazado por cualquier otro que haga las veces de dicho carácter. Para la mayoría de los casos, las comillas serán las más apropiadas. El nombre del fichero puede incluir la ruta completa.

#include toma su entrada del fichero referenciado de principio a fin cuando dicha entrada apunta a una entrada previa. Hay actualmente un límite de 20 en los ficheros y macros incluidos.

Otro posible uso de **#include** puede ser definir una serie de macros que sean significativas en el estilo del compositor. Se pueden usar también para escribir secciones repetidas:

```
s
#include "sección1"
;; repite esto
s
#include "sección1"
```

Hay, sin embargo, otra manera de repetir secciones que veremos a continuación, usando **sentencias r, m y n**.

67.5.1 AUTOR

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Abril, 1998 (Nuevo en la versión 3.48)

67.6 EVALUACIÓN DE EXPRESIONES

En versiones más tempranas de Csound los números presentados en la partitura eran usados tal cual. Hay ocasiones en las que algunas operaciones sencillas facilitarían la labor. Esta necesidad se hace más evidente cuando se usan macros. Para ayudar en esta cuestión se ha introducido en la partitura la sintaxis de expresiones aritméticas entre corchetes []. Las expresiones permitidas son las que usan las operaciones +, -, * y / junto con el uso de () para dar prioridad a ciertas operaciones. Estas expresiones pueden incluir números y, naturalmente, macros cuyos valores son cadenas numéricas o aritméticas. Todos los cálculos se efectúan con valores en coma flotante. Observa que el operador de negación todavía no ha sido implementado.

Nuevos en la versión 3.56 son @x (siguiente potencia de 2 mayor o igual que x) y @@x (siguiente "potencia de 2 + 1" mayor o igual que x).

67.6.1 EJEMPLO

```
r3 CNT
i1 0 [0.3*$CNT.]
i1 + [($CNT./3)+0.2]
e
```

Al contener las tres copias de la sección la macro \$CNT, con diferentes valores 1,2 y 3, este ejemplo dará lugar a:

```
s
i1 0 0.3
i1 0.3 0.533333
s
i1 0 0.6
i1 0.6 0.866667
s
i1 0 0.9
i1 0.9 1.2
e
```

Este ejemplo era un tanto radical, pero el sistema de evaluación de expresiones puede ser usado para producir cambios más sutiles en las secciones repetidas.

67.6.2 AUTOR

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Abril, 1998 (Nuevo en la versión 3.48)

67.7 SENTENCIA f (O SENTENCIA DE TABLA DE FUNCIÓN)

f p1 p2 p3 p4 ...

67.7.1 DESCRIPCIÓN

Esta sentencia hace que una subrutina GEN almacene valores en una tabla de función, para usarla con los instrumentos de la orquesta.

67.7.2 CAMPOS P

p1 número de la tabla (de 1 a 200) con el que será conocida la función almacenada
Un número negativo indica que la tabla debe ser destruida.

p2 instante de comienzo de la generación (o destrucción) de la función en pulsos.

p3 Tamaño de la tabla de función (es decir, número de puntos)
Debe ser una potencia de 2 o una potencia de 2 más 1 (ver abajo)
El tamaño máximo de la tabla es de 16777216 (2^{24}) puntos

p4 número de la rutina GEN a llamar (ver Rutinas GEN)
Un valor negativo hará que se omita el proceso de reescala.

p5		
p6		Parámetros cuyo significado se determina por una rutina GEN en particular
.		
.		

67.7.3 CONSIDERACIONES ESPECIALES

Las tablas de función son arrays de valores en coma flotante. Estos arrays pueden tener cualquier longitud que coincida con una potencia de 2; la asignación de espacio siempre proporciona 2 elevado a n puntos, más un elemento límite añadido. Este elemento límite, usado en lecturas con interpolación, puede ser automáticamente asignado de acuerdo con el propósito de la tabla: si el tamaño *size* es una potencia exacta de 2, el elemento límite será una copia del primer elemento de la tabla. Esto es muy apropiado para lecturas cíclicas con interpolación, como por ejemplo la de **oscili**, etc. y puede ser usado incluso con osciladores que no usen interpolación, como **oscil**, para mayor seguridad. Si el tamaño *size* es " $2^n + 1$ ", el elemento límite añadido prolonga automáticamente el dibujo de los valores de la tabla. Esto es apropiado para funciones que van a ser escaneadas sólo una vez, como en **envplx**, **oscil1**, **oscilli**, etc.

El espacio de la tabla se asigna en memoria primaria, junto con el espacio de los datos del instrumento. El número máximo de tabla es 200, valor que puede ser aumentado a voluntad.

Una tabla de función existente puede ser eliminada por una **sentencia f** que tenga un valor de **p1** negativo y un instante de comienzo adecuado. También puede ser eliminada una tabla de función por la generación de otra tabla con el mismo valor de **p1**. Las funciones no son borradas automáticamente al final de la sección. El valor del tiempo de acción **p2** se trata de la misma manera que en las **sentencias i** con respecto a la ordenación y modificación mediante **sentencias t**. Si una **sentencia f** y una **sentencia i** tienen el mismo valor de **p2**, la ordenación dará precedencia a la **sentencia f** para que la tabla de función esté disponible durante la inicialización de la nota.

Una sentencia **f 0** ($p1 = 0$, $p2 =$ valor positivo) puede ser usada para crear una indicación de comienzo sin ninguna acción real asociada. Este tipo de marcadores son útiles para rellenar una sección de la partitura (ver **sentencias s**).

67.8 SENTENCIAS **i** (SENTENCIAS DE NOTA O INSTRUMENTO)

`i p1 p2 p3 p4 ...`

67.8.1 DESCRIPCIÓN

Esta sentencia llama a un instrumento para hacerlo activo en un instante específico y durante un tiempo determinado. Los valores de los campos `p` se pasan a ese instrumento antes de su inicialización, y permanecen válidos durante toda su ejecución.

67.8.2 CAMPOS **P**

p1 número de instrumento (de 1 a 200), normalmente un entero no negativo. Una parte fraccionaria puede proporcionar un indicador adicional para especificar ligaduras entre notas determinadas. Un valor negativo (incluido el indicador) puede usarse para desactivar una nota tenida determinada.

p2 instante de comienzo en unidades arbitrarias llamadas pulsos.

p3 duración en pulsos (normalmente positiva).
Un valor negativo iniciará una nota tenida (ver también **hold**)
Un valor 0 invocará una pasada de inicialización sin ejecución (ver **instr**)

p4 |
p5 | Parámetros cuyo significado se determina por una rutina GEN en particular
: |
: |

67.8.3 CONSIDERACIONES ESPECIALES

Los pulsos son evaluados como segundos, a menos que haya una **sentencia t** en esa sección de la partitura o un **indicador t** en la línea de comando de Csound que cambie este criterio.

El instante de comienzo (o instante de activación) es relativo al principio de una sección (ver **sentencia s**), a la que se asigna un tiempo 0.

Las sentencias de nota dentro de una sección pueden ser colocadas en cualquier orden. Antes de ser enviadas a la orquesta, las sentencias de la partitura aún sin ordenar deben ser procesadas primero por el algoritmo de ordenación, que las reorganizará ascendentemente según los valores de `p2`. Las notas que tengan el mismo valor de `p2`, serán ordenadas ascendentemente de acuerdo a los valores de `p1`. Si los `p1` son también iguales, entonces se ordenan ascendentemente según los valores de `p3`.

Las notas pueden ser apiladas verticalmente, es decir, un único instrumento puede ejecutar acordes de cualquier número de notas (las copias necesarias del espacio de datos del instrumento se repartirán en memoria dinámicamente por la orquesta). Cada nota normalmente se desactiva cuando su duración `p3` ha concluido o cuando se recibe un mensaje MIDI de desactivación de nota. Un instrumento puede modificar su propia duración bien cambiando su valor de `p3` en la inicialización de la nota, bien prolongándose a sí mismo mediante la acción de una unidad **linenr**.

Un instrumento puede ser activado indefinidamente bien asignándole un valor negativo de `p3`, bien incluyendo un opcode **ihold** en su código de inicialización. Si una nota tenida está activa, una **sentencia i**

con un p1 idéntico no causará una nueva asignación de memoria sino que usará el espacio de la nota mantenida. Los nuevos campos p (incluyendo p3) estarán ahora activos y se ejecutará una pasada de inicialización en la que las unidades podrán bien ser inicializadas nuevamente, bien continuar si es que hay una nota ligada (ver **tigoto**). Una nota tenida puede ser sucedida tanto por otra nota tenida, como por una nota de duración finita. Una nota mantenida seguirá ejecutándose a través de los finales de sección (ver **sentencias s**). Sólo se parará por un opcode **turnoff** o por una **sentencia i** con un valor negativo idéntico de p1 o por una **sentencia e**.

Las activaciones múltiples de un mismo instrumento son posibles en Csound (normalmente, aunque no necesariamente, con notas distintas), mantenidas simultáneamente mediante valores de p3 negativos. El instrumento puede entonces tomar nuevos valores desde la partitura. Esto es especialmente útil para evitar código plagado de opcodes **linseg**, y puede ser conseguido añadiendo una parte decimal al número del instrumento.

Por ejemplo, para mantener sonando tres copias del instrumento 10 formando un acorde simple:

```
i10.1 0 -1 7.00
i10.2 0 -1 7.04
i10.3 0 -1 7.07
```

Las siguientes sentencias **i** pueden referirse a las notas que ya estaban sonando y, si la definición del instrumento es correcta, los nuevos campos p pueden ser usados para alterar el carácter de las notas en curso.

Por ejemplo, para subir una octava el anterior acorde y luego soltarlo:

```
i10.1 1 1 8.00
i10.2 1 1 8.04
i10.3 1 1 8.07
```

La definición del instrumento debe tener en cuenta esto, especialmente si se quieren evitar clicks (ver el ejemplo de abajo).

Observa que la notación decimal del instrumento no puede ser usada junto con MIDI en tiempo real. En este caso, el instrumento sería monofónico cuando una nota fuera mantenida.

Las notas que son ligadas a otras precedentes en el mismo instrumento, deberían evitar su inicialización mediante el opcode **tigoto**, excepto para los valores que se proporcionan en la partitura. Por ejemplo, deberían saltarse normalmente todos los opcodes de lectura en tabla del instrumento, ya que ellos almacenan su fase internamente. Si se cambia esto súbitamente habrá clicks audibles en la salida.

Observa también que muchos opcodes (como **delay** o **reverb**) poseen la característica de ser inicializados opcionalmente. Para usar dicha opción, puedes valerte del indicador **tival**. Por lo tanto, no necesitan evitarse mediante un salto **tigoto**.

67.8.4 EJEMPLO

Aquí tenemos un instrumento que puede averiguar si hay ligadura con la nota previa (en cuyo caso **tival** devuelve 1), o si es una nota mantenida (p3 negativo). El ataque y la caída son configurados de acuerdo a ello.

```
instr 10
  icps      init      cpspch(p4) ; altura final desde la partitura
  iportime  init      abs(p3)/7 ; tiempo de Portamento (depende de la
                                ; duración de la nota)
  iamp0     init      p5          ; Amplitudes pòr defecto
  iamp1     init      p5
  iamp2     init      p5
  itie      tival          ; Comprueba si la nota está ligada
  if      itie == 1 igoto nofadein
                                ; si no fade in
  iamp0     init      0
  nofadein:
  if      p3 < 0 igoto nofadeout
                                ; Comprueba si la nota está mantenida
                                ; si no fade out
  iamp2     init      0
  nofadeout:
  kamp     linseg      iamp0, .03, iamp1, abs(p3)-.03, iamp2
                                ; Aplica el envolvente de amplitud
                                ; se salta el resto de la inicialización
                                ; de las notas ligadas
  tieskip igoto      tieskip
  kcps      init      icps          ; inicialización de la altura para notas
                                ; no ligadas
  kcps      port      icps, iportime, icps ; portamento hacia la altura final
  kpw       oscil      .4, rnd(1), 1, rnd(.7) ; oscilador simple
  ar        vco      kamp, kcps, 3, kpw+.5, 1, 1/icps
                                ; (Usado para probar - se puede hacer ipch
                                ; cpspch(p4+2)
                                ; y ver el espectro de salida)
                                ; ar oscil kamp, kcps, 1
  tieskip: out      ar
                                ; Se salta la inicialización de las notas
                                ; ligadas
endin
```

Una partitura simple que usa tres activaciones del instrumento anterior:

```
f1 0 8192 10 1 ; Seno
i10.1 0 -1 7.00 10000
i10.2 0 -1 7.04
i10.3 0 -1 7.07
i10.1 1 -1 8.00
i10.2 1 -1 8.04
i10.3 1 -1 8.07
i10.1 2 1 7.11
i10.2 2 1 8.04
i10.3 2 1 8.07
e
```

Texto adicional (versión 4.0) explicando el uso de notas ligadas, editado por Rasmus Ekman según una nota de David Kirsh, aparecida en la lista de correo de Csound. Ejemplo por Rasmus Ekman.

67.9 SENTENCIA a (O SENTENCIA DE AVANCE)

a p1 p2 p3

67.9.1 DESCRIPCIÓN

Esta sentencia provoca que el puntero temporal de la partitura avance una duración especificada sin producir muestras de sonido.

67.9.2 CAMPOS P

p1	sin significado. Normalmente
p2	instante de activación, en pulsos, en el que empezará el avance del contador de pulsos
p3	duración del avance (en pulsos)
p4	
p5	sin significado
:	
:	

67.9.3 CONSIDERACIONES ESPECIALES

Esta sentencia permite que el contador de pulsos de la partitura avance sin generar muestras de sonido. Esto puede ser de ayuda cuando una sección está incompleta (por ejemplo, le falta el principio o la parte media) y el usuario no quiere generar y escuchar un montón de silencio. Los campos p2 y p3 se tratan como en las **sentencias i**, en lo que se refiere a la ordenación y modificación mediante **sentencias t**.

Una **sentencia a** será temporalmente insertada en la partitura por **Score Extract** cuando los segmentos extraídos empiecen después del comienzo de la sección. El propósito de esto es preservar la cuenta de pulsos y de tiempo real de la partitura original, para no confundir los mensajes de los valores de pico de amplitud que se presentan en la consola del usuario.

Siempre que la orquesta que está ejecutándose encuentra una **sentencia a**, su presencia y efecto se presentarán en pantalla.

67.10 SENTENCIAS t (SENTENCIAS DE TIEMPO)

t p1 p2 p3 p4 (ilimitado)

67.10.1 DESCRIPCIÓN

Esta sentencia ajusta el tiempo y especifica las aceleraciones y desaceleraciones de la sección actual. Esto se realiza convirtiendo los pulsos en segundos.

67.10.2 CAMPOS P

p1 debe ser 0

p2 tiempo inicial en pulsos por minuto

p3, p5, p7, ... instantes en los que se produce el cambio de tiempo, en pulsos y en orden creciente

p4, p6, p8, ... tempi aplicados en los instantes anteriores.

67.10.3 CONSIDERACIONES ESPECIALES

Los instantes en los que tendrán lugar los cambios de tiempo y los tempi mismos se presentan en parejas que definen un gráfico "tiempo vs instante" (el eje con los instantes está medido en pulsos y no tiene porqué ser lineal). La velocidad de pulso de una sección puede imaginarse como un movimiento punto a punto en ese gráfico. Los movimientos entre dos puntos de igual altura indican un tempo constante, mientras que aquellos entre puntos de distinta altura representan acelerandos o ritardandos. El gráfico puede contener discontinuidades: dos puntos en el mismo instante pero con tempos diferentes provocarán un inmediato cambio de tempo.

El movimiento entre tempi diferentes, en un instante mayor que 0, es inversamente lineal. Es decir, un acelerando entre dos tempi M1 y M2 se efectúa por interpolación lineal de las duraciones de los pulsos individuales de $60/M1$ a $60/M2$.

El primer tempo especificado debe ser para el instante 0 (primer pulso).

Un determinado tempo, una vez asignado, permanecerá constante a partir ese instante, a menos que sea afectado por otro cambio de tempo posterior. Es decir, el último tempo especificado durará hasta el final de la sección.

Una **sentencia t** se aplica sólo a la sección en la que aparece. Sólo una **sentencia t** es significativa en una sección. Puede ser colocada en cualquier lugar dentro de esa sección. Si no contiene ninguna **sentencia t**, la sección interpretará los pulsos como segundos (es decir, con un tempo implícito de **t 0 60**).

N.B: si la línea de comando de Cosund incluye un **indicador -t**, los tempi indicados por todas las **sentencias t** de la partitura serán sustituidos por el que especifica dicho indicador.

67.11 SENTENCIA b

b p1

67.11.1 DESCRIPCIÓN

Esta sentencia reinicializa el reloj para las siguientes **sentencias i**.

67.11.2 CAMPOS P

p1 especifica cómo se reinicilizará el reloj.

67.11.3 CONSIDERACIONES ESPECIALES

p1 es el número de pulsos por los cuales los valores de p2 de las siguientes **sentencias i** serán modificados. Si p1 es positivo, el reloj es reinicializado hacia delante, y las notas siguientes aparecerán más tarde, según el número de pulsos especificado por p1 al añadirse al p2 de cada nota. Si p1 es negativo, el reloj es reinicializado hacia atrás, y las siguientes notas aparecerán antes, según el número de pulsos especificado por p1 menos el p2 de cada nota. No hay efecto de acumulación. El reloj es reseteado con cada **sentencia b**. Si p1 =0, el reloj es puesto en su posición original, y las siguientes notas aparecerán en su p2 especificado.

67.11.4 EJEMPLO

```
i1 0 2
i1 10 888

b 5 ; adelanta el reloj
i2 1 1 440 ; isntante de comienzo = 6
i2 2 1 480 ; isntante de comienzo = 7

b -1 ; atrasa el reloj
i3 3 2 3.1415 ; isntante de comienzo = 2
i3 5.5 1 1.1111 ; isntante de comienzo = 4.5

b 0 ; pone el reloj normal
i4 10 200 7 ; isntante de comienzo = 10
```

Explicación sugerida y ejemplo proporcionado por Paul Winkler. (Csound, Versión 4.0)

67.12 SENTENCIA v

v p1

67.12.1 DESCRIPCIÓN

Las **sentencias v** proporcionan deformaciones temporales localmente variables de los eventos de la partitura.

67.12.2 CAMPOS P

p1 factor de deformación temporal (debe ser positivo)

67.12.3 CONSIDERACIONES ESPECIALES

La **sentencia v** tiene efecto con la siguiente **sentencia i** que aparezca, y permanece activa hasta la próxima **sentencia s, v, o e**.

El valor de p1 se usa como un factor múltiplo para los instantes (p2) de activación de evento de las siguientes **sentencias i**.

```
i1 0 1 ;nota1
v 2
i1 1 1 ;nota2
```

En este ejemplo, la segunda nota ocurre dos pulsos después de la primera nota, y es dos veces más larga. Aunque la **sentencia v** es similar a la **sentencia t**, la primera opera localmente. Es decir, **v** afecta sólo a las siguientes notas y su efecto puede ser cancelado o cambiado por otra **sentencia v**.

Los valores que se mueven con **Carry** (ver la **Sección 14.1.1**) no se ven afectados por la **sentencia v**.

```
i1 0 1 ;nota1
v2
i1 1 . ;nota2
i1 2 . ;nota3
v1
i1 3 . ;nota4
i1 4 . ;nota5
e
```

En este ejemplo, la nota2 y la nota4 ocurren simultáneamente, mientras que la nota3 tiene lugar realmente antes de la nota2, es decir en su lugar original. Las duraciones no se ven afectadas.

```
i1 0 1
v2
i. + .
i. . .
e
```

En este caso, **la sentencia v** no tiene ningún efecto.

67.13 SENTENCIA S

s cualquier cosa

67.13.1 DESCRIPCIÓN

Indica el final de una sección.

67.13.2 CAMPOS P

Son todos ignorados

67.13.3 CONSIDERACIONES ESPECIALES

La ordenación de las **sentencias i, f y a** según sus instantes de activación (p2) se realiza sección a sección.

Las alteraciones de tempo de la **sentencia t** se realizan sección a sección.

Todos los instantes de activación (p2s) dentro de una sección son relativos a su comienzo. Una sentencia de sección establece un nuevo instante 0 relativo, pero no tiene ningún otro efecto de reinicialización (por ejemplo, las tablas de función almacenadas se conservan más allá de los límites de la sección).

Una sección se considera completa cuando todos los instantes de activación (p2s) y duraciones finitas (p3s) han sido ejecutadas (es decir, el fin de la longitud de una sección viene determinado por la última acción o desactivación que se produzca). Una sección puede ser prolongada por el uso de una sentencia **f0**.

Un final de sección invoca automáticamente una purga de los instrumentos inactivos y el espacio de datos.

N.B: debido a que las sentencias de partitura se procesan sección a sección, la cantidad de memoria requerida depende del máximo número de sentencias de partitura en una determinada sección. La asignación de memoria es dinámica, y el usuario será informado si se requieren más bloques de memoria extra durante el procesado.

Para el final de la última sección de la partitura la **sentencia s** es opcional, ya que se puede usar la sentencia **e** en su lugar.

67.14 SENTENCIA e

e cualquier cosa

67.14.1 DESCRIPCIÓN

Esta sentencia puede ser usada para indicar el final de la última sección de la partitura.

67.14.2 CAMPOS P

Todos los campos p son ignorados.

67.14.3 CONSIDERACIONES ESPECIALES

La **sentencia e** es contextualmente idéntica a una **sentencia s**. Además, la **sentencia e** termina cualquier generación de señal (incluida la ejecución indefinida) y cierra todos los ficheros de entrada y salida.

Si una **sentencia e** tiene lugar antes del final de una partitura, todas las líneas siguientes serán ignoradas.

La **sentencia e** es opcional en un fichero de partitura que no se ha ordenado aún. Si una partitura no tiene **sentencia e**, el algoritmo de ordenación le dará una.

67.15 SENTENCIA r (SENTENCIA DE REPETICIÓN)

`r p1 p2`

67.15.1 DESCRIPCIÓN

Empieza una sección repetida, que dura hasta la siguiente **sentencia s, r o e**.

67.15.2 CAMPOS P

p1 Número de veces que se repite la sección.
p2 Macro (nombre) que incrementar con cada repetición.

67.15.3 CONSIDERACIONES ESPECIALES

Para que las secciones sean más flexibles y simples de editar, la macro llamada p2 toma un valor de 1 para la primera pasada de la sección, 2 para la segunda, 3 para la tercera, ect. Esto puede usarse para cambiar los parámetros de los campos-p, o ser simplemente ignorado.

ATENCIÓN: A causa de serios problemas de interacción con el sistema de macros, las secciones deben empezar y terminar en el mismo fichero y no en una macro.

67.15.4 EJEMPLO

En el siguiente ejemplo, la sección es repetida 3 veces. La macro NN que se usa es incrementada en cada repetición.

```
r3 NN ;principio de la sección repetida - usa la macro NN
algún código
.
.
.
s ;fin de la sección repetida - vuelve al r previo si las repeticiones son menos de 3
```

67.15.5 AUTOR

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Abril, 1998 (Nuevo en la versión 3.48)

67.16 SENTENCIA m (SENTENCIA DE MARCA)

m p1

67.16.1 DESCRIPCIÓN

Coloca una marca en la partitura, asignándole un nombre que puede ser referenciado por cualquier **sentencia n**.

67.16.2 CAMPOS P

p1 Nombre de la marca

67.16.3 CONSIDERACIONES ESPECIALES

Esta sentencia puede ser de ayuda para construir una estructura de estrofa y estribillo en la partitura. Los nombres pueden contener cualquier combinación de letras y números.

67.16.4 AUTOR

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Abril, 1998 (Nuevo en la versión 3.48)

67.17 SENTENCIA n

n p1

67.17.1 DESCRIPCIÓN

Repite una sección desde la **sentencia m** referenciada.

67.17.2 CAMPOS P

p1 Nombre de la marca desde la que repetir

67.17.3 CONSIDERACIONES ESPECIALES

Esta sentencia puede ser de ayuda para construir una estructura de estrofa y estribillo en la partitura. Los nombres pueden contener cualquier combinación de letras y números.

67.17.4 AUTOR

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Abril, 1998 (Nuevo en la versión 3.48)

68 RUTINAS GEN

Las subrutinas GEN son procedimientos de dibujo de funciones llamados por las **sentencias f** para construir y almacenar tablas de función. Están disponibles a lo largo de toda la ejecución de la orquesta y pueden ser invocadas en cualquier instante de la partitura indicado por p2. El campo p1 asigna un número a la tabla, y p3 su tamaño (ver **sentencias f**). p4 especifica la rutina GEN que va a ser llamada. Cada rutina GEN asignará un significado especial a los valores de los campos siguientes.

68.1 GEN01

```
f # time size 1 filcod skiptime format channel
```

68.1.1 DESCRIPCIÓN

Esta subrutina transfiere los datos de un fichero de sonido a una tabla de función.

68.1.2 INICIALIZACIÓN

size - número de puntos de la tabla. Normalmente es una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**); el tamaño máximo es de 16777216 (2^{24}) puntos. Si el fichero de sonido está en formato AIFF, la asignación de la memoria de la tabla puede ser suspendida poniendo este parámetro a 0; el tamaño asignado es el número de puntos del fichero (probablemente no una potencia de 2) y la tabla no será legible para los osciladores normales, aunque sí para el opcode **loscil**. Una fuente AIFF puede ser también mono o estéreo.

filcod - entero o cadena de caracteres que indica el nombre del fichero de sonido fuente. Un entero indica el fichero **soundin.filcod**; una cadena de caracteres (entre comillas, espacios permitidos) proporciona el nombre mismo, opcionalmente con la ruta completa. Si no se indica ésta, el fichero se busca primero en el directorio actual, luego en el que indica la variable de entorno SSDIR (si está definida) y luego en el indicado por SFDIR. Ver también **soundin**.

skiptime - empieza a leer en el fichero a partir de *skiptime* segundos.

channel - número de canal a leer. 0 indica que se lean todos los canales.

format - especifica el formato de los datos del fichero de sonido:

1 - 8-bit signed character	4 - 16-bit short integers
2 - 8-bit A-law bytes	5 - 32-bit long integers
3 - 8-bit U-law bytes	6 - 32-bit floats

Si *format* = 0, el formato de muestreo se toma de la cabecera del fichero de sonido, o por defecto del indicador **-o** de la línea de comando de Csound.

68.1.3 NOTA

La lectura termina al final del fichero o cuando la tabla está llena. Las posiciones de la tabla que no se utilicen contendrán ceros. Si *p4* es positivo, la tabla será postnormalizada (es decir, reescalada a un valor máximo absoluto de 1 después de su generación). Un valor de *p4* negativo causará la omisión del proceso de escala.

68.1.4 EJEMPLOS

```
f 1 0 8192 1 23 0 4
f 2 0 0 -1 "trumpet A#5" 0 4
```

En este ejemplo las tablas se llenan desde dos ficheros, "soundin.23" y "trumpet A#5", esperados en SSDIR o SFDIR. La primera tabla es preasignada; la segunda se asigna dinámicamente en memoria, y no se reescala.

68.2 GEN02

```
f # time size 2 v1 v2 v3 . . .
```

68.2.1 DESCRIPCIÓN

Esta subrutina transfiere los datos de los campos *p* que siguen a una tabla de función.

68.2.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**). El tamaño máximo de la tabla es 16777216 (2^{24}) puntos.

v1, *v2*, *v3*, ... - valores a copiar directamente en la tabla. El número de valores está limitado por la variable de compilación PMAX, que controla el máximo número de campos *p* (actualmente 150). Los valores copiados pueden incluir un elemento límite añadido. Cualquier posición en la tabla no usada contiene ceros.

68.2.3 NOTA

Si *p4* es positivo, la tabla será postnormalizada (es decir, reescalada a un valor máximo absoluto de 1 después de su generación). Un valor de *p4* negativo causará la omisión del proceso de escala.

68.2.4 EJEMPLO

```
f 1 0 16 -2 0 1 2 3 4 5 6 7 8 9 10 11 0
```

Esto ordena a **GEN02** que coloque 12 valores, más un elemento límite añadido explícito, en una tabla de tamaño igual a la potencia de 2 inmediatamente superior. No se reescala.

68.3 GEN03

```
f # time size 3 xval1 xval2 c0 c1 c2 . . . cn
```

68.3.1 DESCRIPCIÓN

Esta subrutina genera una tabla de función calculando un polinomio en x sobre un intervalo fijo con coeficientes especificados.

68.3.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**). El tamaño máximo de la tabla es 16777216 (2^{24}) puntos.

xval1, *xval2* - valores izquierdo y derecho de x con los que se define el intervalo ($xval1 < xval2$). Estos darán lugar en la generación al primer valor almacenado en la tabla y al último respectivamente (potencia de 2 más uno).

c0, *c1*, *c2*, ... *cn* - coeficientes del polinomio de orden n .
 $c0 + c1x + c2x^2 + \dots + cnx^n$

Los coeficientes pueden ser números reales positivos o negativos; un valor 0 indica un término ausente en el polinomio. La lista de coeficientes empieza en *p7*, con un límite superior de 144 términos.

68.3.3 NOTA

El segmento definido [$fn(xval1),fn(xval2)$] se distribuye uniformemente. Así una tabla de 512 puntos en el intervalo $[-1,1]$ tendrá su origen en la posición 257 (al principio de la segunda mitad). Al requerirse un elemento límite añadido, ambos puntos $fn(-1)$ y $fn(1)$ existirán en la tabla.

GEN03 es útil en conjunción a **table** o **tablei** para la síntesis de modelado de ondas (modificación del sonido por procesos de distorsión no lineal). Los coeficientes que van a producir un formante particular, mediante un índice de lectura sinusoidal de amplitud conocida, pueden ser calculados antes del procesado usando algoritmos como las fórmulas de Chebyshev. Ver también **GEN13**.

68.3.4 EJEMPLO

```
f 1 0 1025 3 -1 1 5 4 3 2 2 1
```

Esto llama a la subrutina **GEN03** para llenar una tabla con una función polinómica de cuarto orden en el intervalo de x $[-1,1]$. El origen estará en la posición 512. La función es postnormalizada.

68.4 GEN04

```
f # time size 4 source# sourcemode
```

68.4.1 DESCRIPCIÓN

Esta subrutina genera una función normalizada examinando los contenidos de una tabla existente.

68.4.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 más 1. No debe exceder (excepto en 1) el tamaño de la tabla que está siendo examinada. Está limitado a justo la mitad del tamaño si *sourcemode* es del tipo offset (ver abajo).

source # - número de la tabla de función almacenada a ser examinada.

sourcemode - un valor codificado que especifica cómo va a ser escaneada la tabla fuente para obtener la función normalizadora. Un valor 0 indica que la fuente va a ser escaneada de izquierda a derecha. Un valor distinto a 0 indica que la fuente tiene una estructura bipolar; entonces el escaneado empezará en el punto medio y proseguirá hacia afuera, buscando pares de puntos equidistantes del centro.

68.4.3 NOTA

La función normalizadora se deriva de los máximos absolutos obtenidos progresivamente al escanear la tabla. La nueva tabla se crea de izquierda a derecha, con valores iguales a $1/(\text{máximo absoluto escaneado hasta el momento})$. Los valores almacenados empezarán con $1/(\text{primer valor escaneado})$, luego se irá haciendo progresivamente más pequeño a medida que los nuevos máximos son encontrados. Para una tabla fuente ya normalizada (es decir, cuyos valores son iguales o menores que 1), los valores derivados estarán en el rango de $1/(\text{primer valor escaneado})$ a 1. Si el primer valor escaneado es 0, ese inverso será puesto a uno.

La función normalizadora de **GEN04** no está normalizada en sí misma.

GEN04 es útil para escalar la señal derivada de una tabla para que tenga un pico de amplitud consistente. Una aplicación particular es cuando, en la síntesis por modelado de onda, la señal portadora (o índice) es menor que la amplitud total.

68.4.4 EJEMPLO

```
f 2 0 512 4 1 1
```

Esto crea una función normalizadora para usar junto con la tabla 1 del ejemplo de **GEN03**. Se especifica el offset que indica el punto medio bipolar.

68.5 GEN05, GEN07

```
f # time size 5 a n1 b n2 c . . .  
f # time size 7 a n1 b n2 c . . .
```

68.5.1 DESCRIPCIÓN

Estas subrutinas se usan para construir funciones a base de segmentos de curvas exponenciales (**GEN05**) o líneas rectas (**GEN07**).

68.5.2 INICIALIZACIÓN

size - número de puntos en la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**)

a, *b*, *c*, etc. - valores de las ordenadas, en los campo *p* impares (*p*5, *p*7, *p*9 ...) Para **GEN05** deben ser distintos de 0 e iguales en signo. Para **GEN07** no existen tales restricciones.

*n*1, *n*2, etc. - longitud del segmento (número de posiciones de almacenamiento), en los campos *p* pares. No pueden ser negativos, pero un valor 0 se usa para especificar formas de onda discontinuas (por ejemplo la función de abajo). La suma *n*1 + *n*2 +... será normalmente igual al tamaño *size* para las funciones especificadas en su totalidad. Si la suma es menor, las posiciones de la función no incluidas serán puestas a 0; si la suma es mayor, sólo serán almacenadas las primeras *size* posiciones.

68.5.3 NOTA

Si *p*4 es positivo, las funciones son postnormalizadas (es decir, reescaladas a un valor máximo absoluto de 1 después de su generación). Un valor de *p*4 negativo causará que se omita el proceso de escala.

La interpolación lineal entre dos puntos discretos implica un descenso o aumento por un mismo valor a lo largo de un segmento entre posiciones adyacentes. La interpolación exponencial implica que la progresión se realiza mediante un mismo cociente. En ambas formas la interpolación de *a* a *b* asume que el valor *b* será alcanzado en la posición *n*+1. Para funciones discontinuas, y para el segmento que abarca la posición final, este valor no será alcanzado de verdad, aunque puede aparecer finalmente como resultado del proceso de escala final.

68.5.4 EJEMPLO

```
f 1 0 256 7 0 128 1 0 -1 128 0
```

Esto describe un ciclo de una onda diente de sierra cuya discontinuidad está en el punto medio de la función almacenada.

68.6 GEN06

```
f # time size 6 a n1 b n2 c n3 d . . .
```

68.6.1 DESCRIPCIÓN

Esta subrutina generará una función compuesta por segmentos de polinomios cúbicos, abarcando sólo tres puntos dados a la vez.

68.6.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**).

a, c, e, ... - máximos o mínimos locales de los sucesivos segmentos, dependiendo de la relación de estos puntos con las inflexiones adyacentes. Pueden ser tanto positivos como negativos.

b, d, f, ... - valores de ordenada de los puntos de inflexión al final de cada sucesivo segmento de la curva. Pueden ser positivos o negativos.

n1, n2, n3... - número de valores almacenados entre los puntos especificados. No pueden ser negativos, pero un valor 0 se usa para indicar discontinuidades. La suma $n1 + n2 + \dots$ será normalmente igual al tamaño total de la tabla de la función (para los detalles, ver **GEN05**).

68.6.3 NOTA

GEN06 construye y almacena una función hecha mediante segmentos de funciones polinómicas cúbicas. Dichos segmentos unen los valores de las ordenadas en grupos de 3: punto de inflexión, máximo/mínimo, punto de inflexión. El primer segmento completo abarca b,c,d y tiene una longitud $n2 + n3$, el siguiente abarca d,e,f y tiene una longitud $n4 + n5$, etc. El primer segmento (a,b de longitud $n1$) es parcial y sólo tiene una inflexión. El último segmento puede ser también parcial. Aunque los puntos de inflexión b,d,f ... figuran en dos segmentos (a la izquierda y la derecha), la pendiente de los dos segmentos es independiente en ese punto común (es decir, la primera derivada será probablemente discontinua). Cuando a,c,e... son alternativamente máximos y mínimos, las uniones de las inflexiones serán relativamente suaves; cuando son siempre máximos o mínimos las inflexiones tendrán forma de peine.

68.6.4 EJEMPLO

```
f 1 0 65 6 0 16 .5 16 1 16 0 16 -1
```

Esto crea una curva de 0 a 1 y a -1, con un mínimo, máximo y mínimo en esos valores respectivamente. Las inflexiones están en .5 y 0 y son relativamente suaves.

68.7 GEN08

```
f # time size 8 a n1 b n2 c n3 d . . .
```

68.7.1 DESCRIPCIÓN

Esta subrutina generará una curva cúbica *spline*, la más suave posible que pase por todos los puntos especificados.

68.7.2 INICIALIZACIÓN

size - número de puntos en la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**).

a, b, c ... - valores de ordenada de la función.

n1, n2, n3 ... - longitud de cada segmento, medido en número de valores almacenados. No puede ser 0, pero puede ser un valor fraccionario. Un segmento determinado podría o podría no almacenar valores en realidad; los valores almacenados serán calculados en puntos enteros de la función, desde el principio de ésta. La suma $n1 + n2 + \dots$ será normalmente igual al tamaño total de la tabla de la función (para los detalles, ver **GEN05**).

68.7.3 NOTA

GEN08 construye y almacena una tabla de segmentos de funciones polinómicas cúbicas. Cada segmento se calcula entre dos puntos especificados pero depende también de sus vecinos a cada lado.

Los segmentos vecinos deben concordar tanto en valor como en pendiente en un punto común (la pendiente común es la de la parábola que pasa por ese punto y sus dos vecinos). La pendiente en los extremos de la función debe ser 0 (es decir, plana).

Consejo: Para indicar discontinuidades en la pendiente o en el valor de una función almacenada, organiza una serie de puntos en el intervalo entre dos valores almacenados; de igual manera para un límite de pendiente distinto a 0.

68.7.4 EJEMPLO

```
f 1 0 65 8 0 16 0 16 1 16 0 16 0
```

Este ejemplo crea una curva con una joroba suave en el medio, yendo brevemente hacia los valores negativos fuera de la joroba y aplanándose al final.

```
f 2 0 65 8 0 16 0 .1 0 15.9 1 15.9  
0 .1 0 16 0
```

Este ejemplo es similar pero no pasa por valores negativos.

68.8 GEN09, GEN10, GEN19

```
f # time size 9 pna stra phsa pnb strb phsb . . .  
f # time size 10 str1 str2 str3 str4 . . . .  
f # time size 19 pna stra phsa dcoa pnb strb phsb dcob . . .
```

68.8.1 DESCRIPCIÓN

Estas subrutinas generan formas de onda compuestas de la suma ponderada de ondas sinusoidales simples. La especificación de cada parcial requiere 3 campos *p* usando **GEN09**, 1 usando **GEN10** y 4 usando **GEN19**.

68.8.2 INICIALIZACIÓN

size - número de puntos en la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**).

pna, *pnb*, etc. - número de parcial (en relación a la fundamental que ocuparía *size* posiciones por ciclo) de una onda sinusoidal a, otra b, etc. Deben ser positivos, pero no tienen por qué ser números enteros, es decir, pueden ser parciales inarmónicos. Los parciales pueden estar en cualquier orden.

stra, *strb*, etc. - fuerza de los parciales *pna*, *pnb*, etc. Son fuerzas relativas, ya que la forma de onda compuesta puede ser reescalada después. Están permitidos los valores negativos e implican un desfase de 180 grados.

phsa, *phsb*, etc. - fase inicial de los parciales *pna*, *pnb*, etc., expresadas en grados.

dcoa, *dcob*, etc. - offset DC de los parciales *pna*, *pnb*, etc. que será aplicado después de escalar las amplitudes, es decir un valor de 2 desplazará hacia arriba una onda sinusoidal desde el rango [-2,2] hasta el rango [0,4].

str1, *str2*, *str3*, etc. - fuerza relativa de los parciales armónicos fijos números 1,2,3 etc., empezando en *p5*. Los parciales no deseados pueden ser eliminados dándole un valor 0.

68.8.3 NOTA

Estas subrutinas generan funciones como la suma de ondas sinusoidales de diferentes frecuencias. Las dos mayores restricciones de **GEN10** (los parciales deben ser armónicos y estar en fase) no se aplican a **GEN09** o **GEN19**.

En cada caso la onda compuesta, una vez dibujada, es normalizada si *p4* era positivo. Un valor negativo de *p4* causará que se omita la normalización.

68.8.4 EJEMPLO

```
f 1 0 1024 9 1 3 0 3 1 0 9 .3333  
f 2 0 1024 19 .5 1 270 1
```

f1 mezcla los parciales 1,3 y 9 con las amplitudes típicas de una onda cuadrada, excepto en que el parcial 9 está invertido. f2 crea un sigmoide ascendente [0-2]. Ambos serán reescalados.

68.9 GEN11

```
f # time size ll nh lh r
```

68.9.1 DESCRIPCIÓN

Esta subrutina genera una serie aditiva de parciales coseno, en la manera de los generados **buzz** y **gbuzz**.

68.9.2 INICIALIZACIÓN

size - número de puntos en la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**).

nh - número de armónicos deseados. Debe ser positivo.

lh (opcional) - armónico más bajo presente. Puede ser positivo, 0 o negativo. La serie de parciales puede empezar en cualquier número y proceder hacia arriba. Si *lh* es negativo, todos los parciales por debajo de 0 se reflejarán en la parte positiva para producir parciales positivos sin cambio de fase (ya que la función coseno es una función uniforme), y los añadirá a todos los demás parciales positivos de la serie. El valor por defecto es 1.

r (opcional) - multiplicador en una serie de coeficientes de amplitud. Esta serie es exponencial: si el *lh*ésimo parcial tiene un coeficiente de amplitud *A*, el (*lh* + *n*)ésimo parcial tendrá un coeficiente "*A* * *r*^{*n*}", es decir, los valores de amplitud dibujarán una curva exponencial. *r* puede ser positivo, 0 o negativo, y no está restringido a números enteros. El valor por defecto es 1.

68.9.3 NOTA

Esta subrutina es una versión estática (sin evolución temporal) de los generadores **buzz** y **gbuzz** de Csound y es igualmente útil como sonido fuente complejo en síntesis substractiva. Con *lh* y *r* presentes es igual que **gbuzz**. Con ambos ausente o iguales a 1 se reduce al más simple **buzz** (es decir, *nh* parciales armónicos de igual fuerza empezando por la fundamental).

Muestrear la forma de onda almacenada con un oscilador es más eficiente que usar unidades dinámicas **buzz**. Sin embargo, el contenido espectral es constante, y se necesita cuidado para que los parciales más altos no excedan la frecuencia Nyquist durante el muestreo y produzcan un efecto de foldover.

68.9.4 EJEMPLOS

```
f 1 0 2049 11 4  
f 2 0 2049 11 4 1 1  
f 3 0 2049 -11 7 3 .5
```

Las dos primeras tablas contendrán ondas pulso de bandas limitadas idénticas, cada una con 4 armónicos de igual fuerza empezando por la fundamental. La tercera tabla sumará 7 armónicos consecutivos, empezando por el tercero y con amplitudes progresivamente más débiles (1, .5, .25, .125 ...). No será postnormalizada.

68.10 GEN12

```
f # time size -12 xint
```

68.10.1 DESCRIPCIÓN

Calcula el logaritmo de una función de Bessel modificada de segunda clase y orden 0, útil para usar en síntesis FM con amplitud modulada.

68.10.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**). El valor normal es una potencia de dos más 1.

xint - especifica el intervalo en x [*0 to +int*] en el que será definida la función.

68.10.3 NOTA

esta subrutina dibuja el logaritmo natural de una función de Bessel de segunda clase y orden 0 (normalmente representada como I con subíndice 0), en el intervalo en x deseado. La llamada debe tener omitida la opción de reescala.

La función es útil como factor de escala de amplitud en síntesis FM de amplitud modulada síncrona. (Ver Palamin & Palamin, *J. Audio Eng. Soc.*, 36/9, Sept. 1988, pp.671-684.). El algoritmo es interesante porque permite que el espectro normalmente simétrico de la síntesis FM se convierta en asimétrico alrededor de una frecuencia distinta a la portadora, y es, por tanto, útil para posicionar formantes. Usando un índice de búsqueda en tabla igual a $I(r - 1/r)$, donde I es el índice de modulación FM y r es un parámetro exponencial que afecta a la amplitud de los parciales, el algoritmo de Palamin es relativamente eficiente, requiriendo sólo osciladores con búsqueda en tabla y una única llamada exponencial.

68.10.4 EJEMPLO

```
f 1 0 2049 -12 20
```

Esto dibuja una $\ln(I_0(x))$ sin normalizar entre 0 y 20.

68.11 GEN13, GEN14

```
f # time size 13 xint xamp h0 h1 h2 . . . hn
f # time size 14 xint xamp h0 h1 h2 . . . hn
```

68.11.1 DESCRIPCIÓN

Estas subrutinas usan los coeficientes de Chebyshev para calcular funciones polinómicas almacenadas que, en síntesis por modelado de onda, pueden ser usadas para dividir una onda sinusoidal en parciales armónicos, consiguiendo un espectro predefinido.

68.11.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**). El valor normal es una potencia de 2 más 1.-

xint - proporciona los valores izquierdo y derecho $[-xint, +xint]$ del intervalo en x en el que se va a dibujar el polinomio. Estas subrutinas llaman a **GEN03** para dibujar sus funciones. El valor de *p5* aquí es por tanto expandido a un par (*p5*,*p6*) negativo-positivo antes de llamar realmente a **GEN03**. El valor normal es 1.

xamp - factor de escala de amplitud de la entrada sinusoidal que se espera para producir el siguiente espectro.

h0, *h1*, *h2*, ..., *hn* - amplitudes relativas de los parciales 0 (DC), 1 (fundamental), 2... que resultarán cuando una onda sinusoidal de amplitud $xamp * \sin(\pi * size/2 * xint)$ es modelada usando esta tabla de función. Estos valores describen así un espectro de frecuencias asociado con un determinado valor del factor *xamp* de la señal de entrada.

68.11.3 NOTA:

GEN13 es el generador de función empleado normalmente en la síntesis de modelado de onda estándar. Guarda un polinomio cuyos coeficientes se derivan de los polinomios de primera clase de Chebyshev, para que la onda sinusoidal conductora de amplitud *xamp* muestre el espectro especificado en la salida. Observa que la evolución de este espectro es generalmente no lineal, con un factor *xamp* variable. Sin embargo, es de banda limitada (los únicos parciales que aparecen serán aquellos especificados en el momento del cálculo) y los parciales tenderán a aparecer y desarrollarse en orden ascendente (dominando los parciales más bajos con valores bajos de *xamp* y enriqueciéndose el espectro conforme va creciendo el valor de *xamp*). Un valor negativo de *hn* implica un desfase de 180 grados de ese parcial. El espectro con la amplitud total deseada no se verá afectado por este desfase, aunque si puede verse afectada la evolución de algunos de sus parciales constitutivos. El patrón $+,+,-,-,+,+,...$ para *h0*,*h1*,*h2*... minimizará el problema de normalización para valores bajos de *xamp* (ver arriba), pero no proporciona necesariamente el patrón de evolución más suave.

GEN14 guarda un polinomio cuyos coeficientes se derivan de los de segunda clase de Chebyshev.

68.11.4 EJEMPLO

f 1 0 1025 13 1 1 0 5 0 3 0 1

Esto crea una función que, en síntesis de modelado de onda, dividirá una onda sinusoidal en 3 parciales armónicas pares de amplitud relativa 5,3,1.

68.12 GEN15

`f # time size 15 xint xamp h0 phs0 h1 phs1 h2 phs2 . . .`

68.12.1 DESCRIPCIÓN

Esta subrutina crea dos tablas de funciones polinómicas, útiles para usarlas en operaciones de cuadratura de fase.

68.12.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**). El valor normal es una potencia de dos más 1.

xint - proporciona los valores izquierdo y derecho $[-xint, +xint]$ del intervalo en *x* en el que se va a dibujar el polinomio. Esta subrutina llama finalmente a **GEN03** para dibujar ambas funciones. El valor de *p5* aquí es por tanto expandido a un par *p5,p6* negativo-positivo antes de llamar realmente a **GEN03**. El valor normal es 1.

xamp - factor de escala de amplitud de la entrada sinusoidal que se espera para producir el siguiente espectro.

h0, h1, h2, ..., hn - amplitudes relativas de los parciales 0 (DC), 1 (fundamental), 2... que resultarán cuando una onda sinusoidal de amplitud $xamp * \text{int}(size/2)/xint$ es modelada usando esta tabla de función. Estos valores describen así un espectro de frecuencias asociado con un determinado valor del factor *xamp* de la señal de entrada.

phs0, phs1, ... - fases, expresadas en grados, de los armónicos deseados *h0, h1, ...* cuando las dos funciones de **GEN15** son usadas con cuadratura de fase.

68.12.3 NOTA

GEN15 crea dos tablas de igual tamaño, etiquetadas como **f #** y **f # + 1**. La tabla **#** contendrá una función de Chebyshev de primera clase, dibujada usando **GEN03** con parciales de amplitudes $h0\cos(phs0), h1\cos(phs1), \dots$. La tabla **#+1** contendrá una función de Chebyshev de segunda clase llamando a **GEN14** con parciales $h1\sin(phs1), h2\sin(phs2), \dots$ (observa el desplazamiento de los armónicos). Las dos tablas pueden ser usadas junto con una red de modelado de onda que use cuadratura de fase.

68.13 GEN17

```
f # time size 17 x1 a x2 b x3 c . . .
```

68.13.1 DESCRIPCIÓN

Esta subrutina crea una función escalonada tomando parejas xy dadas.

68.13.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**). El valor normal es una potencia de dos más 1.

x1, *x2*, *x3*, etc. - valores de abscisa, en orden ascendente, empezando por 0.

a, *b*, *c*, etc. - valores de las ordenadas en las abscisas especificadas antes, constantes hasta el próximo valor de *x*.

68.13.3 NOTA

Esta subrutina crea una función escalonada de pares xy cuyas ordenadas son constantes hacia la derecha. El valor de *y* más a la derecha se mantiene entonces constante hasta el final de la tabla. La función es útil para mapear una serie de datos en otra tabla, como por ejemplo convertir números de nota MIDI en valores de una tabla de función de muestras de sonido (ver **loscil**).

68.13.4 EJEMPLO

```
f 1 0 128 -17 0 1 12 2 24 3 36 4 48 5 60 6 72 7 84 8
```

Este ejemplo describe una función escalonada con 8 niveles sucesivos ascendentes, cada uno con 12 posiciones de ancho, excepto el último, que prolonga su valor hasta el final de la tabla. La función no se reescala. La indexación de la tabla según un número de nota MIDI devolverá un valor diferente cada octava hasta la 8, a partir de la cual el valor devuelto se mantendrá constante.

68.14 GEN20

```
f # time size 20 window max opt
```

68.14.1 DESCRIPCIÓN

Esta subrutina calcula funciones de diferentes ventanas. Estas ventanas son usadas normalmente para el análisis espectral o para envolventes granulares.

68.14.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 más 1.

window - Tipo de la ventana a generar.

```
1 - Hamming  
2 - Hanning  
3 - Bartlett ( triangular)  
4 - Blackman ( 3 - términos)  
5 - Blackman - Harris ( 4 - términos)  
6 - Gaussiana  
7 - Kaiser  
8 - Rectangular  
9 - Síncrona
```

max - para p_4 negativos será el valor de pico absoluto de la ventana. Si p_4 es positivo, o p_4 negativo y p_6 está ausente, la tabla será postnormalizada (valor máximo de 1).

opt - argumento opcional para la ventana de Kaiser.

68.14.3 EJEMPLOS

```
f 1 0 1024 20 5
```

Esto crea una función que contiene una ventana del tipo Blackman - Harris de cuatro términos con un valor máximo de 1.

```
f 1 0 1024 -20 2 456
```

Esto crea una función que contiene una ventana de Hanning con un valor máximo de 456.

```
f 1 0 1024 -20 1
```

Esto crea una función que contiene una ventana de Hamming con un valor máximo de 1.

```
f 1 0 1024 20 7 1 2
```

Este último ejemplo crea una función que contiene una ventana de Kaiser con un valor máximo de 1. El argumento extra especifica cuán abierta es la ventana; por ejemplo un valor de 0 daría lugar a una ventana rectangular y un valor de 10 una parecida a la de Hamming.

Ver Apéndice para los tipos de ventana.

68.14.4 AUTORES

Paris Smaragdis
MIT, Cambridge
1995

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Nuevo en la versión 3.2

68.15 GEN21

```
f # time size 21 type lvl arg1 arg2
```

68.15.1 DESCRIPCIÓN

Calcula tablas con diferentes distribuciones aleatorias (ver también los generadores de ruido). El tiempo y el tamaño son los argumentos normales de la función **GEN**. El tipo define la distribución a ser usada.

- 1 - Uniforme
- 2 - Lineal
- 3 - Triangular
- 4 - Exponencial
- 5 - Biexponencial
- 6 - Gaussiana
- 7 - Cauchy
- 8 - Cauchy positiva
- 9 - Beta
- 10 - Weibull
- 11 - Poison

De todos estos casos sólo la 9 y la 10 necesitan argumentos extra. La distribución Beta necesita dos y la de Weibull uno.

68.15.2 EJEMPLO

```
f1 0 1024 21 1 ; Uniforme (ruido blanco)
f1 0 1024 21 6 ; Gaussiana
f1 0 1024 21 9 1 1 2 ; Beta (observa que el nivel precede a los argumentos)
f1 0 1024 21 10 1 2 ; Weibull
```

Todas las anteriores adiciones fueron diseñadas por el autor entre Mayo y Diciembre de 1994, bajo la supervisión del Doctor Richard Boulanger.

68.15.3 AUTORES

Paris Smaragdis
MIT, Cambridge
1995

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Nuevo en la versión 3.2

68.16 GEN23

```
f # time size -23 "filename.txt"
```

68.16.1 DESCRIPCIÓN

Esta subrutina lee valores numéricos de un fichero ASCII externo.

68.16.2 INICIALIZACIÓN

"filename.txt" - valores numéricos contenidos en el fichero "filename.txt" (que indica la ruta completa del fichero de caracteres a leer), que pueden estar separados por espacios, tabuladores, caracteres de nueva línea o comas. Las palabras encontradas que no contengan caracteres numéricos serán interpretadas como comentarios e ignoradas.

size - número de elementos de la tabla. Debe ser una potencia de 2, una potencia de dos más 1, o cero. Si *size* es cero, el tamaño de la tabla vendrá determinado por el número de valores numéricos en *filename.txt* (Nuevo en la versión 3.57).

68.16.3 NOTA

Todos los caracteres después de un ; (comentarios) serán ignorados hasta una nueva línea (incluidos los números).

68.16.4 AUTOR

Gabriel Maldonado

Italia

Febrero, 1998

Nuevo en la versión 3.47

68.17 GEN25, GEN27

```
f # time size 25 x1 y1 x2 y2 x3 . . .  
f # time size 27 x1 y1 x2 y2 x3 . . .
```

68.17.1 DESCRIPCIÓN

Estas subrutinas se usan para construir funciones a base de segmentos de curvas exponenciales (**GEN25**) o líneas rectas (**GEN27**), en formato de puntos de inflexión.

68.17.2 INICIALIZACIÓN

size - número de puntos de la tabla. Debe ser una potencia de 2 o una potencia de 2 más 1 (ver **sentencia f**).

x1, *x2*, *x3*, etc. - posiciones de la tabla en las que obtener el siguiente valor de *y*. Debe estar en orden creciente. Si el último valor es menor que el tamaño de la tabla, el resto de las posiciones se pondrán a 0. No puede ser negativo, aunque puede ser 0.

y1, *y2*, *y3*, etc. - valores de los puntos de inflexión obtenidos en la posición especificada por los valores de *x* anteriores. Para **GEN25** estos deben ser distintos de 0 y concordar en signo. Tales restricciones no existen para **GEN27**.

68.17.3 NOTA

Si *p4* es positivo, las funciones son postnormalizadas (es decir, reescaladas a un valor máximo absoluto de 1 después de su generación). Un valor negativo de *p4* causará que la normalización se omita.

68.17.4 EJEMPLO

```
f 1 0 257 27 0 0 100 1 200 -1 256 0
```

Esto describe una función que empieza en 0, sube hasta 1 en la posición 100, cae a -1 en la 200 y luego vuelve a 0 al final de la tabla. La interpolación es lineal.

68.17.5 AUTORES

John ffitch
Universidad de Bath/Codemist Ltd.
Bath, Reino Unido
Nuevo en la versión 3.49

68.18 GEN28

```
f # time size 28 ifilcod
```

68.18.1 DESCRIPCIÓN

Este generador de función lee un fichero de texto que contiene series de tres valores, que representan las coordenadas y un indicador de tiempo que indica cuando será la señal colocada en dicha posición, permitiendo así al usuario definir una trayectoria a lo largo del tiempo. El formato del fichero es el siguiente:

```
time1 X1 Y1
time2 X2 Y2
time3 X3 Y3
```

La configuración de las coordenadas xy en **space** coloca la señal en los altavoces (a) de la siguiente manera: a1 es (-1,1); a2 es (1, 1); a3 es (-1, -1); a4 es (1, -1). Esto asume que un altavoz configurado como a1 está delante a la izquierda, a2 está delante en la derecha, a3 atrás a la izquierda y a4 está atrás a la derecha. Los valores mayores que 1 darán lugar a sonidos atenuados como si estuvieran a distancia.

GEN28 crea valores con 10 milisegundos de resolución.

68.18.2 INICIALIZACIÓN

size – número de puntos de la tabla. Debe ser 0 ya que **GEN28** toma 0 como tamaño y asigna la memoria automáticamente.

ifilcod - cadena de caracteres que indica el nombre de un fichero fuente. Una cadena de caracteres (entre comillas, espacios permitidos) proporciona el nombre del fichero mismo, opcionalmente con la ruta completa. Si no se proporciona ésta, el fichero se busca en el directorio actual.

68.18.3 EJEMPLO

```
f1 0 0 28 "move"
```

El fichero "move" contendrá:

```
0    -1    1
1     1    1
2     4    4
2.1  -4   -4
3     10  -10
5    -40    0
```

Debido a que **GEN28** crea valores de hasta 10 milisegundos de resolución, en este caso habrá 500 valores creados por interpolación, de x1 a x2 a x3 etc. (lo mismo con las ordenadas), con los valores que proporciona la tabla de función. En el ejemplo de arriba, el sonido empezará en la parte delantera izquierda, durante un segundo se moverá la parte delantera derecha, durante otro segundo se alejará pero todavía en la parte delantera izquierda y luego, en sólo una décima de segundo, se moverá a la parte trasera izquierda, un poco distante. Finalmente durante los últimos .9 segundos el sonido se moverá a la parte trasera derecha, a media distancia, viniendo por fin a descansar entre los canales izquierdos, bastante alejado.

68.18.4 AUTOR

Richard Karpen

Seattle, Wash

1998 (Nuevo en la versión 3.48)

69 LA LÍNEA DE COMANDO DE CSOUND

Csound es el comando que se usa para pasar los ficheros de orquesta y partitura a Csound, a fin de que éste genere el fichero de sonido correspondiente. El fichero partitura puede estar en cualquiera de los formatos permitidos, según el deseo del usuario. La traducción, ordenación y formateado del fichero orquesta se lleva a cabo por varios preprocesadores. Entonces se envía todo o parte de la partitura a la orquesta. La ejecución de la orquesta se ve influenciada por los indicadores de la línea de comando, que configuran el nivel de los informes presentados en pantalla, especifican los nombres y los formatos de los ficheros y declaran el tipo de sensores que se van a usar para controlar ejecuciones en tiempo real.

69.1 ORDEN DE PRECEDENCIA

Con algunas recientes mejoras a Csound, hay ahora tres lugares (y en algunos casos cuatro) dónde las opciones para la ejecución de Csound pueden ser configuradas por el usuario. Se procesan en el siguiente orden:

1. Valores por defecto de Csound
2. Fichero `.csoundrc`
3. Línea de comando de Csound
4. La etiqueta `<CsOptions>` en un fichero `.csd`
5. La Cabecera de la Orquesta (para `sr`, `kr`, `ksmps`, `nchnls`)

. La última asignación de una opción sustituirá cualquiera de las anteriores.

69.2 INDICADORES GENÉRICOS

Estos son los indicadores genéricos de la línea de comando de Csound. Las diversas implementaciones de distintas plataformas pueden no reaccionar de la misma manera a estos indicadores.

El formato de la línea de comando es el siguiente:

```
Csound [-indicadores] nombreorquesta nombrepartitura
```

donde los argumentos son de dos tipos: argumentos *indicadores* (que empiezan por un signo "-") y argumentos de *nombre* (como por ejemplo los nombres de los ficheros). Ciertos indicadores necesitan a continuación un nombre o un argumento numérico. Los indicadores disponibles son los siguientes:

```
-U nombreutil ejecuta la utilidad nombreutil  
-C usa el procesado de partitura de Cscore  
-I ejecución de la orquesta sólo en tiempo de inicialización (i-)  
-n no escribe sonido en el disco  
-i nombrefichero usa nombrefichero como fichero de entrada de audio  
-o nombrefichero usa nombrefichero como fichero de salida de audio  
-b N número de sample frames (períodos de control) por buffer en software de E/S de sonido  
-B N número de muestras por buffer en hardware de E/S de audio  
-A crea un fichero de sonido de salida en formato AIFF  
-W crea un fichero de sonido de salida en formato WAV  
-J crea un fichero de sonido de salida en el formato del IRCAM  
-h elimina la cabecera del fichero de sonido de salida
```

- c muestras de sonido de 8 bits (tipo signed_char)
- a muestras de sonido A-LAW
- 8 muestras de sonido de 8 bits (tipo unsigned_char)
- u muestras de sonido U-LAW
- s muestras de sonido tipo short_int
- l muestras de sonido tipo long_int
- f muestras de sonido float
- r N sustitución de la frecuencia de audio de la orquesta (sustituye el valor de sr)
- k N sustitución de la frecuencia de control de la orquesta (sustituye el valor de kr)
- v traducción de la orquesta con mensajes de explicación
- m N Nivel de mensajes TTY. Suma de: 1=amplitud de las notas, 2=mensajes de nota fuera de rango, 4=advertencias
- d suprime la salida en pantalla
- g suprime los gráficos (usa sólo caracteres ASCII)
- G suprime los gráficos (usa sólo salidas Postscript)
- S la partitura está en formato Scot
- x **nombrefichero** extrae de score.srt usando el fichero de extracción nombrefichero
- t N usa los pulsos no interpretados de la partitura, inicialmente a un tempo N
- L **nombredisp** lee los eventos de partitura en tiempo real del dispositivo nombredisp
- M **nombredisp** lee los eventos MIDI en tiempo real del dispositivo nombredisp
- F **nombrefichero** lee el fulgo de eventos MIDI del fichero nombrefichero
- P N umbral del pedal de sostenimiento MIDI (0 - 128)
- R reescribe continuamente la cabecera mientras escribe el fichero de sonido (WAV o AIFF)
- H1 genera un informe progresivo cada línea
- H2 genera un . cada vez que se escribe el contenido de un buffer
- H3 informa del tamaño de la salida en segundos
- H4 toca una campanita por cada buffer de salida escrito
- N notifica (toca la campanita) cuando se termina una pista MIDI o de la partitura
- T finaliza la ejecución cuando se termina una pista MIDI
- D pospone la carga del fichero de sonido de GEN01 hasta el momento de la ejecución
- z lista los opcodes de la versión de Csound que se está usando
- z1 lista los opcodes con argumentos en la versión actual
- **nombrefichero** manda todas las salidas de texto al fichero *nombrefichero*
- j **nombrefichero** deriva los mensajes de la consola de la base de datos *nombrefichero*

69.3 INDICADORES ESPECÍFICOS DE PC WINDOWS

- j **num** indica el número de filas de texto de la consola (25 por defecto)
- J **num** indica el número de columnas de texto de la consola (80 por defecto)
- K **num** permite la entrada MIDI. num (opcional) es el puerto de entrada MIDI
- q **num** número del dispositivo de salida de audio (usado sólo si hay instalado más de uno)
- p **num** número de buffers de salida de audio (4 por defecto; máximo 40)
- O suprime toda salida de texto en pantalla para una mejor ejecución en tiempo real
- e permite cualquier frecuencia de muestreo (usado sólo con tarjetas de sonido que soporten esta característica)
- y no espera que se pulse una tecla para salir
- E permite la salida en pantalla de gráficos para WCSHELL (por el prof. Riccardo Bianchini)
- Q **num** habilita la salida MIDI. num (opcional) es el número del puerto de salida MIDI
- Y suprime la salida de ondas en tiempo real para una mejor precisión en la salida MIDI
- * cede el control al sistema hasta que el buffer de audio de salida esté lleno

69.4 INDICADORES ESPECÍFICOS DE MACINTOSH

-q **sampdir** indica el directorio en el que se encuentran las muestras
-Q **analdir** indica el directorio en el que se encuentran los análisis
-X **snddir** indica el directorio para guardar los ficheros de sonido
-V **num** indica el tamaño del buffer de la pantalla
-E **num** indica el número de gráficos guardados
-p tocar al terminar
-e **num** indica un factor de reescala
-w indica la grabación de datos MIDI
-y **num** indica la frecuencia para el flujo de los mensajes en pantalla
-Y **num** indica la frecuencia para el archivo de los mensajes en pantalla

69.5 DESCRIPCIÓN

Los indicadores pueden aparecer en cualquier lugar de la línea de comando, tanto juntos como por separado. Un indicador que requiera un nombre o un número lo encontrará en ese argumento o en el inmediatamente posterior. Las siguientes líneas son, por tanto, idénticas:

```
CSound -nm3 nombreorq -Sxxnombrefich nombrepart
```

```
CSound -n -m 3 nombreorq -x xnombrefich -S nombrepart
```

Todos los indicadores y nombres son opcionales. Los valores por defectos son:

```
CSound -s -otest -b1024 -B1024 -m7 -P128 nombreorq nombrepart
```

donde *nombreorq* es un fichero que contiene el código de la orquesta de **Csound** y *nombrepart* es un fichero con los datos de la partitura en formato numérico estándar, opcionalmente preordenado y con sus tempi modificados. Si *nombrepart* se omite, hay dos opciones por defecto:

1) si se esperan entradas en tiempo real (-L, -M or -F), se usa un fichero de partitura con la única sentencia 'f 0 3600' (es decir, estar pendiente de las entradas en tiempo real durante una hora)

2) si no, **Csound** usa el último *score.srt* procesado anteriormente en el directorio actual. **Csound** informa de las varias fases del procesado de la partitura y de la orquesta tal y como se van produciendo, generando comprobaciones de sintaxis y errores durante el proceso. Una vez que la ejecución ha comenzado, cualquier mensaje de error provendrá del cargador del instrumento o de los opcodes mismos.

La línea de comando de **Csound** puede incluir cualquier combinación con sentido de los siguientes indicadores, con los valores por defecto que se indican:

CSound -U

Llama a las utilidades de preprocesado: **sndinfo**, **hetro**, **lpanal**, **pvanal**, **cvanal**.

CSound -I

Sólo pasadas de inicialización. Asigna e inicializa todos los instrumentos según la partitura pero omite todo el procesado de la ejecución (es decir, no evalúa señales de control ni de audio, por tanto ni amplitudes ni sonido alguno). Es una manera rápida de comprobar los campos p de la partitura y las variables de inicialización (i-) de la partitura.

CSound -n

No produce sonido. Realiza todo el procesado pero omite la escritura de sonido a disco. Este indicador no altera la ejecución en ninguna otra forma.

CSound -i isfname

Nombre del fichero de sonido de entrada. Si no es una ruta completa, el fichero se busca primero en el directorio actual, luego en el que señala la variable de entorno SSDIR (si está definida) y después en el que indica SFDIR. El nombre *stdin* hará que se lea el audio de la entrada estándar. Si RTAUDIO (audio en tiempo real) está habilitado, el nombre *devaudio* esperará el sonido del dispositivo de entrada de audio del host.

CSound -o osfname

Nombre del fichero de sonido de salida. . Si no es una ruta completa, el fichero se escribirá primero en el directorio que señala la variable de entorno SFDIR (si está definida), si no en el que directorio actual. El nombre *stdout* hará que las muestras de audio se escriban en la salida estándar. Si no se proporciona ningún nombre, el nombre por defecto es *test*. Si RTAUDIO (audio en tiempo real) está habilitado, el nombre *devaudio* enviará el sonido al dispositivo de salida de audio del host.

CSound -b Numb

Número de períodos de control (kprds, o arrays de muestras) en el buffer de software de E/S. Los valores grandes son eficientes, pero los pequeños reducen el retraso de la E/S de audio. El valor por defecto es 1024. En ejecuciones en tiempo real, Csound espera la E/S de audio en los límites de *Numb*. También procesa el sonido (y obtiene otras entradas como por ejemplo las del MIDI) en los límites *ksmps* de la orquesta. Los dos pueden hacerse sincrónicos. Por conveniencia, si *Numb* es negativo el valor efectivo es $ksmps * N$ (audio sincrónico en los límites de los períodos de control). Con un valor pequeño (por ejemplo 1) las peticiones de entrada son frecuentes y también dentro de los límites de muestra fijos del DAC (convertidor digital-analógico).

CSound -B Numb

Número de arrays de muestras de audio contenidos en el buffer en hardware del DAC. Es un umbral en el que la E/S en software de audio (arriba) debe esperar antes de volver. Un número pequeño reduce el retraso de la E/S de audio. Pero el valor normalmente está limitado por el propio hardware y valores demasiado pequeños pueden provocar retrasos en los datos. El valor por defecto es 1024.

CSound -h

Fichero de sonido de salida sin cabecera. Escribe sólo las muestras digitales pero no la cabecera.

CSound {-c, -a, -u, -s, -l, -f}

Formato de las muestras de audio en el fichero de sonido de salida. Es uno de los siguientes:

c 8-bit signed char)

a 8-bit a-law

u 8-bit u-law

s short int

l long int

f single-precision float (valor en coma flotante de precisión simple, no reproducible pero puede ser leído con el indicador -i, con **soundin** o con **GEN01**).

CSound -A

Escribe el fichero de sonido de salida en formato AIFF. Restringe los formatos anteriores a c,s,l o f (AIFC).

CSound -W

Escribe un fichero de sonido en formato WAV

CSound -J

Escribe un fichero de sonido en el formato del IRCAM

CSound -v

Informa de los procesos de traducción y ejecución. Proporciona detalles de la traducción y ejecución de la orquesta, permitiendo localizar más fácilmente los errores.

CSound -m Numb

Nivel de mensajes de la salida estándar. Es la suma de 3 indicadores de control de impresión, habilitados por los siguientes valores: 1 = mensajes de amplitud de nota, 2 = mensajes de muestras fuera de control, y = mensajes de advertencia. El valor por defecto es *m7* (es decir, 1+2+4, todos los mensajes activos).

CSound -d

Suprime cualquier salida en pantalla.

CSound -g

Representa las salidas gráficas con caracteres ASCII, para poder verlas en cualquier terminal.

CSound -S

Interpreta el nombre de la partitura como un fichero en el formato de Scot y crea con un fichero de partitura estándar (de nombre "score"), luego lo ordena y lo ejecuta.

CSound -x xfile

Extrae una parte de la partitura ordenada *score.srt* de acuerdo a *xfile* (ver **Extract**).

CSound -t Numb

Usa los pulsos no interpretados de *score.srt* de esa ejecución y pone el tiempo inicial a *Numb* pulsos por minuto. Cuando este indicador esta activo, el tiempo de la ejecución de la partitura puede ser controlado también desde dentro de la partitura.

CSound -L devname

Lee líneas de eventos de partitura en tiempo real del dispositivo *devname*. El nombre *stdin* permitirá teclear los eventos desde tu terminal, conducido desde otro proceso. Cada línea de eventos se termina por un retorno de carro. Los eventos deben estar codificados justo igual que en la partitura numérica estándar, excepto en que un evento con un $p2=0$ será ejecutado inmediatamente, mientras que uno con $p2=T$ será ejecutado T segundos después de su llegada. Los eventos pueden llegar en cualquier instante y en cualquier orden. El procesado de Carry en la partitura es legal aquí también, como lo son las notas tenidas ($p3$ negativos) y los argumentos compuestos de cadenas de caracteres. Sin embargo, el ramping o los símbolos *pp* y *np* no pueden usarse aquí.

CSound -M devname

Lee eventos MIDI desde el dispositivo *devname*.

CSound -F mfname

Lee eventos MIDI del fichero MIDI *mfname*.

CSound -P Numb

Ajusta el valor umbral del pedal de sostenimiento MIDI (0-128). El valor oficial de cambio de pedal (64) es normalmente demasiado bajo. Es más realista un valor por encima de 100. El valor por defecto, 128, bloqueará cualquier información de cambio de pedal.

CSound -N

Notifica por medio de un sonido cuando ha finalizado una pista MIDI o de partitura.

CSound -T

Finaliza la ejecución cuando termina la pista MIDI.

Indicadores específicos de PC/Windows

CSound -K num

Número de dispositivo MIDI (opcional, usado sólo si hay más de un dispositivo).

CSound -q num

Número de dispositivo de salida WAVE (opcional, usado sólo si hay más de uno).

CSound -p num

Número de buffers de salida de audio (opcional; valor por defecto 4, máximo 40). -b (longitud del buffer) y -p son indicadores relacionados el uno con el otro. Encontrar los valores óptimos para "-b" y "-p" requiere algo de experiencia: un buffer más largo implica más período de latencia pero también más seguridad para evitar ruidos e interrupciones (el indicador "-B" está ahora obsoleto, así que no lo uses). Puedes reducir drásticamente la longitud del buffer y el retraso usando el indicador -e y frecuencias de audio y control óptimas (ver abajo).

Observa que a veces un buffer más pequeño puede manejar el flujo de audio mejor que uno más largo. Sólo la experimentación puede llevarte a conseguir valores óptimos de -b. Los valores de los indicadores -b y -p pueden ser reducidos usando frecuencias de audio y control "redondeadas" (por ejemplo ar=32000 y kr=320 o ar=40000 y kr400, etc...) junto con el indicador -e (hasta ahora esta característica sólo ha sido comprobada con las tarjetas SB16 ASP y AWE32. Se ignora si otras tarjetas no la soportan). Reduciendo los valores de los indicadores -b y -p se consigue reducir el período de latencia (retraso) y así una ejecución interactiva en tiempo real más fluida.

CSound -j num

Número de filas virtuales de texto en la consola.

CSound -J num

Número de columnas virtuales de texto en la consola.

CSound -O (letra mayúscula)

Suprime todas las salidas en pantalla para una mejor ejecución en tiempo real. Este indicador es mejor que "-m0" porque éste permite algún mensaje de salida en la consola. Usando ambos indicadores juntos puede conseguirse la velocidad máxima de ejecución.

CSound -e

Permite una frecuencia de muestreo arbitraria (sólo para tarjetas que soporten esta característica),

CSound -y

No espera a que se pulse ninguna tecla en la salida del programa.

CSound -E

Salida gráfica para WCSHELL por Riccardo Bianchini.

CSound -Q num

Permite operaciones de salida MIDI y opcionalmente elige el número de dispositivo MIDI (si el argumento *num* está presente). Este indicador permite ejecuciones paralelas de salida MIDI y del DAC. Desafortunadamente, el flujo de datos en tiempo real implementado en Csound es completamente dependiente del buffer del DAC. Así que las operaciones de salida MIDI pueden presentar algunas irregularidades temporales. Estas irregularidades pueden ser totalmente eliminadas cuando se suprimen las operaciones mismas del DAC (ver indicador -Y).

CSound -Y

Deshabilita la salida de onda (para mayor precisión en las ejecuciones de salida MIDI). Esto mejora la resolución temporal de las operaciones de salida MIDI cuando son usadas junto con el indicador "-Q". Se recomienda usar "-Y" con frecuencias de control bajas (máximo $kr=1000$). Como en Win95 la máxima resolución de reloj es de una milésima de segundo, pueden darse resultados impredecibles cuando se usa a frecuencias de control mayores que 1000. También es muy importante configurar sólo valores de frecuencia de control (kr) en los que la división $1000/kr$ produce resultados enteros (algún ejemplo: $kr=10, 20, 50, 100, 125, 200, 250$, etc) porque el reloj de Windows95 sólo maneja períodos enteros en milisegundos.

Si usas una frecuencia de control que produzca un resultado no entero con la fórmula anterior, Csound parece ejecutarse correctamente pero los valores de tiempo no serán fiables. Con mi ordenador trabajo muy bien con una frecuencia de control de 200 Hz ($kr=200$). Puede ser que con ordenadores más lentos un valor algo inferior trabaje mejor. ¡Experimenta! Recomiendo usar $kr=200$ o menos porque con valores mayores se carga demasiado de trabajo al sistema entero y no proporciona una notable mejoría en la resolución. Una resolución de $1/200$ de segundo es suficientemente precisa para casi todas las aplicaciones MIDI. Debes respetar la proporción $sr/kr/ksmps$ incluso si el valor de sr carece de significado cuando se usa el indicador "-Y". De otra manera obtendrás un mensaje de error que parará la ejecución.

CSound -*

Obliga a Csound a ceder el control al sistema hasta que el contenido del buffer de salida de audio supere cierto umbral. Por debajo de ese umbral Csound continúa procesando, mientras que por encima de él Csound cede el control a Windows. Esto proporciona una gran mejora del proceso multitarea. Habilitando esta opción se reduce la polifonía un poco cuando se usa un buffer pequeño. Así que el usuario debe incrementar el número (indicador "-p") y la longitud (indicador "-b") de los buffers cuando se selecciona el indicador "-*". Experimenta para encontrar los mejores resultados. No uses este indicador cuando el tiempo de respuesta a los gestos sea crítica.

70 UNIFICADO FICHERO FORMATO DE PARA ORQUESTAS Y PARTITURAS

70.1 DESCRIPCIÓN

El formato de fichero unificado, introducido en la versión 3.50 de Csound, permite a los ficheros de la orquesta y de la partitura, así como los indicadores de la línea de comando, ser combinados en un único fichero. El fichero tiene la extensión .csd. Este formato fue originalmente introducido por Michael Gogins en AXCsound.

El fichero es un fichero de datos estructurado que usa un lenguaje de marcas, similar a cualquier SGML, como por ejemplo HTML. Las marcas de comienzo (`<tag>`) y de final (`</tag>`) se usan para delimitar los distintos elementos. El fichero se guarda como un fichero de texto.

70.2 FORMATO DE FICHEROS DE DATOS ESTRUCTURADOS

70.2.1 ELEMENTOS

El elemento Csound se usa para alertar al compilador de Csound del uso del formato .csd. El fichero debe empezar con la marca `<CsoundSynthesizer>`. La última línea del fichero debe ser la marca de final `</CsoundSynthesizer>`. Los restante elementos se definen más abajo.

70.2.1.1 Opciones

Los indicadores de la línea de comando de Csound se ponen en el elemento Opciones. Esta sección viene delimitada por las marcas de comienzo `<CsOptions>` y final `</CsOptions>`. Las líneas que empiecen con un signo # o un ; se tratan como comentarios.

70.2.1.2 Instrumentos (Orquesta)

Las definiciones de los instrumentos (orquesta) se ponen en el elemento Instrumentos. Las sentencias y la sintaxis en esta sección son idénticas a las del fichero orquesta de Csound y tienen los mismos requerimientos, incluyendo las sentencias de cabecera (`sr`, `kr`, etc.). Este elemento Instrumento está delimitado por las marcas de comienzo `<CsInstruments>` y final `</CsInstruments>`.

70.2.1.3 Partitura

Las sentencias de partitura de Csound se ponen en el elemento Partitura. Las sentencias y la sintaxis en esta sección son idénticas a las del fichero partitura de Csound y tienen los mismos requerimientos. El elemento Partitura viene delimitado por las marcas de comienzo `<CsScore>` y final `</CsScore>`.

70.2.1.4 Ejemplo

Abajo hay un fichero csd de ejemplo, *test.csd* que renderiza un fichero Wav a una frecuencia de muestreo de 44.1 kHz que contiene un segundo de una onda sinusoidal de frecuencia 1kHz. Las salidas en pantalla son suprimidas. *test.csd* fue creado por dos ficheros, *tone.orc* y *tone.sco* con la adición de los indicadores de la línea de comando.

```
<CsoundSynthesizer>
    ; test.csd - a Csound structured data file
<CsOptions>
    -W -d -o tone.wav
</CsOptions>
<CsInstruments>
    ; originally tone.orc
    sr = 44100
    kr = 4410
    ksmps = 10
    nchnls = 1
    instr 1
    al oscil p4, p5, 1 ; simple oscillator
    out al
    endin
</CsInstruments>
<CsScore>
    ; originally tone.sco
    f1 0 8192 10 1
    i1 0 1 20000 1000 ;play one second of one kHz tone
    e
</CsScore>
</CsoundSynthesizer>
```

70.3 EL FICHERO DE PARÁMETROS DE LA LÍNEA DE COMANDO

Si el fichero **.csoundrc** existe, será usado para configurar los parámetros de la línea de comando. Estos pueden ser sustituidos. Usa el mismo formato que un fichero **.csd**. Las líneas que empiezan con un **#** o un **;** se consideran comentarios.

71 PREPROCESADO DEL FICHERO PARTITURA

71.1 LA OPCIÓN EXTRACT

Esta característica extraerá un segmento de un fichero de partitura numérica estándar ordenado, de acuerdo con las instrucciones que proporcione un fichero de control. Este fichero de control contiene una lista de instrumentos y dos instantes en el tiempo, "desde" y "hasta", en la forma:

```
instruments 1 2 from 1:27.5 to 2:2
```

Las etiquetas de los componentes pueden ser abreviadas como i, f y t (respectivamente). Los instantes en el tiempo indican el principio y el final de la extracción en los términos:

```
[sección no.] : [pulso no.].
```

cualquiera de las tres partes es opcional. Los valores por defecto para i, f y t ausentes son:

```
todos los instrumentos, comienzo de la partitura, final de la partitura.
```

71.2 PREPROCESADO INDEPENDIENTE CON SCSORT

Aunque el resultado de todo el preprocesado de la partitura es concentrado en el fichero `score.srt` después de la ejecución de la orquesta (existe tan pronto como el preprocesado de la partitura haya tenido lugar), el usuario puede algunas veces desear ejecutar estas fases independientemente. El comando:

```
scot nombrefichero
```

procesará el fichero formateado por Scot, y dejará como resultado una partitura numérica estándar en un fichero su para posterior revisión o procesado.

El comando:

```
scscort < fichent > fichsal
```

pasará un fichero de partitura de entrada *fichent* a través de las rutinas de preprocesado Carry, Tempo y Sort, dejando el resultado en el fichero de salida *fichsal*.

Igualmente, **extract**, que es también normalmente invocado como parte de la línea de comando de Csound, puede ser llamado como un programa independiente con:

```
extract xfile < score.sort > score.extract
```

Este comando espera una partitura previamente ordenada. Una partitura no ordenada debe ser enviada primero a Scsort y luego conducida al programa de extracción **extract**.

```
scsort < scorefile | extract xfile > score.extract
```

72 UTILIDADES PARA LOS FICHEROS DE SONIDO

Las utilidades de Csound son programas de preprocesado de ficheros de sonido que devuelven información sobre un fichero de sonido o crean alguna versión analizada de él para usarla con ciertos generadores de Csound. Aunque diferentes en objetivos, todos comparten un mismo mecanismo de acceso a los ficheros de sonido y pueden ser descritos como una única colección de utilidades.

Las utilidades para los ficheros de sonido pueden ser invocadas de dos formas equivalentes:

```
CSound -U nombreutilidad [indicadores] nombrefichero . . .  
nombreutilidad [indicadores] nombrefichero . . .
```

En la primera forma, la utilidad es llamada como parte del ejecutable Csound, mientras que en la segunda se llama como un programa aislado. La segunda forma es 200K más pequeña pero ambas son idénticas en funcionamiento. La primera es conveniente porque no requiere el mantenimiento y uso de varios programas independientes - un programa lo hace todo. Cuando se usa esta forma, un indicador **-U** detectado en la línea de comando hará que todos los demás indicadores y nombres sean interpretados por la utilidad invocada, es decir, Csound no generará nada y el programa terminará al final de la ejecución de la utilidad.

Directorios. Los nombres de los ficheros son de dos tipos: ficheros de sonido y ficheros de los análisis resultantes. Cada uno tiene una convención jerárquica de nombres, influida por el directorio desde el que la utilidad es invocada. Los ficheros de sonido fuente con una ruta completa (empiezan con un punto ".", una barra "/" o para ThinckC incluye dos puntos ":"), serán buscados sólo en el directorio especificado. Los ficheros de sonido sin una ruta se buscarán primero en el directorio actual, luego en el directorio indicado por la variable de entorno SSDIR (si está definida) y después en el directorio especificado por SFDIR. Una búsqueda infructuosa devolverá un mensaje de error al abrir el fichero.

Los ficheros de análisis resultante se escriben en el directorio actual, o en el directorio especificado si se incluye la ruta. Es conveniente guardar los ficheros de análisis separados de los ficheros de sonido, normalmente en el directorio de la variable SADIR. Es mejor hacer el análisis desde dentro del directorio SADIR. Cuando un fichero de análisis es invocado más tarde por un generador de Csound, éste se busca primero en el directorio actual y luego el directorio especificado por SADIR.

Formatos de los Ficheros de Sonido. Csound puede leer y escribir ficheros de audio en varios formatos, los cuales son especificados por los indicadores de la línea de comando de Csound. En la lectura, el formato queda determinado según la cabecera del fichero, siendo los datos automáticamente convertidos a valores en coma flotante durante el procesado interno. Cuando Csound está instalado en un ordenador con convenciones locales para ficheros de sonido (SUN, NeXT, Macintosh) puede incluir condicionalmente código de empaquetamiento local que cree ficheros de sonido que no son portables a otros ordenadores. Sin embargo, Csound puede siempre en cualquier ordenador generar y leer ficheros AIFF, que sí es un formato portable. Las librerías de sonido portables son típicamente AIFF, y la variable SSDIR normalmente apunta a un directorio que contiene ese tipo de sonidos. Si está definida, el directorio SSDIR está siempre en la ruta de búsqueda durante el acceso al fichero de sonido.

Observa que algunos sonidos sampleados AIFF tienen un bucle de audio para su ejecución mantenida. Los programas de análisis atravesarán cualquier segmento de bucle sólo una vez.

Para los ficheros de sonido sin cabecera, puede especificarse una frecuencia de muestreo en la línea de comando (o su valor por defecto). Si ambos, la cabecera y los indicadores, están presentes, el valor de los indicadores sustituirán a los de la cabecera.

Cuando se accede al sonido mediante alguno de los programas de análisis de audio, se lee sólo un único canal. Para los archivos estéreo o cuadrafónicos, el valor por defecto es el canal 1. Los demás canales pueden ser obtenidos especificándolo.

Autor

Dan Ellis

MIT Media Lab

Cambridge, Massachusetts

1990

72.1 sndinfo

72.1.1 DESCRIPCIÓN

Consigue la información básica sobre un o más ficheros de sonido.

```
Csound -U sndinfo nombreficheros . . .  
o  
sndinfo nombreficheros . . .
```

sndinfo intentará encontrar cada fichero especificado, abrirlo para su lectura, leer su cabecera y luego informar de la información básica que encuentre. El orden de búsqueda por los directorios de ficheros de sonido es el descrito arriba. Si el fichero es de tipo AIFF, se listan al principio algunos detalles más.

72.1.2 EJEMPLO

```
Csound -U sndinfo test Bosendorfer/"BOSEN mf A0 st" foo foo2
```

donde la variable de entorno `SFDIR = /u/bv/sound`, y `SSDIR = /so/bv/Samples`, producirán lo siguiente:

```
util SNDINFO:  
/u/bv/sound/test:  
srate 22050, monaural, 16 bit shorts, 1.10 seconds  
headersiz 1024, datasiz 48500 (24250 sample frames)  
/so/bv/Samples/Bosendorfer/BOSEN mf A0 st: AIFF, 197586 stereo samples,  
base Frq 261.6 (midi 60), sustnLp: mode 1, 121642 to 197454, relesLp: mode 0  
AIFF soundfile, looping with modes 1, 0  
srate 44100, stereo, 16 bit shorts, 4.48 seconds  
headersiz 402, datasiz 790344 (197586 sample frames)  
/u/bv/sound/foo:  
no recognizable soundfile header  
/u/bv/sound/foo2:  
couldn't find
```

72.2 hetro

72.2.1 DESCRIPCIÓN

Análisis por filtrado heterodino para el generador **adsyn** de Csound.

```
Csound -U hetro [indicadores] fichentrada fichsalida  
o  
hetro [indicadores] fichentrada fichsalida
```

hetro coge un fichero de sonido de entrada, lo descompone en sus parciales sinusoidales y devuelve una descripción de los parciales componentes en forma de pistas con los puntos de inflexión de las amplitudes y las frecuencias. El análisis está condicionado por los indicadores de control descritos abajo. Un espacio entre el indicador y su valor es opcional.

-s frecuencia de muestreo del fichero de audio de entrada. Esta sustituirá la frecuencia de muestreo de la cabecera del fichero de sonido, que de otra manera, sería la que se aplicaría. Si ninguno de los dos está presente, el valor por defecto es 1000. Observa que para la síntesis con **adsyn** la frecuencia de muestreo del fichero fuente no tiene que coincidir con la de la orquesta que lo genera.

-c número de canal buscado. El valor por defecto es 1.

-b instante inicial (en segundos) del segmento de audio a ser analizado. El valor por defecto es 0.0

-d duración (en segundos) del segmento de audio a ser analizado. El valor por defecto, 0.0, significa hasta el final del fichero. La longitud máxima es de 32.766 segundos.

-f frecuencia inicial estimada de la fundamental, necesaria para inicializar el análisis del filtrado. El valor por defecto es 100 (Hz)

-h número de armónicos buscados en el fichero de sonido. El valor por defecto es 10. El máximo está en función de la memoria disponible.

-M máxima amplitud sumada por todas la pistas. El valor por defecto es 32767.

-m umbral de amplitud por debajo del cual un par amplitud/frecuencia es considerado inactivo y por tanto no contribuirá a la suma total de salida. Los valores típicos son: 128 (48 dB por debajo del máximo), 64 (54 dB por debajo), 32 (60 dB por debajo), 0 (sin umbral). El valor por defecto es 64 (54dB por debajo).

-n número inicial de puntos de inflexión del análisis en cada pista de amplitud y frecuencia, antes de la valoración del umbral (-m) y de la consolidación lineal de dichos puntos de inflexión. Los puntos iniciales se distribuyen uniformemente en la duración total. El valor por defecto es 256.

-l sustituye un filtro Butterworth pasa bajos de tercer orden con una frecuencia de corte *cutfreq* (en csp), en lugar del filtro peine de ponderación usado por defecto. El valor por defecto es 0 (no usarlo).

72.2.2 EJEMPLO

hetro -s44100 -b.5 -d2.5 -h16 -M24000 audiofile.test adsynfile7

Esto analizará 2.5 segundos del canal 1 de un fichero llamado "audiofile.test", grabado a 44.1 kHz, empezando a los .5 segundos del principio del fichero y luego coloca el resultado en un fichero llamado "adsynfile7". Sólo pedimos los primeros 16 armónicos del sonido, con 256 puntos de inflexión iniciales por pista de amplitud o frecuencia y un nivel de pico total de 24000 (amplitud). Se estima que la frecuencia fundamental empieza a 100 Hz. El umbral de amplitud está en 54 dB por debajo del tope. El filtro Butterworth pasa bajos no está habilitado.

72.2.3 FORMATO DEL FICHERO

El fichero de salida contiene valores de amplitudes y frecuencias secuenciados en el tiempo para cada parcial de una fuente de audio compleja. La información está forma de puntos de inflexión (instante, valor, instante, valor, etc...), usando enteros de 16 bits en el rango de 0 a 32767. El tiempo se proporciona en milisegundos y la frecuencia en Herzios (cps). Los datos de los puntos de inflexión son exclusivamente positivos, y los valores -1 y -2 únicamente indican el comienzo de nuevas pistas de de amplitud y frecuencia. Una pista concluye con el valor 32767. Antes de empezar a escribir, cada pista es reducida en datos de acuerdo con un umbral de amplitud y un proceso de consolidación lineal de los puntos de inflexión.

Un parcial componente viene definido por dos series de puntos de inflexión: una serie de amplitudes y una serie de frecuencias. Dentro de un fichero complejo estas series pueden aparecer en cualquier orden (amplitud, frecuencia, amplitud... o amplitud, amplitud... luego frecuencia, frecuencia,...). Durante la resíntesis llevada a cabo por **adsyn** las series son automáticamente emparejadas (amplitud, frecuencia) en el orden en que fueron encontradas. Esto debería dar un número igual para cada una.

Un fichero legal de **adsyn** podría tener el siguiente formato:

```
-1 instante1 valor1 .. instanteK valorK 32767 ; punto de inflexión para la amplitud
del parcial 1
-2 instante1 valor1 .. instanteL valorL 32767 ; punto de inflexión para la frecuencia
del parcial 1
-1 instante1 valor1 .. instanteM valorM 32767 ; punto de inflexión para la amplitud
del parcial 2
-2 instante1 valor1 .. instanteN valorN 32767 ; punto de inflexión para la frecuencia
del parcial 2
-2 instante1 valor1 .....
-2 instante1 valor1 ..... ; pistas emparejables para los parciales 3 y 4
-1 instante1 valor1 .....
-1 instante2 valor1 .....
```

72.3 lpanal

72.3.1 DESCRIPCIÓN

Análisis predictivo lineal (lpc) para los generadores **lp** de Csound.

```
Csound -U lpanal [indicadores] fichentrada fichsalida  
o lpanal [indicadores] fichentrada fichsalida
```

lpanal realiza tanto el análisis lpc como el rastreo de altura de un fichero de sonido para producir una secuencia de *frames* (cuadros), ordenada en el tiempo, de información de control apropiada para la resíntesis con Csound. El análisis viene condicionado por los indicadores de control descritos a continuación. Un espacio entre el indicador y su valor es opcional.

-a (almacenamiento alternativo) pide a lpanal que escriba un fichero con los valores polares, en vez de los coeficientes usuales, usados en el filtrado. Cuando **lpread/lpreson** se usan con ficheros de valores polares, la estabilización es automáticamente ejecutada y el filtro no debería comportarse de manera incoherente (este es el valor por defecto en el interfaz gráfico de Windows). Cambiado por Marc Resibois.

-s frecuencia de muestreo del fichero de audio de entrada. Esto sustituirá la frecuencia de muestreo de la cabecera del fichero de sonido, que de otra forma sería aplicada. Si ninguno de los dos está presente, el valor por defecto es 10000.

-c número del canal buscado. El valor por defecto es 1.

-b instante inicial (en segundos) del segmento de audio a ser analizado. El valor por defecto es 0.0

-d duración (en segundos) del segmento de audio a ser analizado. El valor por defecto, 0.0, significa hasta el final del fichero.

-p número de polos para el análisis. El valor por defecto es 34. El máximo es 50.

-h tamaño del salto (en muestras) entre *frames* sucesivos del análisis. Esto determina el número de cuadros (frames) por segundo (frecuencia de muestreo / tamaño del salto) en el fichero de control de salida. El tamaño del cuadro del análisis es el doble de muestras que el tamaño del salto. El valor por defecto es 200. El máximo es 500.

-C cadena de caracteres para los campos de comentarios de la cabecera del fichero lp. El valor por defecto es una cadena nula.

-P frecuencia más baja (en Hz) del rastreo de altura. -P0 indica que no hay rastreo.

-Q frecuencia más alta (en Hz) del rastreo de altura. Cuanto más estrecho sea el rango de alturas, más precisa será la estimación. Los valores por defecto son -P70, -Q200.

-v nivel de detalle en los mensajes de información en el terminal. 0 = ninguno, 1 = mensaje, 2 = depuración. El valor por defecto es 0.

72.3.2 EJEMPLO

```
lpanal -a -p26 -d2.5 -P100 -Q400 audiofile.test lpfil22
```

analizará los primeros 2.5 segundos del fichero "audiofile.test", produciendo *srata/200* frames por segundo, cada uno con los coeficientes para el filtro de 26 polos y una estimación de altura entre 100 y 400 Hz. La salida estabilizada (-a) será guardada en el fichero "lpfil22" en el directorio actual.

72.3.3 FORMATO DE FICHERO

La salida de un fichero se compone de una cabecera de identificación más una serie de cuadros de datos de análisis en coma flotante. Cada cuadro contiene 4 valores de altura y ganancia, seguidos por los coeficientes para el filtro de npolar. El fichero puede ser leído por el opcode **lpread** de Csound.

lpanal es una amplia modificación de un programa de análisis lpc de Paul Lansky.

72.4 pvanal

72.4.1 DESCRIPCIÓN

Análisis de Fourier para el generador **pvoc** de Csound.

```
CSound -U pvanal [indicadores] fichentrada fichsalida  
o pvanal [indicadores] fichentrada fichsalida
```

pvanal convierte un fichero de sonido en una serie de transformadas de Fourier de corto plazo (Short Time Fourier Transform, STFT) tomadas a intervalos regulares de tiempo (es decir, una representación del sonido en el dominio de la frecuencia). El fichero de salida puede ser usado por **pvoc** para generar fragmentos de audio basados en la muestra original, con una escala temporal y una altura arbitraria y dinámicamente modificadas a voluntad. El análisis viene condicionado por los indicadores descritos abajo. Un espacio entre el indicador y su valor es opcional.

-s frecuencia de muestreo del fichero de audio de entrada. Esta sustituirá la frecuencia de muestreo de la cabecera del fichero de sonido, que de otra manera sería aplicada. Si ninguno de los dos está presente, el valor por defecto es 10000.

-c número del canal buscado. El valor por defecto es 1.

-b instante inicial (en segundos) del segmento de audio a ser analizado. El valor por defecto es 0.0

-d duración (en segundos) del segmento de audio a ser analizado. El valor por defecto, 0.0, significa hasta el final del fichero.

-n tamaño del frame de la STFT, es decir, el número de muestras en cada frame del análisis de Fourier. Debe ser una potencia de 2, en el rango de 16 a 16384. Para resultados limpios, el frame debe ser mayor que el período más largo de todas las frecuencias de los parciales de la muestra. Sin embargo, frames muy largos producirán reverberación. El ancho de banda de cada pista de STFT viene determinado por el cociente frecuencia de muestreo/tamaño del frame. El tamaño por defecto es la potencia de 2 más pequeña que corresponda a más de 20 milisegundos de la fuente (es decir, 256 puntos en un muestreo a 10 kHz, dando un frame de 25.6 ms)

-w factor de solapamiento de ventanas. Esto controla el número de frames de la transformada de Fourier por segundo. El opcode **pvoc** de Csound interpolará los valores de frames sucesivos, pero demasiados pocos frames pueden generar una distorsión fácilmente audible. Un buen compromiso para el factor de solapamiento es 4, que indica que cada punto de entrada aparece en 4 ventanas distintas, o, a la inversa, que el offset (desplazamiento) entre frames sucesivos es la cuarta parte del tamaño del frame. El valor por defecto es 4. No uses este indicador con **-h**.

-h offset entre los frames de la STFT. Al revés que el anterior, especifica el incremento en muestras entre frames sucesivos del análisis (ver también **lpanal**). No lo uses con **-w**.

72.4.2 EJEMPLO

```
pvanal asound pvfile
```

analizará el fichero de sonido "asound" usando el tamaño de frame y el factor de solapamiento por defecto para producir el fichero "pvfile", listo para usar con el opcode **pvoc**.

72.4.3 FICHEROS

El fichero de salida tiene una cabecera de **pvoc** especial que contiene detalles del fichero de audio fuente, la frecuencia de frames del análisis y el solapamiento. Los datos de los frames del análisis son guardados como valores en coma flotante, con la magnitud y la frecuencia (Hz) para las primeras $N/2 + 1$ pistas de cada frame. La frecuencia codifica el incremento de fase de tal manera que para armónicos potentes da una buena aproximación de la frecuencia real. Para amplitudes bajas o armónicos que cambian rápidamente es menos significativa.

72.4.4 DIAGNOSTICOS

Imprime el número total de frames y los frames completados de cada 20.

72.4.5 AUTOR

Don Ellis
MIT Media Lab
Cambridge, Massachusetts
1990

72.5 cvanal

72.5.1 DESCRIPCIÓN

Análisis de Fourier de Respuesta a Impulso para el operador **convolve**.

```
CSound -U cvanal [indicadores] fichentrada fichsalida
```

cvanal convierte un fichero de sonido en un único frame de la transformada de Fourier. El fichero de salida puede ser usado por el operador **convolve** para realizar un proceso de Convolución rápida entre una señal de entrada y la respuesta al impulso original. El análisis viene condicionado por los indicadores descritos a continuación. Un espacio es opcional entre el indicador y su argumento.

-s frecuencia de muestreo del fichero de audio de entrada. Esto sustituirá la frecuencia de muestreo de la cabecera del fichero de sonido, que de otra manera sería aplicada. Si ninguno de los dos está presente, el valor por defecto es 10000.

-c número del canal buscado. Si se omite, se procesarán por defecto todos los canales. Si se proporciona un valor, sólo se procesará el canal seleccionado.

-b instante inicial (en segundos) del segmento de audio a ser analizado. El valor por defecto es 0.0

-d duración (en segundos) del segmento de audio a ser analizado. El valor por defecto, 0.0, significa hasta el final del fichero.

72.5.2 EJEMPLO

```
cvanal asound cvfile
```

analizará el fichero de sonido "asound" para producir el fichero "cvfile" para usar con **convolve**.

Consejo: Para usar los datos que todavía no están contenidos en un fichero de sonido, un convertidor que acepte ficheros de texto puede ser usado para crear un fichero de audio estándar. Es decir, el formato .DAT para SOX. Esto puede ser útil para implementar filtros FIR (Finite impulse response).

72.5.3 FICHEROS

El fichero de salida tiene una cabecera especial para **convolve**, que contiene detalles del fichero de audio fuente. Los datos del análisis se guardan como valores en coma flotante, en la forma real/imaginario.

NOTA: el fichero de análisis NO es un sistema independiente. Asegúrate de que los datos grabados del impulso original no se pierdan. Cuando se quiera, el fichero de análisis podrá ser recreado.

72.5.4 AUTOR

Greg Sullivan (Basado en el algoritmo expuesto en 'Elements Of Computer Music', de F. Richard Moore).

3.291 3.291 3.294 3.291 3.290 3.291 3.288 3.291 3.293 3.291
3.293 3.292 3.288 3.291 3.290 3.290 3.294 3.291 3.291 3.292
3.288 3.290 3.291 3.290 3.294 3.293 3.290 3.292 3.289 3.289
3.293 3.290 3.292 3.293 3.289 3.291 3.290 3.289 3.293 3.292
3.291 3.293 3.289 3.289 3.291 3.289 3.292 3.293 3.290 3.292
3.290 3.288 3.292 3.291 3.291 3.294 3.290 3.290 3.291 3.288
3.291 3.292 3.291 3.293 3.291 3.288 3.291 3.289 3.290 3.293
3.290 3.292 3.292 3.288 3.291 3.291 3.290 3.293 3.291 3.290
3.292 3.288 3.289 3.292 3.290 3.292 3.293 3.289 3.291 3.289
3.288 3.293 3.291 3.291 3.292 3.288 3.289 3.290 3.288 3.292
3.293 3.290 3.292 3.289 3.288 3.291 3.290 3.291 3.293 3.289
3.290 3.290 3.287 3.291 3.291 3.290 3.293 3.290 3.288 3.290
3.288 3.290 3.293 3.291 3.292 3.291 3.288 3.290 3.289 3.289
3.293 3.290 3.290 3.291 3.287 3.289 3.291 3.289 3.292 3.291
3.288 3.290 3.288 3.288 3.292 3.290 3.291 3.292 3.288 3.289
3.290 3.288 3.292 3.292 3.290 3.292 3.289 3.288 3.291 3.289
3.291 3.293 3.289 3.291 3.290 3.287 3.291 3.290 3.290 3.293
3.289 3.289 3.290 3.287 3.290 3.292 3.290 3.292 3.290 3.287
3.290 3.289 3.289 3.292 3.290 3.290 3.291 3.287 3.289 3.290
3.289 3.292 3.291 3.289 3.291 3.288

etc...

72.6.3 AUTOR

Richard Karpen
Seattle, Wash
1993 (Nuevo en la versión 3.57)

73 CSCORE

Cscore es un programa para generar y manipular partituras numéricas. Comprende varias funciones o subprogramas, llamadas por un programa de control escrito por el usuario, y puede ser invocado bien como un preprocesador de partituras aislado, bien como parte del sistema de Csound.

```
Cscore partituraentrada partiturasalida  
○  
CSound -C [otros indicadores] nombreorquesta nombrepartitura
```

Las funciones disponibles se suman a las funciones de la librería estándar del lenguaje C. Estas funciones pueden leer tanto partituras estándar o partituras preordenadas, pueden manejar y expandir los datos de mil maneras y luego ponerlas a disposición de la orquesta de Csound para su ejecución.

El programa de control escrito por el usuario debe estar escrito también en C y se compila y se enlaza a las funciones de Cscore (o al código entero de Csound) por el usuario. No es esencial conocer bien el lenguaje C para escribir este programa, ya que las llamadas a estas funciones tienen una sintaxis muy sencilla y son suficientemente potentes como para hacer la mayor parte del trabajo más complicado. Poco a poco se podrán ir escribiendo programas más potentes según se vaya cogiendo soltura con el lenguaje C.

73.1 EVENTOS, LISTAS Y OPERACIONES

Un evento en Cscore es equivalente a una sentencia de una *partitura numérica estándar* o una partitura a la que se han aplicado ya las modificaciones temporales oportunas (ver cualquier score.srt), almacenada internamente en este último formato. Se crea bien en línea, bien leída de un fichero de partitura existente (en cualquier formato). Sus componentes principales son un opcode y un array de campos p. Es almacenada en algún lugar en la memoria y organizada por una estructura que empieza de la siguiente manera:

```
typedef struct {
    CSHDR h;           /* cabecera que organiza el espacio */
    long op;          /* opcode—t, w, f, i, a, s o e */
    long pcnt;        /* número de campos p p1, p2, p3 ... */
    long strlen;     /* longitud de la cadena del argumento opcional */
    char *strarg;    /* dirección de la cadena del argumento opcional */
    float p2orig;    /* p2, p3 sin modificar*/
    float p3orig;
    float offtim;    /* almacenamiento durante la ejecución */
    float p[1];      /* array de campos p p0, p1, p2 ... */
} EVENT;
```

Cualquier función que cree, lea o copie un evento devolverá un puntero a la estructura de almacenamiento que contenga los datos del evento. El puntero al evento puede ser usado para acceder a cualquier componente de la estructura, en la forma *e-op* o *e-p[n]*. Cada nuevo evento almacenado dará lugar a un nuevo puntero y una secuencia de nuevos eventos generará una secuencia de distintos punteros que deben ser también almacenados. Los grupos de punteros a eventos se almacenan en una lista de eventos, que tiene su propia estructura:

```
typedef struct {
    CSHDR h;
    int nslots;      /* máximo número de eventos en la lista */
    int nevents;    /* número de eventos presentes */
    EVENT *e[1];    /* array de punteros a eventos e0, e1, e2.. */
} EVLIST;
```

Cualquier función que cree o modifique una lista devolverá un puntero a la nueva lista. El puntero a la lista puede ser usado para acceder cualquiera de los punteros a eventos que la componen, en la forma *a-e[n]*. Los punteros a eventos y los punteros a listas son principalmente herramientas para manipular los datos de un fichero partitura. Los punteros y las listas de punteros pueden ser copiados y reordenados sin modificar el valor de los datos a los que apuntan. Alternativamente, los datos dentro de un evento o grupo de eventos pueden ser modificados sin cambiar los punteros a evento o los punteros a lista. Las funciones de Cscore permiten crear y manipular las partituras de esta manera.

En el siguiente resumen de las llamadas a función de Cscore, se usan algunos criterios simples para darles nombre:

```
los símbolos e, f son punteros a eventos (notas);
los símbolos a, b son punteros a listas (arrays) de eventos;
las letras ev al final del nombre de una función indican una operación en un evento;
la letra l al principio del nombre de una función indica una operación en una lista;
el símbolo fp es un puntero a un fichero partitura de entrada (FILE *);
```

sintaxis de llamada	descripción
<code>e = createv(n);</code> int n;	crea un evento en blanco con n campos p
<code>e = defev("...");</code>	define un evento por cada carácter de la cadena ...
<code>e = copyev(f);</code>	hace una nueva copia del evento f
<code>e = getev();</code>	lee el siguiente evento del fichero partitura de entrada
<code>putev(e);</code>	escribe un evento e al fichero partitura de salida
<code>putstr("...");</code>	escribe el evento de cadena definido a la partitura
<code>a = lcreat(n);</code> int n;	crea una lista de eventos vacía con n posiciones
<code>a = lappev(a,e);</code>	añade el evento e a la lista a
<code>a = lappstrev(a,"...");</code>	añade el evento de cadena definido a la lista a;
<code>a = lcopy(b);</code>	copia la lista b (pero no los eventos)
<code>a = lcopyev(b);</code>	copia los eventos de b, haciendo una nueva lista
<code>a = lget();</code>	lee todos los eventos de una partitura, hasta un nuevo s o e
<code>a = lgetnext(nbeats);</code> float nbeats;	lee los siguientes nbeats pulsos de la partitura
<code>a = lgetuntil(beatno);</code> float beatno;	lee todos los eventos de la partitura hasta el pulso beatno
<code>a = lsepf(b);</code>	separa las sentencias f de la lista b a la lista a
<code>a = lseptwf(b);</code> list a	separa las sentencias t,w & f de la lista b a la lista a
<code>a = lcat(a,b);</code>	concatena la lista b con la lista a
<code>lsort(a);</code>	ordena la lista cronológicamente según los valores de p2
<code>a = lxins(b,"...");</code>	extrae notas de un instrumento (no eventos nuevos)
<code>a = lxtimev(b,from,to);</code> float from, to;	extrae notas del lapso de tiempo, creando eventos nuevos
<code>lput(a);</code>	escribe los eventos de una lista a a la partitura de salida
<code>lplay(a);</code>	envía los eventos de una lista a a la orquesta de Csound para su ejecución inmediata (o imprime los eventos si no hay orquesta)
<code>relev(e);</code>	libera el espacio del evento e
<code>lrel(a);</code>	libera el espacio de una lista a (pero no los eventos)
<code>lrelev(a);</code>	libera los eventos de una lista a, y el espacio de la lista
<code>fp = getcurfp();</code>	consigue el puntero a la partitura de entrada activa actual (inicialmente encuentra el puntero de la línea de comando del fichero de entrada)
<code>fp = filopen("filename");</code>	abre otra partitura de entrada (máximo 5)
<code>setcurfp(fp);</code>	hace fp el puntero a la partitura de entrada activa actual
<code>filclose(fp);</code>	cierra la partitura referenciada por FILE *fp

73.2 ESCRIBIENDO UN PROGRAMA PRINCIPAL

El formato general para un programa de control es:

```
#include "cscore.h"
cscore()
{
/* DECLARACIONES DE VARIABLES */
/* CUERPO DEL PROGRAMA */
}
```

La sentencia `include` definirá las estructuras para los eventos y las listas del programa. El siguiente programa en C leerá una *partitura numérica estándar*, hasta la primera sentencia `s` o `e` (excluidas éstas), escribiendo posteriormente los datos (sin alterar) en la salida.

```
#include "cscore.h"
cscore()
{
EVLIST *a;          /* se permite que a apunte a una lista de eventos */
a = lget();         /* lee eventos y devuelve el puntero a la lista */
lput(a);           /* escribe estos eventos(sin alterar) en la salida */
putstr("e");       /* escribe la cadena e en la salida */
}
```

Después de la ejecución de `lget()`, la variable `a` apunta a una lista de direcciones de eventos, cada una de las cuales apunta a un evento almacenado. Hemos usado el mismo puntero para permitir a otra función de lista (`lput`) que lea y escriba todos los eventos que fueron leídos. Si ahora definimos otro símbolo `e` como un puntero a un evento, entonces la sentencia:

```
e = a-e[4];
```

pondrá en él el contenido de la cuarta posición en la estructura `evlist`. El contenido es un puntero a un evento, que a su vez se compone de un array de valores de campos-p. Así, el término `e-p[5]` significará el valor del campo-p número 5 del cuarto evento en `evlist` (`a`). El programa de abajo multiplicará el valor de ese campo-p por 2 antes de escribirlo en la salida.

```
#include "cscore.h"
cscore()
{
EVENT *e;          /* un puntero a evento */
EVLIST *a;
a = lget();        /* lee una partitura como una lista de eventos */
e = a-e[4];        /* apunta al evento 4 en la lista a */
e-p[5] *= 2;       /* encuentra el campo-p 5, y multiplica su valor por 2 */
lput(a);          /* escribe la lista de eventos en la salida */
putstr("e");      /* añade una sentencia de final de partitura */
}
```

Ahora consideremos la siguiente partitura, en la que $p[5]$ contiene la frecuencia en Hz:

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
e
```

Si esta partitura fuera dada al programa precedente el resultado de salida sería:

```
f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
i 1 1 3 0 440 10000
i 1 4 3 0 512 10000 ; p[5] se ha convertido en 512 en vez 256
i 1 7 3 0 880 10000
e
```

Observa que el cuarto evento es de hecho la segunda nota de la partitura. Hasta ahora no habíamos distinguido entre notas y la configuración de tablas de función en una partitura numérica. Ambas pueden ser clasificadas como eventos. Observa también que nuestro cuarto evento ha sido almacenado en $e[4]$ en la estructura. Por compatibilidad con la notación de campos- p de Csound, ignoramos $p[0]$ y $e[0]$ de las estructuras de eventos y listas, guardando $p1$ en $p[1]$, evento 1 en $e[1]$, etc. Las funciones de Cscore adoptan todas este criterio.

Como extensión de lo anterior, podríamos decidir usar a y e para examinar cada uno de los eventos en la lista. Observa que e no almacena el número 4, sino el contenido de esa posición. Para inspeccionar $p5$ de las lista de eventos previa sólo necesitamos redefinir e con la asignación:

```
e = a-e[3];
```

Más generalmente, si declaramos una nueva variable f como un puntero a un puntero a un evento:

```
f = &a-e[4];
```

asignará a f la dirección del cuarto evento en la lista de eventos a , y $*f$ indicará el contenido de la posición, es decir el puntero al evento mismo. La expresión:

```
(*f)-p[5],
```

como $e-p[5]$, indica el quinto campo- p del evento seleccionado. Sin embargo, podemos avanzar a la próxima posición en **evlist** incrementando el puntero f . En C esto se indica por $f++$.

En el siguiente programa usaremos la misma partitura de entrada. Esta vez separaremos las sentencias *f*table de las sentencias *note*. A continuación escribiremos los tres eventos de nota almacenados en la lista a , luego crearemos una segunda sección de partitura consistente en la serie de alturas originales y una versión transportada de ésta. Esto dará lugar una duplicación a la octava.

Haciendo que la variable f apunte al primer evento de nota e incrementando f dentro de un bucle **while** que se itera n veces (el número de eventos de la lista), una sentencia puede ser configurada para actuar sobre el mismo campo- p de cada evento sucesivo:

```

#include "cscore.h"
cscore()
{
EVENT *e,**f;          /* declaraciones. ver pp.8-9 en el */
EVLIST *a,*b;         /* manual de programación en C */
int n;
a = lget();           /* lee la partitura en la lista a */
b = lsepf(a);         /* separa las sentencias f */
lput(b);              /* escribe las sentencias f en la partitura de salida*/
lrelev(b);           /* y libera el espacio usado */
e = defev("t 0 120"); /* define un evento para una sentencia de tiempo */
putev(e);            /* escribe la sentencia de tiempo en la partitura */
lput(a);              /* escribe las notas */
putstr("s");          /* final de sección */
putev(e);            /* escribe la sentencia de tiempo otra vez */
b = lcopyev(a);       /* hace una copia de las notas en a */
n = b-nevents;       /* y consigue el número presente */
f = &a-e[1];
while (n--)           /* repite n veces la siguiente línea */
(*f++)-p[5] *= .5;    /* transporta la altura una octava abajo */
a = lcat(b,a);        /* añade estas notas a las originales*/
lput(a);
putstr("e");
}

```

La salida de este programa es la siguiente:

```

f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
s
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
i 1 1 3 0 220 10000
i 1 4 3 0 128 10000
i 1 7 3 0 440 10000
e

```

A continuación ampliaremos el programa anterior usando la sentencia `while` para mirar a `p[5]` y `p[6]`. En la partitura original `p[6]` indica amplitudes. Para crear un `diminuendo` en la octava inferior añadida, que es independiente de la serie original de notas, usaremos una variable llamada `dim`.

```

#include "cscore.h"
cscore()
{
EVENT *e,**f;
EVLIST *a,*b;
int n, dim;          /* declara dos variables enteras */
a = lget();
b = lsepf(a);
lput(b);
lrelev(b);
e = defev("t 0 120");
putev(e);
lput(a);
putstr("s");
putev(e);           /* escribe otra sentencia de tiempo en la salida */
b = lcopyev(a);
n = b-nevents;
dim = 0;            /* inicializa dim a 0 */

```

```

f = &a-e[1];
while (n--){
(*f)-p[6] -= dim;      /* resta el valor actual de dim */
(*f++)-p[5] *= .5;    /* transporta y mueve f al siguiente evento */
dim += 2000;         /* aumenta dim para cada nota */
}
a = lcat(b,a);
lput(a);
putstr("e");
}

```

El incremento de *f* en los programas precedentes ha dependido de ciertas reglas de precedencia del lenguaje C.

Aunque hace al código compacto, esta práctica puede ser peligrosa para los principiantes. El incremento puede escribirse como una sentencia separada para hacerlo más claro.

```

while (n--){
(*f)-p[6] -= dim;
(*f)-p[5] *= .5;
dim += 2000;
f++;
}

```

Usando de nuevo la misma partitura de entrada, la salida de este programa sería:

```

f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
s
t 0 120
i 1 1 3 0 440 10000 ; tres notas originales
i 1 4 3 0 256 10000 ; pulsos 1,4 y 7 sin dim.
i 1 7 3 0 880 10000
i 1 1 3 0 220 10000 ; tres notas transportadas una octava abajo
i 1 4 3 0 128 8000 ; también en los pulsos 1,4 y 7 con dim.
i 1 7 3 0 440 6000
e

```

En el próximo programa las mismas tres notas serán repetidas en varios intervalos de tiempo. El instante de comienzo de cada grupo viene determinado por los valores de la cola del array. Este vez *dim* ocurrirá cada grupo de notas en vez de cada nota. Observa la posición de la sentencia que incrementa la variable *dim* fuera del bucle while:

```

#include "cscore.h"
int cue[3]={0,10,17};          /* declara un array de 3 elementos */
cscore()
{
EVENT *e, **f;
EVLIST *a, *b;
int n, dim, cuecount, holdn; /* declara nuevas variables */
a = lget();
b = lsepf(a);
lput(b);
lreleev(b);
e = defev("t 0 120");
putev(e);
n = a-nevents;
holdn = n;                    /* mantiene el valor de "n" */
cuecount = 0;                 /* inicializa cuecount a "0" */
dim = 0;
while (cuecount <= 2) {      /* cuenta tres repeticiones del bucle "while"*/
f = &a-e[1];                 /* resetea el puntero al primer */
/* evento de la lista "a" */
n = holdn;                   /* resetea el valor de "n" al */
/* notecount original */

while (n--) {
(*f)-p[6] -= dim;
(*f)-p[2] += cue[cuecount]; /* añade valores a la cola */
f++;
}
printf("; diagnostic: cue = %d\n", cue[cuecount]);
cuecount++;
dim += 2000;
lput(a);
}
putstr("e");
}

```

Aquí el bucle while interior mira a los eventos de una lista a (las notas) y el bucle exterior mira cada repetición de los eventos de la lista a (las repeticiones del grupo de alturas). Este programa también muestra un mecanismo útil para encontrar problemas con la sentencia **printf**. El carácter ; es el primero en la cadena para producir una sentencia de comentario en la partitura resultante. En este caso el valor de la cola se imprime en la salida para asegurar que el programa está tratando con el elemento del array correcto en el instante apropiado. Cuando los datos de salida son erróneos o se encuentran mensajes de error, la función **printf** puede ayudarte a precisar el problema.

Usando el mismo fichero de entrada de siempre, el programa en C anterior generará:

```

f 1 0 257 10 1
f 2 0 257 7 0 300 1 212 .8
t 0 120
; diagnostic: cue = 0
i 1 1 3 0 440 10000
i 1 4 3 0 256 10000
i 1 7 3 0 880 10000
; diagnostic: cue = 10
i 1 11 3 0 440 8000
i 1 14 3 0 256 8000
i 1 17 3 0 880 8000

```

```
; diagnostic: cue = 17
i 1 28 3 0 440 4000
i 1 31 3 0 256 4000
i 1 34 3 0 880 4000
e;
```

73.3 EJEMPLOS MÁS AVANZADOS

El siguiente programa muestra la lectura de dos ficheros de entrada diferentes. La idea es intercambiar dos secciones de la partitura y escribir en la salida las secciones en un único fichero.

```
./htmlinclude "cscore.h"          /* CSCORE_SWITCH.C */
cscore()                          /* puede ser llamado desde Csound */
                                  /* o como un programa de Cscore aislado */
{
EVLIST *a, *b;
FILE *fp1, *fp2;                  /* dos punteros a dos ficheros de entrada */
fp1 = getcurfp();                 /* esta es la partitura de línea de la comando */
fp2 = filopen("score2.srt");      /* esta es una partitura adicional */
a = lget();                       /* lee la sección de la partitura 1 */
lput(a);                          /* la escribe en la salida tal cual */
putstr("s");
setcurfp(fp2);
b = lget();                       /* lee la sección de la partitura 2 */
lput(b);                          /* la escribe en la salida tal cual */
putstr("s");
lrelev(a);                        /* opcional para pedir espacio */
lrelev(b);
setcurfp(fp1);
a = lget();                       /* lee la siguiente sección de la partitura 1 */
lput(a);                          /* la escribe en la salida */
putstr("s");
setcurfp(fp2);
b = lget();                       /* lee la siguiente sección de la partitura 2 */
lput(b);                          /* la escribe en la salida */
putstr("e");
}
```

Finalmente, mostraremos cómo tomar literalmente un fichero partitura sin interpretar y aplicarles algunos cambios de tiempo expresivos. La teoría de los pulsos métricos relacionados con los compositores ha sido investigada en profundidad por Manfred Clynes, y lo siguiente se inspira en su obra. La estrategia aquí es primero crear un array de nuevos instantes de comienzo para cada posible semicorchea y luego indexarlo para ajustar el comienzo y la duración de cada nota de la partitura de entrada a los instantes de comienzo calculados. Esto también muestra cómo una orquesta de Csound puede ser invocada repetidamente desde un generador de partitura como éste.

```
./htmlinclude "cscore.h"          /* CSCORE_PULSE.C */
                                  /* programa para aplicar un tempo musical */
                                  /* a una partitura existente en 3/4 */
                                  /* primeros pulsos en 0, 3, 6 ... */
static float four[4] = { 1.05, 0.97, 1.03, 0.95 };
                                  /* anchura de pulso para 4's*/
static float three[3] = { 1.03, 1.05, .92 };
                                  /* anchura de pulso para 3's*/
cscore()                          /* se puede llamar desde Csound o independientemente */
{
EVLIST *a, *b;
register EVENT *e, **ep;
float pulse16[4*4*4*4*3*4];      /* array de semicorcheas, 3/4, 256 compases */
float acc16, acc1,inc1, acc3,inc3, acc12,inc12, acc48,inc48,acc192,inc192;
register float *p = pulse16;
register int n16, n1, n3, n12, n48, n192;
                                  /* llena el array con los instantes de comienzo */
for (acc192=0.,n192=0; n192<4; acc192+=192.*inc192,n192++)
for (acc48=acc192,inc192=four[n192],n48=0; n48<4; acc48+=48.*inc48,n48++)
for (acc12=acc48,inc48=inc192*four[n48],n12=0;n12<4; acc12+=12.*inc12,n12++)
for (acc3=acc12,inc12=inc48*four[n12],n3=0; n3<4; acc3+=3.*inc3,n3++)
for (acc1=acc3,inc3=inc12*four[n3],n1=0; n1<3; acc1+=inc1,n1++)
```

```

for (accl6=accl, incl=inc3*three[nl], n16=0; n16<4; accl6+=.25*incl*four[n16], n16++)
*p++ = accl6;

/* for (p = pulse16, n1 = 48; n1--; p += 4) /* show vals & diffs */
/* printf("%g %g %g %g %g %g %g %g\n", *p, *(p+1), *(p+2), *(p+3),
/* *(p+1)-*p, *(p+2)-*(p+1), *(p+3)-*(p+2), *(p+4)-*(p+3)); */

a = lget(); /* lee una sección de la partitura con los cambios de tempo*/
b = lseptwf(a); /* separa las sentencias t y f */
lplay(b); /* y manda estas para su ejecución */
a = lappstrev(a, "s"); /* añade una sentencia de sección a la lista de notas */
lplay(a); /* ejecuta la lista de notas sin cambios de tempo */
for (ep = &a-e[1], n1 = a-nevents; n1--; ) {
/* ahora modifica la duración de los pulsos */

e = *ep++;
if (e-op == 'i') {
e-p[2] = pulse16[(int)(4. * e-p2orig)];
e-p[3] = pulse16[(int)(4. * (e-p2orig + e-p3orig))] - e-p[2];
}
lplay(a); /* ejecuta la lista modificada */
}

```

Como se dijo arriba, los ficheros de entrada de Cscore pueden estar en formato preordenado o con los tempi ya modificados. Esta modalidad será preservada (sección a sección) durante la lectura, el procesado y la escritura de partituras. El procesado independiente de Cscore será a menudo más útil con partituras con los tempi no aplicados aún. Cuando se ejecuta desde Csound la partitura de entrada llegará con los cambios de tempo aplicados y ordenada, para poder ser así directamente enviada (normalmente sección a sección) a la orquesta.

Una lista de eventos pueden ser llevada a una orquesta de Csound usando **lplay**. Puede haber cualquier número de llamadas a **lplay** en un programa de Cscore. Cada lista enviada de esta manera puede tener los cambios de tempo aplicados o no, pero cada lista debe estar en estricto orden cronológico según los valores de p2 (bien usando el proceso de preordenamiento, bien usando **lsort**). Si no hay **lplay** en un módulo de Cscore que se ejecute desde Csound, todos los eventos escritos en la salida (via *putev*, *putstr* o *lput*) constituyen una nueva orquesta, que será enviada inicialmente a **scsort** y luego a la orquesta de Csound para su ejecución. Estos eventos pueden ser examinados en los ficheros '*cscore.out*' y '*cscore.srt*'.

Un programa Cscore aislado usará normalmente los comandos *put* para escribir en su fichero de salida. Si uno de tales programas contiene algún **lplay**, los eventos entendidos para ser ejecutados serán impresos en pantalla.

Una lista de notas enviada por **lplay** para que sea ejecutada debe ser temporalmente distinta de las siguientes listas de notas. Ningún final de nota debería extenderse más allá del instante de comienzo de la próxima lista, ya que **lplay** completará cada lista antes de empezar con la siguiente (es decir, como una marca de sección que no resetea su reloj local a 0). Esto es importante cuando se están usando **lgetnext()** o **lgetuntil()** buscar y procesar segmentos de partitura antes de la ejecución.

73.4 COMPILANDO UN PROGRAMA DE CSCORE

Un programa de Cscore puede ser invocado bien como un programa independiente o como parte de Csound.

```
cscore nombrepartitura ficherosalida
o
CSound -C [otros indicadores] nombreorquesta nombrepartitura
```

Para crear un programa independiente, escribe un programa **cscore.c** como se muestra arriba y compilalo con '`cc cscore.c`'. Si el compilador no puede encontrar "`cscore.h`", intenta usar el indicador `/usr/local/include`, o copia el módulo `cscore.h` del directorio donde esté el código fuente de Csound en el directorio actual. Aún habrá referencias no resueltas, así que debes enlazar tu programa con ciertos módulos de entrada/salida de Csound. Si la instalación de Csound ha creado `libcscore.a`, puedes teclear:

```
cc -o cscore cscore.c -lcscore
```

Si no, define una variable de entorno con el directorio de Csound que contenga los módulos ya compilados, e invócalos explícitamente:

```
setenv CSOUND /ti/u/bv/CSound
cc -o cscore cscore.c $CSOUND/cscoremain.o $CSOUND/cscorefns.o \
$CSOUND/rdscore.o $CSOUND/memalloc.o
```

El fichero ejecutable resultante puede ser aplicado a cualquier fichero de entrada tecleando:

```
cscore scorefilein scorefileout
```

Para operar desde dentro de Csound, primero procede como arriba y luego enlaza tu programa con la serie completa de módulos de Csound. Si tu instalación de Csound ha creado `libcsound.a`, puedes hacer esto mediante:

```
cc -o mycsound cscore.c -lcsound -lX11 -lm (X11 si tu instalación lo incluye)
```

Si no, copia `*.c`, `*.h` y `Makefile` del directorio fuente de Csound, reemplaza `cscore.c` por el tuyo propio, y luego ejecuta '`make CSound`'. El ejecutable resultante es tu especial Csound propio, listo para usar como de costumbre. El indicador `-C` invocará tu programa de Cscore después de que el fichero partitura de entrada haya sido ordenado en '`score.srt`'. Si nohay ningún **lplay**, las siguientes etapas del procesado pueden ser vistas en los ficheros '`cscore.out`' y '`cscore.srt`'.

74 AÑADIENDO TUS PROPIOS CMÓDULOS A CSOUND

Si los generadores existentes en Csound no se ajustan a tus necesidades, puedes escribir tus propios módulos en C y luego añadirlos al sistema de ejecución. Cuando invocas Csound sobre una orquesta y una partitura, la orquesta se lee primero mediante un traductor de búsqueda en tablas "otran" y los bloques de instrumento son convertidos a plantillas de código listas para ser cargadas en memoria por "oload" a petición del lector de partituras. Para usar tus propios módulos con una orquesta estándar necesitas añadir sólo una entrada en la tabla de "otran" y reenlazar Csound a tu código.

El traductor, el cargador y monitor de ejecución tratarán tu módulo como cualquier otro, con tal que sigas algunos convenios. Necesitas una *estructura* que defina las entradas, las salidas y el espacio de trabajo, más algún *código de inicialización* y algún *código de ejecución*. Pongamos un ejemplo usando dos nuevos ficheros, **newgen.h** y **newgen.c**:

```
typedef struct {                               /* newgen.h - define una estructura*/
OPDS                                           /* cabecera requerida */
h;                                             /* añade argumentos de E/S */
float *result, *istrt, *incr, *itime, *icontin; /* espacio de trabajo privado */
float curval, vincr;                          /* ditto */
long countdown;
} RMP;
#include "cs.h"                                /* newgen.c - código de inicialización
                                              /* y ejecución */

#include "newgen.h"
void rampset(RMP *p)                          /* en la inicialización de la nota */
{
if (*p-icontin == 0.)                         /* opcionalmente consigue el valor
                                              /* inicial */
p-curval = *p-istrt;                          /* ajusta la frecuencia */
p-vincr = *p-incr / esr;                     /* contador por itime segundos */
p-countdown = *p-itime * esr;
}
void ramp(RMP *p)                             /* durante la ejecución de la nota */
{
float *rsltp = p-result;                    /* inicializa un puntero a un array
                                              /* de salida*/
int nn = ksmpls;                             /* tamaño del array, de la orquesta*/
do {
*rsltp++ = p-curval;                         /* copia valor actual en la salida*/
if (--p-countdown = 0)                       /* durante los primeros itime seg. */
p-curval += p-vincr;                         /* hace una rampa con el valor */
} while (--nn);
}
```

Ahora añadimos este módulo a la tabla del traductor **entry.c**, bajo el nombre del opcode, **rampt**:

```
#include "newgen.h"
void rampset(), ramp();

/* opcode dspace thread outarg inargs isub ksub asub */
{ "ramp", S(RMP), 5, "a", "iio", rampset, NULL,
ramp },
```

Finalmente enlazamos de nuevo Csound para que incluya el nuevo módulo. Si tu instalación creó *libcsound.a*, puedes hacer esto tecleando:

```
cc -o mycsound newgen.c entry.c -lcsound -lX11 -lm
(X11 si se incluía en la instalación)
```

Si no, copia **.c*, **.h* y *Makefile* del directorio fuente de Csound, añade **newgen.o** a la lista de objetos de Makefile, añade **newgen.h** a la lista de **entry.o** y añade también '**newgen.o: newgen.h**' y ejecuta entonces "*make Csound*". Si tu ordenador es un Macintosh, simplemente añade **newgen.h** y **newgen.c** a uno de los segmentos en el proyecto Csound, e invoca el compilador C.

Las acciones anteriores habrán añadido un nuevo generador al lenguaje de Csound. Es la función de una rampa lineal a frecuencia de audio que modifica un valor de entrada a una determinada pendiente durante cierto tiempo. Una rampa puede continuar opcionalmente desde el último valor de la nota precedente. La entrada en el manual de Csound sería:

```
ar rampt istart, islope, itime [, icontin]
```

istart - valor de inicio de una rampa lineal a frecuencia de audio. Opcionalmente sustituido por un indicador de continuación.

islope - pendiente de la rampa, expresada como la variación en segundos en el eje y.

itime - duración de la rampa en segundos, después de la cual el valor es mantenido durante el resto de la nota.

icontin (opcional) - indicador de continuación. Si es cero, la rampa empezará desde el valor de entrada *istart*. Si no es 0, la rampa empezará desde el último valor de la nota precedente. El valor por defecto es 0.

El fichero *newgen.h* incluye una lista de una línea de los parámetros de entrada y salida. Estos son los puertos a través de los cuales el nuevo generador se comunicará con otros generadores en un instrumento.

La comunicación se realiza por *direcciones*, no por *valores*, y esta lista contiene punteros a valores float. No hay restricciones en los nombres, pero los tipos de argumento de entrada-salida son definidos como cadenas de caracteres en **entry.c** (argumentos de entrada-salida). Los argumentos de entrada *inarg* son normalmente **x**, **a**, **k**, e **i**, según el convenio estándar en Csound. También están disponibles los tipos **o** (opcional, con valor por defecto 0), **p** (opcional, con valor por defecto 1).

Los tipos de salida *outarg* incluyen **a**, **k**, **i** y **s** (asig o ksig). Es importante asignar a todos los nombres de argumentos listados un tipo correspondiente de argumento en **entry.c**. Los argumentos de tipo *i*- son sólo válidos en tiempo de inicialización y los demás tipos son sólo válidos en tiempo de ejecución. Las líneas siguientes en la estructura RMP declaran el espacio de trabajo necesario para guardar el código en cada entrada. Esto permite al módulo ser usado múltiples veces en múltiples copias de instrumentos, mientras se preservan todos los datos de cada llamada.

El fichero *newgen.c* contiene dos subrutinas, cada una de las cuales es llamada por un puntero a la estructura RMP asignada y todos sus datos. Las subrutinas pueden ser de tres tipos: de inicialización de nota, de generación de señales a frecuencia de control, de generación de señales a frecuencia de audio. Un módulo normalmente requiere dos de estas subrutinas de inicialización y bien una subrutina a frecuencia

de control, bien una subrutina a frecuencia de audio que se insertan en varias listas enlazadas de tareas de ejecución cuando un instrumento es activado. Los tipos de enlace aparecen en **entry.c** en dos formas: nombres *isub*, *ksub* y *asub* y un índice de enlace que es la suma de *isub*=1, *ksub*=2, *asub*=4. El código mismo puede referenciar variables globales, definidas en **cs.h** y **oload.c**. Las más útiles de estas variables globales son:

```
extern OPARMS O ; float esr
user-defined sampling rate float ekr
user-defined control rate float ensmpts
user-defined ksmps int ksmps
user-defined ksmps int nchnls
user-defined nchnls int O.odebug
command-line -v flag int O.msglevel
command-line -m level float pi, twopi obvious
constants float tpidsr twopi / esr float
sstrcod special code for string arguments
```

74.1.1 TABLAS DE FUNCIÓN

Para acceder a tablas de función almacenadas se requiere ayuda adicional. La nueva estructura definida debe incluir un puntero:

```
FUNC *ftp;
```

inicializado por la sentencia:

```
ftp = ftpfind(p-ifuncno);
```

donde el float **ifuncno* es un argumento de entrada de tipo *i-* que contiene el número de la tabla de función. La tabla almacenada es entonces una *ftp-ftable* y cualquier otro dato como la longitud, la fase, los convertidores "cps a incremento", etc. son también accedidos desde este puntero. Ver la estructura *FUNC* en *cs.h*, el código *ftfind()* en *fgens.c* y el código para *oscset()* y *koscil()* en *ugens2.c*.

74.1.2 ESPACIO ADICIONAL

Algunas veces la demanda de espacio de un módulo es demasiado grande para ser parte de una estructura (el límite superior es de 65535 bytes) o depende del valor de un argumento de tipo *i-* que no se conoce hasta el instante de la inicialización. Se puede asignar y manejar adecuadamente espacio adicional incluyendo la línea:

```
AUXCH auxch;
```

en la estructura definida (**p*). Luego hay que usar el siguiente código en el módulo de inicialización:

```
if (p-auxch.auxp == NULL)
    auxalloc(npoints * sizeof(float), &p-auxch);
```

La dirección de este espacio adicional es guardada en una cadena de tales espacios que pertenecen al instrumento y es automáticamente manejada mientras el instrumento está siendo duplicado o reciclado durante la ejecución. La asignación:

```
char *auxp = p-auxch.auxp;
```

encontrará el espacio asignado para el uso en tiempo de inicialización y ejecución. Ver la estructura de **linseg** en *ugens1.h* y el código para *lsgset()* y *klseg()* en *ugens1.c*.

74.1.3 COMPARTIENDO FICHEROS

Cuando se accede a un fichero externo a menudo, o desde varios lugares, es normalmente mejor mantener el fichero entero en memoria. Esto se consigue incluyendo la línea:

```
MEMFIL *mfp;
```

en la estructura definida (*p). Luego hay que usar este código en el módulo de inicialización:

```
if (p-mfp == NULL)
p-mfp = ldmemfile(filename);
```

donde el char *filename es el nombre de una cadena del fichero requerido. Los datos leídos serán encontrados entre:

```
(char *) p-mfp-beginp; y (char *) p-mfp-endp;
```

Los ficheros cargados no pertenecen a ningún instrumento en particular, pero son automáticamente compartidos por múltiples accesos. Ver la estructura de **adsyn** en `ugens3.h` y el código para `adset()` y `adsyn()` en `ugens3.c`.

74.1.4 CADENAS COMO ARGUMENTOS

Para permitir a una cadena entre comillas ser un argumento de entrada (digamos float *ifilnam) en nuestra estructura definida (*p), asignala como un argumento de tipo **S** en **entry.c**, incluye otro elemento char *strarg en la estructura, inserta una línea:

```
TSTRARG( "rampt", RMP) \
```

en el fichero **oload.h**, e incluye el siguiente código en el módulo de inicialización:

```
if (*p-ifilnam == sstrcod)
strcpy(filename, unquote(p-strarg));
```

Ver el código para `adset()` en `ugens3.c`, `lprdset()` en `ugens5.c`, y `pvset()` en `ugens8.c`.

75 APÉNDICE A: INFORMACIÓN MISCELÁNEA

75.1 TABLA 1: Conversión de Alturas

Note	CPS	cpspch	MIDI	Note	CPS	cpspch	MIDI
C0	8.176	3.00	0	E5	329.628	8.04	64
C#0	8.662	3.01	1	F5	349.228	8.05	65
D0	9.177	3.02	2	F#5	369.994	8.06	66
D#0	9.723	3.03	3	G5	391.995	8.07	67
E0	10.301	3.04	4	G#5	415.305	8.08	68
F0	10.913	3.05	5	A5	440.000	8.09	69
F#0	11.562	3.06	6	A#5	466.164	8.10	70
G0	12.250	3.07	7	B5	493.883	8.11	71
G#0	12.978	3.08	8	C6	523.251	9.00	72
A0	13.750	3.09	9	C#6	554.365	9.01	73
A#0	14.568	3.10	10	D6	587.330	9.02	74
B0	15.434	3.11	11	D#6	622.254	9.03	75
C1	16.352	4.00	12	E6	659.255	9.04	76
C#1	17.324	4.01	13	F6	698.456	9.05	77
D1	18.354	4.02	14	F#6	739.989	9.06	78
D#1	19.445	4.03	15	G6	783.991	9.07	79
E1	20.602	4.04	16	G#6	830.609	9.08	80
F1	21.827	4.05	17	A6	880.000	9.09	81
F#1	23.125	4.06	18	A#6	932.328	9.10	82
G1	24.500	4.07	19	B6	987.767	9.11	83
G#1	25.957	4.08	20	C7	1046.502	10.00	84
A1	27.500	4.09	21	C#7	1108.731	10.01	85
A#1	29.135	4.10	22	D7	1174.659	10.02	86
B1	30.868	4.11	23	D#7	1244.508	10.03	87
C2	32.703	5.00	24	E7	1318.510	10.04	88
C#2	34.648	5.01	25	F7	1396.913	10.05	89
D2	36.708	5.02	26	F#7	1479.978	10.06	90
D#2	38.891	5.03	27	G7	1567.982	10.07	91
E2	41.203	5.04	28	G#7	1661.219	10.08	92
F2	43.654	5.05	29	A7	1760.000	10.09	93
F#2	46.249	5.06	30	A#7	1864.655	10.10	94
G2	48.999	5.07	31	B7	1975.533	10.11	95
G#2	51.913	5.08	32	C8	2093.005	11.00	96
A2	55.000	5.09	33	C#8	2217.461	11.01	97
A#2	58.270	5.10	34	D8	2349.318	11.02	98
B2	61.735	5.11	35	D#8	2489.016	11.03	99
C3	65.406	6.00	36	E8	2637.020	11.04	100
C#3	69.296	6.01	37	F8	2793.826	11.05	101
D3	73.416	6.02	38	F#8	2959.955	11.06	102
D#3	77.782	6.03	39	G8	3135.963	11.07	103
E3	82.407	6.04	40	G#8	3322.438	11.08	104
F3	87.307	6.05	41	A8	3520.000	11.09	105
F#3	92.499	6.06	42	A#8	3729.310	11.10	106
G3	97.999	6.07	43	B8	3951.066	11.11	107
G#3	103.826	6.08	44	C9	4186.009	12.00	108
A3	110.000	6.09	45	C#9	4434.922	12.01	109
A#3	116.541	6.10	46	D9	4698.636	12.02	110
B3	123.471	6.11	47	D#9	4978.032	12.03	111
C4	130.813	7.00	48	E9	5274.041	12.04	112
C#4	138.591	7.01	49	F9	5587.652	12.05	113
D4	146.832	7.02	50	F#9	5919.911	12.06	114
D#4	155.563	7.03	51	G9	6271.927	12.07	115
E4	164.814	7.04	52	G#9	6644.875	12.08	116
F4	174.614	7.05	53	A9	7040.000	12.09	117
F#4	184.997	7.06	54	A#9	7458.620	12.10	118
G4	195.998	7.07	55	B9	7902.133	12.11	119
G#4	207.652	7.08	56	C10	8372.018	13.00	120
A4	220.000	7.09	57	C#10	8869.844	13.01	121
A#4	233.082	7.10	58	D10	9397.273	13.02	122
B4	246.942	7.11	59	D#10	9956.063	13.03	123
C5	261.626	8.00	60	E10	10548.08	13.04	124
C#5	277.183	8.01	61	F10	11175.30	13.05	125
D5	293.665	8.02	62	F#10	11839.82	13.06	126
D#5	311.127	8.03	63	G10	12543.85	13.07	127

75.2 TABLA II: Valores de Intensidad Sonora (para un tono de 1 KHz)

Dinámica	Intensidad (W/m ²)	Nivel (dB)
dolor	1	120
fff	10 ⁻²	100
f	10 ⁻⁴	80
p	10 ⁻⁶	60
ppp	10 ⁻⁸	40
umbral	10 ⁻¹²	0

75.3 TABLA III: Valores de Formantes

f1 f2 f3 f4 f5

soprano "a"

freq (Hz) 800 1150 2900 3900 4950
 amp (dB) 0 -6 -32 -20 -50
 bw (Hz) 80 90 120 130 140

soprano "e"

freq (Hz) 350 2000 2800 3600 4950
 amp (dB) 0 -20 -15 -40 -56
 bw (Hz) 60 100 120 150 200

soprano "i"

freq (Hz) 270 2140 2950 3900 4950
 amp (dB) 0 -12 -26 -26 -44
 bw (Hz) 60 90 100 120 120

soprano "o"

freq (Hz) 450 800 2830 3800 4950
 amp (dB) 0 -11 -22 -22 -50
 bw (Hz) 70 80 100 130 135

soprano "u"

freq (Hz) 325 700 2700 3800 4950
 amp (dB) 0 -16 -35 -40 -60
 bw (Hz) 50 60 170 180 200

alto "a"

freq (Hz) 800 1150 2800 3500 4950
 amp (dB) 0 -4 -20 -36 -60
 bw (Hz) 80 90 120 130 140

alto "e"

freq (Hz) 400 1600 2700 3300 4950
 amp (dB) 0 -24 -30 -35 -60
 bw (Hz) 60 80 120 150 200

alto "i"

freq (Hz) 350 1700 2700 3700 4950
 amp (dB) 0 -20 -30 -36 -60
 bw (Hz) 50 100 120 150 200

alto "o"

freq (Hz) 450 800 2830 3500 4950
 amp (dB) 0 -9 -16 -28 -55
 bw (Hz) 70 80 100 130 135

f1 f2 f3 f4 f5

tenor "a"

freq (Hz) 650 1080 2650 2900 3250
 amp (dB) 0 -6 -7 -8 -22
 bw (Hz) 80 90 120 130 140

tenor "e"

freq (Hz) 400 1700 2600 3200 3580
 amp (dB) 0 -14 -12 -14 -20
 bw (Hz) 70 80 100 120 120

tenor "i"

freq (Hz) 290 1870 2800 3250 3540
 amp (dB) 0 -15 -18 -20 -30
 bw (Hz) 40 90 100 120 120

tenor "o"

freq (Hz) 400 800 2600 2800 3000
 amp (dB) 0 -10 -12 -12 -26
 bw (Hz) 40 80 100 120 120

tenor "u"

freq (Hz) 350 600 2700 2900 3300
 amp (dB) 0 -20 -17 -14 -26
 bw (Hz) 40 60 100 120 120

bass "a"

freq (Hz) 600 1040 2250 2450 2750
 amp (dB) 0 -7 -9 -9 -20
 bw (Hz) 60 70 110 120 130

bass "e"

freq (Hz) 400 1620 2400 2800 3100
 amp (dB) 0 -12 -9 -12 -18
 bw (Hz) 40 80 100 120 120

bass "i"

freq (Hz) 250 1750 2600 3050 3340
 amp (dB) 0 -30 -16 -22 -28
 bw (Hz) 60 90 100 120 120

bass "o"

freq (Hz) 400 750 2400 2600 2900
 amp (dB) 0 -11 -21 -20 -40
 bw (Hz) 40 80 100 120 120

f1 f2 f3 f4 f5

alto "u"

freq (Hz) 325 700 2530 3500 4950
amp (dB) 0 -12 -30 -40 -64
bw (Hz) 50 60 170 180 200

f1 f2 f3 f4 f5

bass "u"

freq (Hz) 350 600 2400 2675 2950
amp (dB) 0 -20 -32 -28 -36
bw (Hz) 40 80 100 120 120

f1 f2 f3 f4 f5

countertenor "a"

freq (Hz) 660 1120 2750 3000 3350
amp (dB) 0 -6 -23 -24 -38
bw (Hz) 80 90 120 130 140

countertenor "e"

freq (Hz) 440 1800 2700 3000 3300
amp (dB) 0 -14 -18 -20 -20
bw (Hz) 70 80 100 120 120

countertenor "i"

freq (Hz) 270 1850 2900 3350 3590
amp (dB) 0 -24 -24 -36 -36
bw (Hz) 40 90 100 120 120

countertenor "o"

freq (Hz) 430 820 2700 3000 3300
amp (dB) 0 -10 -26 -22 -34
bw (Hz) 40 80 100 120 120

countertenor "u"

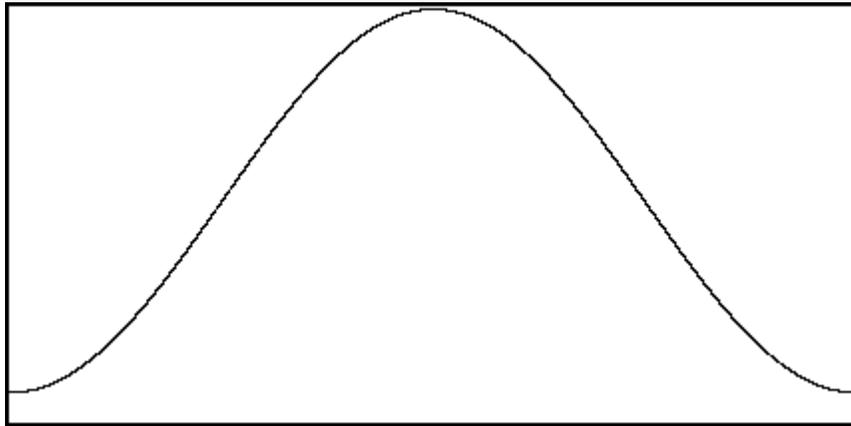
freq (Hz) 370 630 2750 3000 3400
amp (dB) 0 -20 -23 -30 -34
bw (Hz) 40 60 100 120 120

75.4 Funciones de Ventana

Las funciones de ventana son usadas para el análisis y como generadores de envolvente, particularmente en síntesis granular. Las funciones de ventana son incorporadas en algunos opcodes, pero otros requieren una tabla de función para generar la ventana. **GEN20** es la rutina que se usa para este propósito. A continuación se listan las **sentencias f** necesarias para generar algunas de las ventanas más usadas:

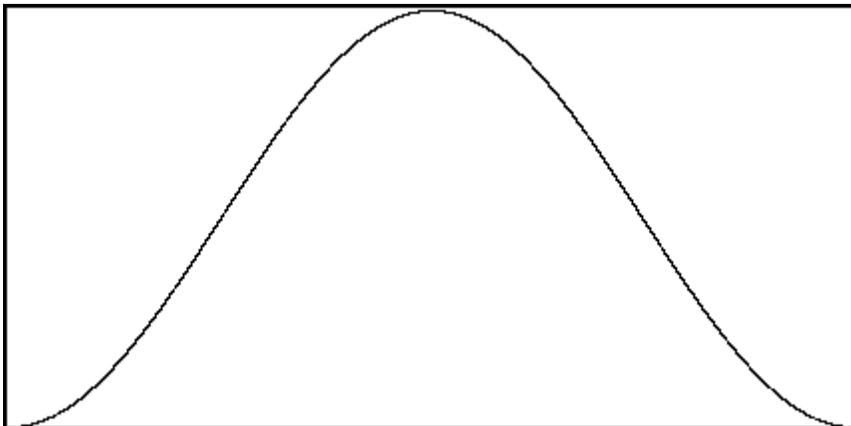
75.4.1 HAMMING

```
f81 0 8192 20 1 1
```



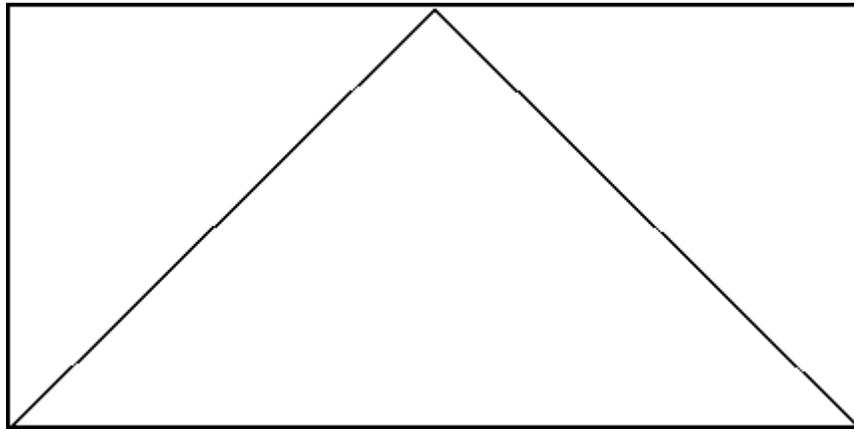
75.4.2 HANNING

```
f82 0 8192 20 2 1
```



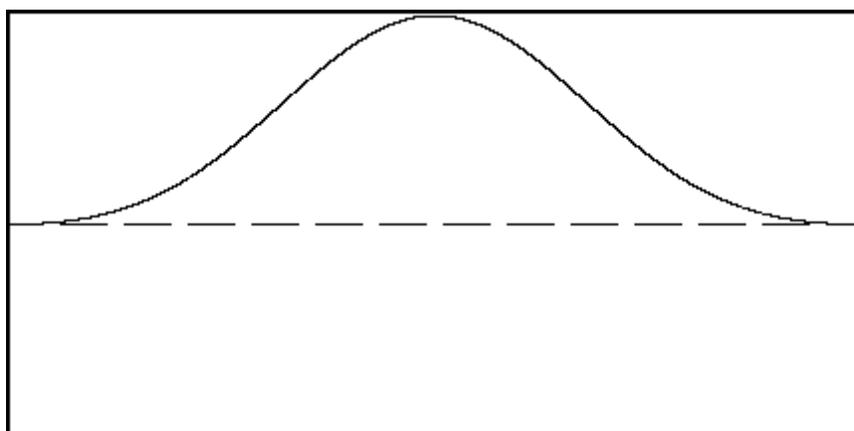
75.4.3 BARTLETT

f83 0 8192 20 3 1



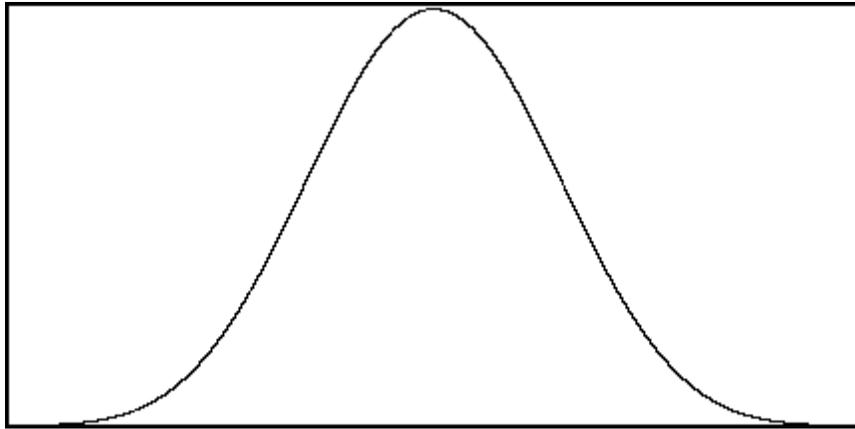
75.4.4 BLACKMAN

f84 0 8192 20 4 1



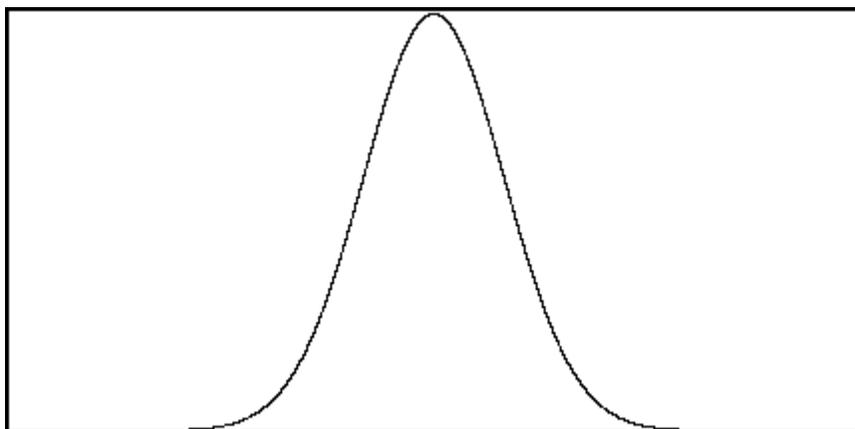
75.4.5 BLACKMAN-HARRIS

f85 0 8192 20 5 1



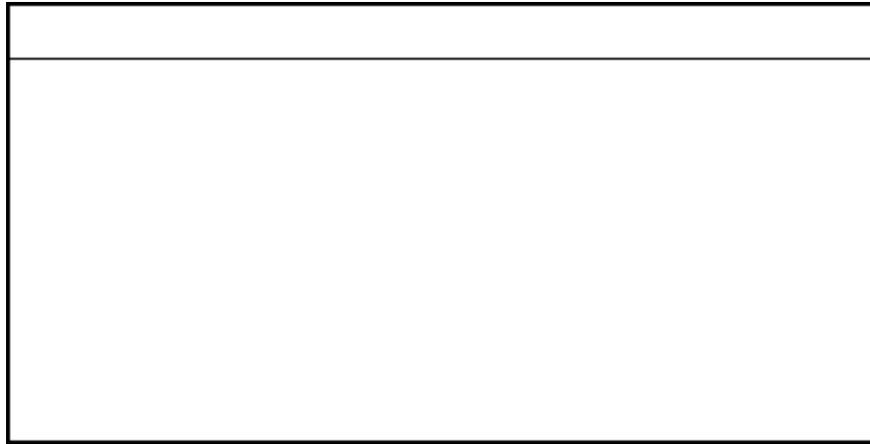
75.4.6 GAUSSIAN

f86 0 8192 20 6 1



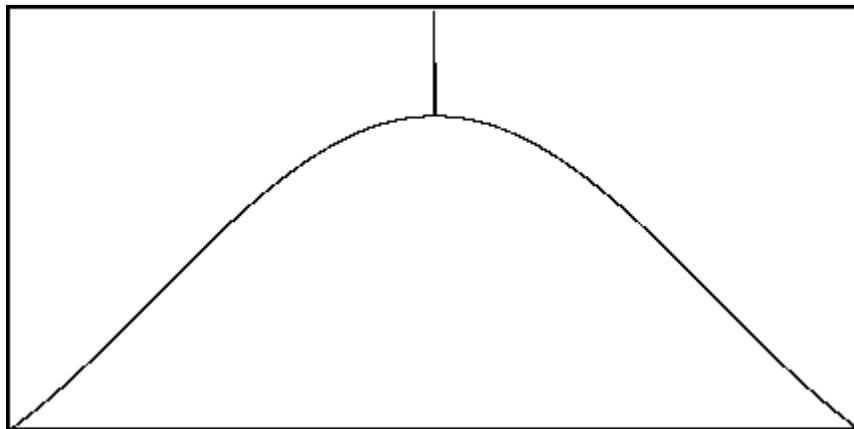
75.4.7 RECTANGLE

f88 0 8192 -20 8 .1



75.4.8 SYNC

f89 0 4096 -20 9 .75



Referencia Rápida de Csound

Sintaxis de la Orquesta: Sentencias de Cabecera de la Orquesta

```
sr          =          iarg
kr          =          iarg
ksmps      =          iarg
nchnls     =          iarg
            strset    iarg, "cadena de caracteres"
            pset      con1, con2, ...
            seed      Ival
gir        ftgen      ifn, itime, isize, igen, iarga[, iargb, ...iargz]
            massign   ichnl, insnum
            ctrlinit  ichnkm, ictlno1, ival1[, ictlno2, ival2[, ictlno3,
            ival3[, ...ival32]]
```

Sintaxis de la Orquesta: Tipos de Variables

```
iname      (variable de inicialización - sólo inicialización)
kname      (señal de control - en la ejecución, frec. de control)
aname      (señal de audio - en la ejecución, frec. de audio)
giname     (variable de inicialización global- sólo inicialización)
gkname     (señal global de control - en la ejecución, frec de control)
ganame     (señal global de audio - en la ejecución, frec. de audio)
wname     (datos espectrales - en la ejecución, frec. de control)
```

Sintaxis de la Orquesta: Sentencias de Bloques de Instrumentos

```
instr      NN
endin
```

Sintaxis de la Orquesta: Inicialización de Variables

```
i/k/ar    =          iarg
i/k/ar    init      iarg
ir        tival
i/k/ar    divz      ia, ib, isubst
```

Control de los Instrumentos: Llamadas a Instrumentos

```
schedule  insnum, iwhen, idur[, p4, p5, ...]
schedwhen ktrigger, kinsnum, kwhen, kdur[, p4, p5, ...]
schedkwhen ktrigger, kmintim, kmaxnum, kinsnum, kwhen, kdur[,
kp4, kp5, ...]
turnon    insnum[, itime]
```


Control de los Instrumentos: Macros

```
#define      NOMBRE # texto de reemplazo #
#define      NOMBRE(a'b'c) # texto de reemplazo #
$NAME.
#undef       NOMBRE
#include     "nombre de fichero"
```

Control de los Instrumentos: Control del Flujo del Programa

```
igoto      label
tigoto     label
kgoto     label
goto      label
if         ia      R      ib      igoto      label
if         ka      R      kb      kgoto      label
if         ia      R      ib      goto       label
timeout    istrtr idur  label
label     :
```

Control de los Instrumentos: Reinicialización

```
reinit     label
rigoto     label
rireturn
```

Operaciones Matemáticas: Operaciones Aritméticas y Lógicas

```
- a        (sin restricción de frecuencia)
+ a        (sin restricción de frecuencia)
a && b     (AND lógico; no a frecuencia de audio)
a || b     (OR lógico; no a frecuencia de audio)
a + b      (sin restricción de frecuencia)
a - b      (sin restricción de frecuencia)
a * b      (sin restricción de frecuencia)
a / b      (sin restricción de frecuencia)
a ^ b      (b no puede estar en frecuencia de audio)
a % b      (sin restricción de frecuencia)
```

Operaciones Matemáticas: Funciones Matemáticas

int(x) (argumentos sólo a frecuencia de inic. o control)
frac(x) (argumentos sólo a frecuencia de inic. o control)
i(x) (argumentos sólo a frecuencia de control)
abs(x) (sin restricción de frecuencia)
exp(x) (sin restricción de frecuencia)
log(x) (sin restricción de frecuencia)
log10(x) (sin restricción de frecuencia)
sqrt(x) (sin restricción de frecuencia)
powoftwo(x) (argumentos sólo a frecuencia de inic. o control)
logbtwo(x) (argumentos sólo a frecuencia de inic. o control)

Operaciones Matemáticas: Funciones Trigonométricas

sin(x) (sin restricción de frecuencia)
cos(x) (sin restricción de frecuencia)
tan(x) (sin restricción de frecuencia)
sininv(x) (sin restricción de frecuencia)
cosinv(x) (sin restricción de frecuencia)
taninv(x) (sin restricción de frecuencia)
sinh(x) (sin restricción de frecuencia)
cosh(x) (sin restricción de frecuencia)
tanh(x) (sin restricción de frecuencia)

Operaciones Matemáticas: Funciones de Amplitud

dbamp(x) (argumentos sólo a frecuencia de inic. o control)
ampdb(x) (sin restricción de frecuencia)

Operaciones Matemáticas: Funciones Aleatorias

rnd(x) (argumentos sólo a frecuencia de inic. o control)
birnd(x) (argumentos sólo a frecuencia de inic. o control)

Operaciones Matemáticas: Opcodes equivalentes a Funciones

ar	sum	asig1, asig2[,asig3...asigN]
ar	product	asig1, asig2[,asig3...asigN]
i/k/ar	pow	i/k/aarg, i/k/pow
i/k/ar	taninv2	i/k/ax, i/k/ay
ar	mac	asig1, ksig1, asig2, ksig2, asig3, ...
ar	maca	asig1, ksig1, asig2, ksig2, asig3, ...

Convertidores de Altura: Funciones

octpch(pch) (argumentos sólo a frecuencia de inic. o control)
pchoct(oct) (argumentos sólo a frecuencia de inic. o control)
cpspch(pch) (argumentos sólo a frecuencia de inic. o control)
octcps(cps) (argumentos sólo a frecuencia de inic. o control)
cpsoct(oct) (sin restricción de frecuencia)

Convertidores de Altura: Opcodes de Afinación

icps **cps2pch** ipch, iequal
icps **cpsxpch** ipch, iequal, irepeat, ibase

Soporte MIDI: Convertidores

ival **notnum**
ival **veloc** [ilow, ihigh]
icps **cpsmidi**
i/kcps **cpsmidib** [irange]
icps **cpstmidi** ifn
ioct **octmidi**
i/koct **octmidib** [irange]
ipch **pchmidi**
i/kpch **pchmidib** [irange]
iamp **ampmidi** iscal[, ifn]
kaft **aftouch** [imin[, imax]]
i/kbend **pchbend** [imin[, imax]]
i/kval **midictrl** inum[, imin[, imax]]

Soporte MIDI: Entrada de Controladores

initc7 ichan, ictrlno, ivalue
initc14 ichan, ictrlno1, ictrlno2, ivalue
initc21 ichan, ictrlno1, ictrlno2, ictrlno3, ivalue
i/kdest **midic7** ictrlno, i/kmin, i/kmax[, ifn]
i/kdest **midic14** ictrlno1, ictrlno2, i/kmin, i/kmax[, ifn]
i/kdest **midic21** ictrlno1, ictrlno2, ictrlno3, i/kmin, i/kmax[, ifn]
i/kdest **ctrl7** ichan, ictrlno, i/kmin, i/kmax[, ifn]
i/kdest **ctrl14** ichan, ictrlno1, ictrlno2, i/kmin, i/kmax[, ifn]
i/kdest **ctrl21** ichan, ictrlno1, ictrlno2, ictrlno3, i/kmin, i/kmax[, ifn]
i/kval **chanctrl** ichnl, ictrlno[, ilow, ihigh]

Soporte MIDI: Bancos de Mandos Deslizantes

i/k1, ..., i/k8	slider8	ichan, ictlnum1, imin1, imax1, init1, ifn1, ..., ictlnum8, imin8, imax8, init8, ifn8
i/k1, ..., i/k16	slider16	ichan, ictlnum1, imin1, imax1, init1, ifn1, ..., ictlnum16, imin16, imax16, init16, ifn16
i/k1, ..., i/k32	slider32	ichan, ictlnum1, imin1, imax1, init1, ifn1, ..., ictlnum32, imin32, imax32, init32, ifn32
i/k1, ..., i/k64	slider64	ichan, ictlnum1, imin1, imax1, init1, ifn1, ..., ictlnum64, imin64, imax64, init64, ifn64
k1, ..., k8	slider8f	ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, ..., ictlnum8, imin8, imax8, init8, ifn8, icutoff8
k1, ..., k16	slider16f	ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, ..., ictlnum16, imin16, imax16, init16, ifn16, icutoff16
k1, ..., k32	slider32f	ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, ..., ictlnum32, imin32, imax32, init32, ifn32, icutoff32
k1, ..., k64	slider64f	ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, ..., ictlnum64, imin64, imax64, init64, ifn64, icutoff64
i/k1, ..., i/k16	s16b14	ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, initvalue1, ifn1, ..., ictlno_msb16, ictlno_lsb16, imin16, imax16, initvalue16, ifn16
i/k1, ..., i/k32	s32b14	ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, initvalue1, ifn1, ..., ictlno_msb32, ictlno_lsb32, imin32, imax32, initvalue32, ifn32

Soporte MIDI: Entrada / Salida Genérica

kstatus, kchan, kdata1, kdata2	midiin	
	midiout	kstatus, kchan, kdata1, kdata2

Soporte MIDI: Activación / Desactivación de Nota

noteon	ichn, inum, ivel
noteoff	ichn, inum, ivel
noteondur	ichn, inum, ivel, idur
noteondur2	ichn, inum, ivel, idur
moscil	kchn, knum, kvel, kdur, kpause
midion	kchn, knum, kvel
midion2	kchn, knum, kvel, ktrig

Soporte MIDI: Salida de Mensajes MIDI

outic	ichn, inum, ivalue, imin, imax
outkc	kchn, knum, kvalue, kmin, kmax
outic14	ichn, imsb, ilsb, ivalue, imin, imax
outkc14	kchn, kmsb, klsb, kvalue, kmin, kmax
outipb	ichn, ivalue, imin, imax
outkpb	kchn, kvalue, kmin, kmax
outiat	ichn, ivalue, imin, imax
outkat	kchn, kvalue, kmin, kmax
outipc	ichn, iprog, imin, imax
outkpc	kchn, kprog, kmin, kmax
outipat	ichn, inotenum, ivalue, imin, imax
outkpat	kchn, knotenum, kvalue, kmin, kmax
nrpn	kchan, kparmnum, kparmvalue
mdelay	kstatus, kchan, kd1, kd2, kdelay

Soporte MIDI: Mensajes en Tiempo Real

mclock	ifreq
mrtmsg	imsgtype

Soporte MIDI: Extendedores de Evento MIDI

xtratim	iextradur
kflag	release

Generadores de Señal: Generadores Lineales y Exponenciales

k/ar	line	ia, idur1, ib
k/ar	expon	ia, idur1, ib
k/ar	linseg	ia, idur1, ib[, idur2, ic[...]]
k/ar	linsegr	ia, idur1, ib[, idur2, ic[...]], irel, iz
k/ar	expseg	ia, idur1, ib[, idur2, ic[...]]
k/ar	expsegr	ia, idur1, ib[, idur2, ic[...]], irel, iz
ar	expsega	ia, idur1, ib[, idur2, ic[...]]
k/ar	adsr	iatt, idec, islev, irel[, idel]
k/ar	madsr	iatt, idec, islev, irel[, idel]
k/ar	xadsr	iatt, idec, islev, irel[, idel]
k/ar	mxadsr	iatt, idec, islev, irel[, idel]

Generadores de Señal: Acceso a Tablas

i/k/ar	table	i/k/andx, ifn[, ixmode[, ixoff[, iwrap]]]
i/k/ar	tablei	i/k/andx, ifn[, ixmode[, ixoff[, iwrap]]]
i/k/ar	table3	i/k/andx, ifn[, ixmode[, ixoff[, iwrap]]]
kr	oscil1	idel, kamp, idur, ifn
kr	oscil1i	idel, kamp, idur, ifn
ar	osciln	kamp, ifrq, ifn, itimes

Generadores de Señal: Generadores de Fase

k/ar	phasor	k/xcps[, iphs]
k/ar	phasorbnk	k/xcps, kindx, icnt [, iphs]

Generadores de Señal: Osciladores Básicos

k/ar	oscil	k/xamp, k/xcps, ifn[, iphs]
k/ar	oscili	k/xamp, k/xcps, ifn[, iphs]
k/ar	oscil3	k/xamp, k/xcps, ifn[, iphs]
k/ar	poscil	kamp, kcps, ifn[, iphs]
k/ar	poscil3	kamp, kcps, ifn[, iphs]
k/ar	lfo	kamp, kcps[, itype]

Generadores de Señal: Osciladores de Espectro Dinámico

ar	buzz	xamp, xcps, knh, ifn[, iphs]
ar	gbuzz	xamp, xcps, knh, klh, kr, ifn[, iphs]
ar	vco	kamp, kfq, iwave, kpw, ifn, imaxd

Generadores de Señal: Síntesis / Resíntesis Aditiva

ar	adsyn	kamod, kfmod, ksmod, ifilcod
ar	adsynt	kamp, kcps, iwfn, ifreqfn, iampfn, icnt[, iphs]
ar	hsboscil	kamp, ktone, kbrite, ibasfreq, iwfn, ioctfn[, ioctcnt [, iphs]]

Generadores de Señal: Modelado Físico Waveguide

ar	pluck	kamp, kcps, icps, ifn, imeth[, iparm1, iparm2]
ar	wgpluck	icps, iamp, kpick, iplk, idamp, ifilt, axcite
ar	repluck	iplk, xam, icps, kpick, krefl, axcite
ar	wgpluck2	iplk, xam, icps, kpick, krefl
ar	wgbow	kamp, kfreq, kpres, krat, kvibf, kvamp, ifn[, iminfreq]
ar	wgflute	kamp, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, ifn[, iminfreq[, kjetrf[, kendrf]]]
ar	wgbrass	kamp, kfreq, iatt, kvibf, kvamp, ifn[, iminfreq]
ar	wgclar	kamp, kfreq, kstiff, iatt, idetk, kngain, kvibf, kvamp, ifn[, iminfreq]

Generadores de Señal: Modelos y Emulaciones

ar	moog	kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn
ar	shaker	kamp, kfreq, kbeans, kdamp, ktimes[, idecay]
ar	marimba	kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec
ar	vibes	kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec
ar	mandol	kamp, kfreq, kpluck, kdetune, kgain, ksize, ifn[, iminfreq]
ar	gogobel	kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn
ar	voice	kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn
ax, ay, az	lorenz	ks, kr, kb, kh, ix, iy, iz, iskip
ax, ay, az	planet	kmass1, kmass2, ksep, ix, iy, iz, ivx, ivy, ivz, idelta, ifriction

Generadores de Señal: Resíntesis STFT (Vocoding)

ar	pvoc	ktimpnt, kfmod, ifilcod, ifn, ibins[, ibinoffset, ibinincr, iextractmode, ifreqlim, igatefn]
kfr, kap	pvread	ktimpnt, ifile, ibin
	pvbufread	ktimpnt, ifile
ar	pvinterp	ktimpnt, kfmod, ifile, kfreqscale1, kfreqscale2, kampscale1, kampscale2, kfreqinterp, kampinterp
ar	pvcross	ktimpnt, kfmod, ifile, kamp1, kamp2[, ispecwp]
	tableseg	ifn1, idur1, ifn2[, idur2, ifn3[...]]
	tablexseg	ifn1, idur1, ifn2[, idur2, ifn3[...]]
ar	vpvoc	ktimpnt, kfmod, ifile[, ispecwp]
ar	pvadd	ktimpnt, kfmod, ifilcod, ifn, ibins[, ibinoffset, ibinincr, iextractmode, ifreqlim, igatefn]

Generadores de Señal: Resíntesis LPC

krmsr, krms0, kerr, kcps	lpread	ktimpnt, ifilcod[, inpoles[, ifrmrate]]
ar	lpreson	asig
ar	lpfreson	asig, kfrqratio
	lpslot	islot
	lpinterp	islot1, islot2, kmix

Generadores de Señal: Generadores Aleatorios (Ruido)

k/ar	rand	k/xamp[, iseed[, isize]]
k/ar	randh	k/xamp, k/xcps[, iseed[, isize]]
k/ar	randi	k/xamp, k/xcps[, iseed[, isize]]
i/k/ar	linrand	krange
i/k/ar	trirand	krange
i/k/ar	exprand	krange
i/k/ar	bexprnd	krange
i/k/ar	cauchy	kalpha
i/k/ar	pcauchy	kalpha
i/k/ar	poisson	klambda
i/k/ar	gauss	krange
i/k/ar	weibull	ksigma, ktau
i/k/ar	betarand	krange, kalpha, kbeta
i/k/ar	unirand	krange

Control de Tablas de Función: Consulta a Tablas

	nsamp(x)	(argumentos sólo a frecuencia de inicialización)
	ftlen(x)	(argumentos sólo a frecuencia de inicialización)
	ftlptim(x)	(argumentos sólo a frecuencia de inicialización)
	ftsr(x)	(argumentos sólo a frecuencia de inicialización)
i/kr	tableng	i/kfn

Control de Tablas de Función: Selección de Tablas

k/ar	tablekt	k/xndx, i/kfn[, ixmode[, ixoff[, iwrap]]]
k/ar	tableikt	k/xndx, kfn[, ixmode[, ixoff[, iwrap]]]

Control de Tablas de Función: Operaciones de Lectura / Escritura

	tablew	i/k/asig, i/k/andx, ifn[, ixmode[, ixoff[, iwgmde]]]
	tablewkt	k/asig, k/andx, kfn[, ixmode[, ixoff[, iwgmde]]]
	tableiw	isig, indx, ifn[, ixmode[, ixoff[, iwrap]]]
	tableigpw	ifn
	tablegpw	kfn
	tableimix	idft, idoff, ilen, is1ft, isloff, is1g, is2ft, is2off, is2g
	tablemix	kdft, kdoff, klen, ks1ft, ksloff, ks1g, ks2ft, ks2off, ks2g
	tableicopy	idft, isft
	tablecopy	kdft, ksft
ar	tablera	kfn, kstart, koff
kstart	tablewa	kfn, asig, koff

Modificadores de Señal: Filtros Estándar

kr	portk	ksig, khtim[, isig]
kr	port	ksig, ihtim[, isig]
kr	tonek	ksig, khp[, iskip]
ar	tone	asig, khp[, iskip]
kr	atonek	ksig, khp[, iskip]
ar	atone	asig, khp[, iskip]
kr	resonk	ksig, kcf, kbw[, iscl, iskip]
ar	reson	asig, kcf, kbw[, iscl, iskip]
kr	aresonk	ksig, kcf, kbw[, iscl, iskip]
ar	areson	asig, kcf, kbw[, iscl, iskip]
ar	tonex	asig, khp[, inumlayer, iskip]
ar	atonex	asig, khp[, inumlayer, iskip]
ar	resonx	asig, kcf, kbw[, inumlayer, iscl, iskip]
ar	resonr	asig, kcf, kbw[, iscl, iskip]
ar	resonz	asig, kcf, kbw[, iscl, iskip]
ar	resony	asig, kbf, kbw, inum, ksep[, iscl, iskip]
ar	lowres	asig, kcutoff, kresonance[, iskip]
ar	lowresx	asig, kcutoff, kresonance[, inumlayer, iskip]
ar	vlowres	asig, kfco, kres, iord, ksep
ar	lowpass2	asig, kcf, kq[, iskip]
ar	biquad	asig, kb0, kb1, kb2, ka0, ka1, ka2[, iskip]
ar	rezzy	asig, xfco, xres[, imode]
ar	moogvcf	asig, xfco, xres[, iscale]
alow, ahigh, aband	svfilt	asig, kcf, kq[, iscl]
ar1, ar2	hilbert	asig
ar	butterhp	asig, kfreq[, iskip]
ar	butterlp	asig, kfreq[, iskip]
ar	butterbp	asig, kfreq, kband[, iskip]
ar	butterbr	asig, kfreq, kband[, iskip]
k/ar	filter2	k/asig, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN
ar	zfilter2	asig, kdamp, kfreq, iM, iN, ib0, ib1, ..., ibM, ia1, ia2, ..., iaN

Modificadores de Señal: Filtros Especializados

ar	nlfilt	ain, ka, kb, kd, kL, kC
ar	pareq	asig, kc, iv, iq, imode
ar	dcblock	asig[, ig]

Modificadores de Señal: Modificadores de Envolverte

k/ar	linen	k/xamp, irise, idur, idec
k/ar	linenr	k/xamp, irise, idec, iatdec
k/ar	envlpx	k/xamp, irise, idur, idec, ifn, iatss, iatdec[, ixmod]
k/ar	envlpxr	k/xamp, irise, idur, idec, ifn, iatss, iatdec[, ixmod[, irind]]

Modificadores de Señal: Modificadores de Amplitud

kr	rms	asig[, ihp, iskip]
ar	gain	asig, krms[, ihp, iskip]
ar	balance	asig, acomp[, ihp, iskip]
ar	dam	ain, kthreshold, icomp1, icomp2, rtime, ftime

Modificadores de Señal: Limitadores de Señal

i/k/ar	wrap	i/k/asig, i/k/klow, i/k/khigh
i/k/ar	mirror	i/k/asig, i/k/klow, i/k/khigh
i/k/ar	limit	i/k/asig, i/k/klow, i/k/khigh

Modificadores de Señal: Líneas de Retardo

ar	delayr	idlt[, iskip]
	delayw	asig
ar	delay	asig, idlt[, iskip]
ar	delay1	asig[, iskip]
ar	deltap	kdlt
ar	deltapi	xdlt
ar	deltapn	xnumsamps
ar	deltap3	xdlt
ar	multitap	asig, itime1, igain1, itime2, igain2...
ar	vdelay	asig, adel, imaxdel[, iskip]
ar	vdelay3	asig, adel, imaxdel[, iskip]

Modificadores de Señal: Reverberación

ar	reverb	asig, krvt[, iskip]
ar	reverb2	asig, ktime, khdif[, iskip]
ar	nreverb	asig, ktime, khdif[, iskip]
ar	comb	asig, krvt, ilpt[, iskip]
ar	alpass	asig, krvt, ilpt[, iskip]
ar	nestedap	asig, imode, imaxdel, idel1, igain1[, idel2, igain2[, idel3, igain3]]

Modificadores de Señal: Guías de Onda

ar	wguide1	asig, kfreq, kcutoff, kfeedback
ar	wguide2	asig, kfreq1, kfreq2, kcutoff1, kcutoff2, kfeedback1, kfeedback2
ar	streson	asig, kfr, ifdbgain
ar	nlalp	asig, klcf, knlcf [, iskip[, iupdm]]

Modificadores de Señal: Efectos Especiales

ar	harmon	asig, kestfrq, kmaxvar, kgenfreq1, kgenfreq2, imode, iminfrq, iprd
ar	flanger	asig, adel, kfeedback[, imaxd]
ar	distort1	asig[, kpregain[, kpostgain[, kshape1[, kshape2]]]]
ar	phaser1	asig, kfreq, iord, kfeedback[, iskip]
ar	phaser2	asig, kfreq, iord, imode, ksep, kfeedback

Modificadores de Señal: Convolución y Morphing

ar1[, ar2[, ar3[, ar4]]]	convolve	ain, ifilcod, ichannel
ar	cross2	ain1, ain2, isize, ioverlap, iwin, kbias

Modificadores de Señal: Panoramización y Espacialización

a1, a2, a3, a4	pan	asig, kx, ky, ifn[, imode[, ioffset]]
a1, a2	locsig	asig, kdegree, kdistance, kreverbsend
a1, a2, a3, a4	locsig	asig, kdegree, kdistance, kreverbsend
a1, a2	locsend	
a1, a2, a3, a4	locsend	
a1, a2, a3, a4	space	asig, ifn, ktime, kreverbsend[, kx, ky]
a1, a2, a3, a4	spsend	
k1	spdist	ifn, ktime[, kx, ky]
aleft, aright	hrtfer	asig, kaz, kelev, "HRTFcompact"

Modificadores de Señal: Operadores a Nivel de Muestra

kr	downsamp	asig[, iwlen]
ar	upsamp	ksig
ar	interp	ksig[, iskip]
k/ar	integ	k/asig[, iskip]
k/ar	diff	k/asig[, iskip]
k/ar	samphold	x/asig, k/xgate[, ival, ivstor]
i/k/ar	ntrpol	i/k/asig1, i/k/asig2, i/k/kpoint[, imin, imax]
ar	fold	asig, kincr

El Sistema de Cableado Zak

	zakinit	isizea, isizek
	ziw	isig, indx
	zkw	ksig, kndx
	zaw	asig, kndx
	ziwm	isig, indx[, imix]
	zkwm	ksig, kndx[, kmix]
	zawm	asig, kndx[, kmix]
ir	zir	indx
kr	zkr	kndx
ar	zar	kndx
ar	zarg	kndx, kgain
kr	zkmod	ksig, kzmod
ar	zamod	asig, kzmod
	zkcl	kfirst, klast
	zacl	kfirst, klast

Operaciones usando Tipos de Datos Espectrales

wsig	specaddm	wsig1, wsig2[, imul2]
wsig	specdiff	wsigin
wsig	specscal	wsigin, ifscale, ifthresh
wsig	spechist	wsigin
wsig	specfilt	wsigin, ifhtim
koct, kamp	specptrk	wsig, kvar, ilo, ihi, istrtr, idbthresh, inptls, irolloff[, ioddd, iconfs, interp, ifprd, iwtflg]
ksum	specsum	wsig[, interp]
	specdisp	wsig, iprd[, iwtflg]
wsig	spectrum	xsig, iprd, iocts, ifrqa, iq[, ihann, idbout, idsprd, idsinrs]

Entrada y Salida de Señal: Entrada de Señal

```
a1          in
a1, a2     ins
a1, a2,    inq
a3, a4
a1, a2,    inh
a3, a4,
a5, a6
a1, a2,    ino
a3, a4,
a5, a6,
a7, a8
a1          soundin  ifilcod[, iskptim[, iformat]]
a1, a2     soundin  ifilcod[, iskptim[, iformat]]
a1, a2,    soundin  ifilcod[, iskptim[, iformat]]
a3, a4
a1[,a2     diskin   ifilcod, kpitch[, iskiptim[, iwraparound[, iformat]]]
[,a3,a4]]
```

Entrada y Salida de Señal: Salida de Señal

```
out         asig
outs1       asig
outs2       asig
outs        asig1, asig2
outq1       asig
outq2       asig
outq3       asig
outq4       asig
outq        asig1, asig2, asig3, asig4
outh        asig1, asig2, asig3, asig4, asig5, asig6
outo        asig1, asig2, asig3, asig4, asig5, asig6, asig7, asig8
soundout    asig1, ifilcod[, iformat]
soundouts   asig1, asig2, ifilcod[, iformat] (**No implementado**)
```

Entrada y Salida de Señal: Entrada / Salida de Ficheros

	dumpk	ksig, ifilename, iformat, iprd
	dumpk2	ksig1, ksig2, ifilename, iformat, iprd
	dumpk3	ksig1, ksig2, ksig3, ifilename, iformat, iprd
	dumpk4	ksig1, ksig2, ksig3, ksig4, ifilename, iformat, iprd
ksig	readk	ifilename, iformat[, ipol]
k1, k2	readk2	ifilename, iformat[, ipol]
k1,k2,k3	readk3	ifilename, iformat[, ipol]
k1,k2, k3,k4	readk4	ifilename, iformat[, ipol]
	fout	"ifilename", iformat, aout1[, aout2, aout3,...,aoutN]
	foutk	"ifilename", iformat, aout1[, aout2, aout3,...,aoutN]
	fouti	ihandle, iformat, iflag, iout1[, iout2, iout3,...,ioutN]
	foutir	ihandle, iformat, iflag, iout1[, iout2, out3,...,ioutN]
ihandle	fiopen	"ifilename", imode
	fin	"ifilename", iskipframes, iformat, ain1[, ain2, ain3,...,ainN]
	fink	"ifilename", iskipframes, iformat, kin1[, kin2, kin3,...,kinN]
	fini	"ifilename", iskipframes, iformat, in1[, in2, in3,...,inN]
	vincr	asig, aincr
	clear	avar1[,avar2, avar3,...,avarN]

Entrada y Salida de Señal: Consultas a Ficheros de Sonido

ir	filelen	"ifilcod"
ir	filesr	"ifilcod"
ir	filenchnls	"ifilcod"
ir	filepeak	"ifilcod"[, ichnl]

Entrada y Salida de Señal: Impresión y Presentación en Pantalla

	print	iarg[, iarg, ...]
	display	xsig, iprd[, inprds[, iwtflg]]
	dispfft	xsig, iprd, iwsiz[, iwtyp[, idbouti[, iwtflg]]]
	printk	kval, ispace[, itime]
	printks	"txtstring", itime, kval1, kval2, kval3, kval4
	printk2	kvar[, numspaces]

Partitura Numérica Estándar: Sentencias

f	"número tabla" "instante activación" "tamaño" "rutina GEN" arg1[arg2...arg...]
f0	"instante activación" (Falsa tabla de función para rellenar secciones de la partitura con silencio y optimizar el trabajo con obras largas).
b	"tiempo base del reloj" (Efectivo antes de la ordenación de la partitura. Es premodificado).
t	0 "tempo inicial" "instante en pulsos" "tempo2"["instante en pulsos" "tempo3" "instante en..."]
a	0 "instante del principio de avance en pulsos" "duración del avance en pulsos"
i	"número instrumento" "comienzo" "duración" [p4 p5 p...]
s	(marca el final de una sección y hace que la siguiente empiece a contar a partir del instante 0)
m	"nombre de la marca" (pone nombre a una sección de la partitura)
n	"nombre de la marca" (la sección indicada por "nombre de la marca" será releída dentro de la partitura en el punto donde aparezca n)
r	"contador de repetición" "nombre de macro" (comienza la repetición de una sección)
e	(marca el final de la partitura. Es opcional)

Partitura Numérica Estándar: Sustitución de Campos-P

.	(toma el valor del campo p de la sentencia "i" precedente con el mismo número de instrumento).
+	(determina el instante de comienzo actual según la suma de los campos p2 y p3 de la sentencia "i" previa. Sólo es legal en p2).
^+x	(determina el instante de comienzo de un instrumento sumando x al campo p2 del evento precedente. Sólo es legal en p2).
^-x	(determina el instante de comienzo de un instrumento restando x al campo p2 del evento precedente. Sólo es legal en p2).
np_x	(toma el valor del campo p(x) de la siguiente sentencia de nota. Ilegal en p1 p2 p3).
pp_x	(toma el valor del campo p(x) de la sentencia de nota anterior. Ilegal en p1 p2 p3).
<	(el campo p toma un valor calculado de la interpolación lineal entre los valores fijos previo y siguiente del mismo campo p. Ilegal en p1 p2 p3).
>	(el campo p toma un valor calculado de la interpolación lineal entre los valores fijos previo y siguiente del mismo campo p. Ilegal en p1 p2 p3).
)	(el campo p toma un valor calculado de la interpolación exponencial entre los valores fijos previo y siguiente del mismo campo p. Ilegal en p1 p2 p3).
((el campo p toma un valor calculado de la interpolación exponencial entre los valores fijos previo y siguiente del mismo campo p. Ilegal en p1 p2 p3).
~	(el campo p toma un valor aleatorio dentro del rango establecido por los valores fijos previo y siguiente del mismo campo p. Ilegal en p1 p2 p3).

Partitura Numérica Estándar: Expresiones

- [x+y] (añade el valor x al valor y en un campo p . Observa que las expresiones deben ir entre corchetes [])
- [x-y] (resta del valor x el valor y en un campo p . Observa que las expresiones deben ir entre corchetes [])
- [x*y] (multiplica el valor x por el valor y en un campo p . Observa que las expresiones deben ir entre corchetes [])
- [x/y] (divide el valor x por el valor y en un campo p . Observa que las expresiones deben ir entre corchetes [])
- [x%y] (resto de la división del valor x por el valor y en un campo p . Observa que las expresiones deben ir entre corchetes [])
- [x^y] (eleva el valor x a la potencia indicada por el valor y en un campo p . Observa que las expresiones deben ir entre corchetes [])
- [@x] (calcula la potencia de 2 mayor o igual más cercana a x . Observa que las expresiones deben ir entre corchetes [])
- [@@x] (calcula la potencia de 2 más 1, mayor o igual más cercana a x . Observa que las expresiones deben ir entre corchetes [])

Macros de la Partitura

```
#define NOMBRE # texto de reemplazo #  
#define NOMBRE(a'b'c) # texto de reemplazo #  
$NAME.  
#undef NOMBRE  
  
#include "nombrefichero"
```

Rutinas GEN: Generadores Seno / Coseno

```
f # time size 9 pna      stra      phsa      pnb      strb      phsb      ...
f # time size 10 str1     str2     str3     str4     ...
f # time size 19 pna      stra      phsa      dcoa     pnb      strb      phsb      dcob
f # time size 11 nh       lh       r
```

Rutinas GEN: Generadores de Segmentos Lineales / Exponenciales

```
f # time size 5 a        n1       b        n2       c        ...
f # time size 6 a        n1       b        n2       c        n3       d        ...
f # time size 7 a        n1       b        n2       c        ...
f # time size 8 a        n1       b        n2       c        n3       d        ...
f # time size 25 x1      y1      x2      y2      x3      ...
f # time size 27 x1      y1      x2      y2      x3      ...
```

Rutinas GEN: Acceso a Ficheros

```
f # time size 1  filcod  skiptime  format  channel
f # time size 23 "filename.txt"
f # time  0   28  filcod
```

Rutinas GEN: Acceso a Valores Numéricos

```
f # time size 2  v1      v2      v3      ...
f # time size 17 x1      a       x2      b       x3      c       ...
```

Rutinas GEN: Funciones de Ventana

```
f # time size 20 window  max     op
```

Rutinas GEN: Funciones Aleatorias

```
f # time size 21 type     lvl     arg1    arg2
```

Rutinas GEN: Modelado de Ondas

```
f # time size 3  xvall  xval2  c0     c1     c2     ...     cn
f # time size 13 xint   xamp   h0     h1     h2     ...     hn
f # time size 14 xint   xamp   h0     h1     h2     ...     hn
f # time size 15 xint   xamp   h0     phs0   h1     phs1   h2     phs2
```

Rutinas GEN: Escalado de Amplitudes

```
f # time size 4  source#  sourcemode
f # time size 12 xint
```

Indicadores de la Línea de Comando

-C	usa Cscore para el procesado del fichero partitura
-I	ejecución de la orquesta sólo en frecuencia de inicialización
-n	no escribe sonido en el disco
-i <i>fnam</i>	entrada de sonido del fichero <i>fnam</i>
-o <i>fnam</i>	salida de sonido al fichero <i>fnam</i>
-b <i>N</i>	<i>sample frames</i> (o <i>-kprds</i>) por buffer de E/S de sonido en software
-B <i>N</i>	muestras por buffer de E/S de sonido en hardware
-A	crea un fichero de sonido de salida con formato AIFF
-W	crea un fichero de sonido de salida con formato WAV
-J	crea un fichero de sonido de salida con formato IRCAM
-h	crea un fichero de sonido de salida sin cabecera
-c	muestras de sonido de 8 bits (<i>signed_char</i>)
-a	muestras de sonido <i>alaw</i>
-8	muestras de sonido de 8 bits (<i>unsigned_char</i>)
-u	muestras de sonido <i>ulaw</i>
-s	muestras de sonido <i>short_int</i>
-l	muestras de sonido <i>long_int</i>
-f	muestras de sonido <i>float</i>
-r <i>N</i>	sustituye la frecuencia de muestreo (<i>sr</i>) de la orquesta
-k <i>N</i>	sustituye la frecuencia de control (<i>kr</i>) de la orquesta
-v	muestra mensajes durante la traducción de la orquesta
-m <i>N</i>	Nivel de mensaje. <i>N</i> = suma de: 1 = amplitudes, 2 = muestras fuera de rango, 4 = advertencias
-d	suprime toda salida en pantalla
-g	Suprime los gráficos, usa representaciones ASCII
-G	Crea salidas Postscript de cualquier salida en pantalla
-S	la partitura está en formato Scot
-x <i>fnam</i>	extrae de <i>score.srt</i> usando el fichero de extracción <i>fnam</i>
-t <i>N</i>	usa los pulsos no interpretados de la partitura, inicialmente en tiempo <i>N</i>
-L <i>dnam</i>	lee eventos de partitura en tiempo real del dispositivo <i>dnam</i>
-M <i>dnam</i>	lee eventos MIDI en tiempo real del dispositivo <i>dnam</i>
-F <i>fnam</i>	lee los eventos del fichero MIDI <i>fnam</i>
-P <i>N</i>	umbral del pedal de sostenimiento MIDI (<i>N</i> = 0 - 128)
-R	reescribe continuamente la cabecera mientras escribe el fichero de sonido (WAV/AIFF)

Indicadores de la Línea de Comando (Continuación)

-H/H1	imprime un caracter heartbeat cada fichero de sonido escrito
-H2	genera un "." cada vez que se llena un buffer
-H3	informa del tamaño de la salida en segundos
-H4	toca un sonido cada vez que se escribe un buffer en la salida
-N	notifica (por un sonido) cuando se acaba el fichero MIDI o la partitura
-T	termina la ejecución cuando el fichero MIDI ha acabado
-D	difiere la carga de ficheros de sonido en GEN01 hasta el instante de ejecución
-z	Lista todos los opcodes de la versión actual
-z1	Lista todos los opcodes y argumentos de la versión actual
-- <i>lognam</i>	Manda toda la salida de texto a <i>lognam</i>
-j <i>fnam</i>	Consigue todos los mensajes de la consola de la base de datos <i>fnam</i>

Indicadores de la Línea de Comando: Llamadas a Utilidades

-U <i>sndinfo</i>	ejecuta la utilidad sndinfo
-U <i>hetro</i>	ejecuta la utilidad hetro
-U <i>lpanal</i>	ejecuta la utilidad lpanal
-U <i>pvanal</i>	ejecuta la utilidad pvanal
-U <i>cvanal</i>	ejecuta la utilidad cvanal
-U <i>pvlook</i>	ejecuta la utilidad pvlook
-C	Usar el procesado de Cscore

Indicadores Específicos de PC Windows

-j <i>num</i>	número de filas de texto en la consola (por defecto 25)
-J <i>num</i>	número de columnas de texto en la consola (por defecto 80)
-K <i>num</i>	habilita la entrada MIDI. <i>num</i> (opcional) = número del puerto de entrada MIDI
-q <i>num</i>	número del puerto de salida de onda (usar sólo si hay más de uno)
-p <i>num</i>	número de buffers de la salida de onda (por defecto 4; max. 40)
-O	suprime todas las salidas en pantalla para optimizar la ejecución en tiempo real
-e	permite cualquier frecuencia de muestreo (usar sólo con las tarjetas de sonido que soporten esta opción)
-y	no espera que se pulse una tecla para salir
-E	permite la salida gráfica para el WCSHELL de Riccardo Bianchini
-Q <i>num</i>	habilita el puerto de salida MIDI. <i>num</i> (opcional) = número del puerto de salida MIDI
-Y	suprime la salida de forma de onda para optimizar la salida MIDI en tiempo real.
-*	cede el control al sistema hasta que el buffer de audio este lleno

Indicadores Específicos de Macintosh

-q <i>sampdir</i>	establece el directorio en el que encontrar las muestras
-------------------	--

-Q *anadir* establece el directorio en el que encontrar los análisis
-X *snmdir* establece el directorio en el que salvar los ficheros
-V *num* establece el tamaño del buffer de pantalla
-E *num* establece el número de gráficos grabados
-p tocar al finalizar
-e *num* establece el factor de reescala
-w habilita la grabación de datos MIDI
-y *num* establece la frecuencia de mensajes de estado
-Y *num* establece la frecuencia de mensajes de resumen

Utilidades: Generación de Ficheros de Análisis

hetro	-sr n	ficentrada	ficsalida	frecuencia de muestreo del análisis heterodino
	-c n	ficentrada	ficsalida	número de canal del análisis
	-b n	ficentrada	ficsalida	instante de comienzo del segmento del análisis
	-d n	ficentrada	ficsalida	duración del segmento del análisis
	-f n	ficentrada	ficsalida	frecuencia inicial del análisis
	-h n	ficentrada	ficsalida	número de parciales del análisis
	-M n	ficentrada	ficsalida	amplitud máxima del análisis
	-m n	ficentrada	ficsalida	amplitud mínima del análisis
	-n n	ficentrada	ficsalida	número de puntos de inflexión del
	-l n	ficentrada	ficsalida	Se usa un filtro pasa bajos de tercer orden con f_c de n
lpanal	-a	ficentrada	ficsalida	El análisis LPC escribe polos de filtro en vez de coeficientes
	-s n	ficentrada	ficsalida	frecuencia de muestreo del análisis
	-c n	ficentrada	ficsalida	número de canal del análisis
	-b n	ficentrada	ficsalida	instante de comienzo del segmento del análisis
	-d n	ficentrada	ficsalida	duración del segmento del análisis
	-p n	ficentrada	ficsalida	número de polos del análisis
	-h n	ficentrada	ficsalida	tamaño del salto del análisis en muestras
	-C s	ficentrada	ficsalida	cadena de caracteres para los comentarios del análisis
	-P n	ficentrada	ficsalida	frecuencia más baja del análisis
	-Q n	ficentrada	ficsalida	frecuencia más alta del análisis
	-v n	ficentrada	ficsalida	nivel de detalle de los mensajes en pantalla del análisis
pvanal	-s n	ficentrada	ficsalida	frecuencia de muestreo del análisis STFT
	-c n	ficentrada	ficsalida	número de canal del análisis
	-b n	ficentrada	ficsalida	instante de comienzo del segmento del análisis
	-d n	ficentrada	ficsalida	duración del segmento del análisis
	-n n	ficentrada	ficsalida	tamaño del frame del análisis
	-w n	ficentrada	ficsalida	factor de solapamiento de ventanas del análisis
	-h n	ficentrada	ficsalida	tamaño del salto del análisis en muestras
cvanal	-s n	ficentrada	ficsalida	frecuencia de muestreo del análisis FFT
	-c n	ficentrada	ficsalida	número de canal del análisis
	-b n	ficentrada	ficsalida	instante de comienzo del segmento del análisis
	-d n	ficentrada	ficsalida	duración del segmento del análisis

pvlook	-bb	n	ficentrada	<i>número de pista inicial</i> del análisis de examen de fichero STFT
	-eb	n	ficentrada	<i>número de pista final</i> del análisis de examen de fichero STFT
	-bf	n	ficentrada	<i>número de frame inicial</i> del análisis de examen de fichero STFT
	-ef	n	ficentrada	<i>número de frame final</i> del análisis de examen de fichero STFT
	-i		ficentrada	el análisis de examen de fichero STFT produce valores enteros de salida