

Kurze Einführung in die Klangprogrammierung mit Csound

Peter Remmers, pe.re@online.de

22. Mai 2002

1 Allgemeines

Die Produktion von Klang mit Csound läuft in zwei Schritten ab. Zunächst werden vom Benutzer zwei Textdateien erstellt, namentlich die Orchestra-Datei (entspricht dem »Orchester«, also den Instrumenten bzw. den Klangerzeugern) und die Score-Datei (entspricht der »Partitur«, also den Spielanweisungen für das Orchester). Diese beiden Dateien¹ können beliebig benannt werden; zur besseren Übersicht erhalten zueinander gehörende Orchestra und Score-Dateien normalerweise den gleichen Namen und unterscheiden sich nur durch unterschiedliche Datei-Endungen – .orc für die Orchestra-Datei und .sco für die Score-Datei.

Im zweiten Schritt wird Csound mit diesen beiden Dateien gestartet. Dies geschieht durch den Aufruf des Programms auf Kommandozeilenebene, wo bestimmte Optionen für den Ablauf des Berechnungsprozesses festgelegt werden können.

Zur Benutzung von Csound hat man also drei verschiedene Werkzeuge zur Hand: die Orchestra-Datei, die Score-Datei sowie die Kommandozeile; die Bedienung dieser drei Elemente wird in den Kapiteln 2, 3 und 4 erläutert. Zunächst jedoch folgen Beschreibungen einiger grundlegender Funktionsprinzipien.

¹Die beiden Dateien können auch im sogenannten *Unified File Format* zusammengefasst werden; Orchestra- und Score-Datei sowie Kommandozeilenoptionen werden durch sogenannte *Tags*, also spezielle Dokument-Struktur-Befehle, voneinander unterschieden und in einer einzigen Datei, der .csd-Datei, gespeichert.

1.1 Bitrate und Samplerate

Im digitalen System werden Signale durch zwei Parameter repräsentiert, den Amplitudenwert und den Zeitwert. Die Werte dieser Parameter sind diskret, wodurch jeweils nur eine begrenzte Menge an Werten für beide Parameter zur Verfügung steht. Diese Menge diskreter Werte wird für die Amplitudenwerte durch die Bitrate (in Bit) und für die Zeitwerte durch die Abtastrate (auch: *Samplerate*) in Hertz ausgedrückt. Zum Beispiel kann ein 16-Bit System nur genau $16^2 = 65536$ unterschiedliche Amplitudenwerte darstellen; eine Samplerate von z.B. 44100 Hertz stellt genau 44100 Amplitudenwerte pro Sekunde dar. Der wohlbedachte Umgang mit Amplitudenwerten und Samplerate spielen in Csound eine grosse Rolle, wie im folgenden erläutert wird.

Amplitudenwerte

Die Amplitude eines Signals wird in Csound durch den Absolutwert in Zweierkomplement-Darstellung² repräsentiert. Soll ein Ton mit maximaler Lautstärke von Csound berechnet werden, muss im 16 Bit System³ für die Amplitude der Wert 32767 eingegeben werden (zur Vereinfachung können verschiedene Konverter verwendet werden, z.B. von dB in Absolutwert). Dabei gilt es zu beachten, dass sich die Amplitudenwerte von mehreren gleichzeitig gespielten Klängen addieren. Dies ist schwierig zu kontrollieren, da z.B. drei Klänge mit Amplitudenwerten von jeweils 10000 *nicht unbedingt* zusammen einen Wert von 30000 ergeben. Der resultierende Amplitudenwert hängt vielmehr von Wellenform, Frequenz und Phasenlage der einzelnen Signale ab. Wird der maximale im 16-Bit System darstellbare Wert überschritten, führt dies zu digitaler Übersteuerung (Clipping), die starke Verzerrungen nach sich zieht. Der Benutzer muss also entweder durch ausprobieren die richtigen Amplitudenwerte herausfinden, oder bestimmte Hilfsmittel verwenden, wie z.B. eine Limiter-Funktionseinheit.

Samplerate

Csound erlaubt die freie Einstellung einer globalen Samplerate für den Syntheseprozess (abhängig von der Audio-Hardware). Nach dem von Nyquist und Shannon formulierten *Sampling-Theorem* muss die Samplerate doppelt so hoch wie die höchste exakt darzustellende Frequenz gewählt werden. Mit anderen Worten: die höchste Frequenz, die im Spektrum eines Signals enthalten sein soll, kann nicht höher als die halbe Samplerate sein. Frequenzen, die oberhalb der halben Abtastfrequenz liegen, verursachen sogenannte *Aliasing-Geräusche*. Csound schützt nicht implizit vor Aliasing-Geräuschen; der Benutzer muss selbst beim Entwurf von Instrumenten darauf achten, dass keine Frequenzen oberhalb der halben Samplerate entstehen, sofern Aliasing vermieden werden soll.

²In der Zweierkomplement-Darstellung werden die Werte nicht von Null bis z.B. 65536 (16 Bit System) dargestellt, sondern von -32768 bis $+32767$.

³In Systemen mit anderer Bitrate müssen entsprechend andere Absolutwerte gewählt werden

1.2 Signaltypen und Variablen

Signale werden in Csound in drei verschiedene Typen unterteilt:

- Audiosignale (audio-rate bzw. a-rate-signals): Audiosignale sind im bekannten Sinne Klänge. Sie werden mit der vom Benutzer festgelegten Samplerate erzeugt und bearbeitet, z.B. mit 44100 Hertz.
- Kontrollsignale (control-rate bzw. k-rate-signals): Kontrollsignale sind Steuersignale, die selbst nicht unbedingt als Klang hörbar werden, aber die Audiosignale auf verschiedenste Weise beeinflussen. Verschiedene Kontrollsignale sind z.B. Hüllkurven, Vibrato, Tremolo, usw. In den meisten Fällen variieren solcherlei Bearbeitungen nur relativ langsam, im Vergleich zu Audiosignalen. Dieses Merkmal erlaubt es nun, mittels eines einfachen Tricks Rechenzeit zu sparen. Um Steuersignale exakt darzustellen, wird eine wesentlich geringere Aktualisierungs-Rate gebraucht, als für die Darstellung von Audiosignalen. Kontrollsignale werden in Csound daher auf einer anderen Zeitebene erzeugt und bearbeitet. Zusätzlich zur Samplerate wird die *Kontrollrate* eingeführt, die üblicherweise einen kleinere Wert als die Samplerate hat.
- Initialisierungssignale (init-rate bzw. i-rate-signals): Diese Signale werden bei der Initialisierung eines Instruments – etwa beim Spielen einer Note – erzeugt und ändern ihren Wert bis zum Ende der Note nicht mehr. I-rate-Signale werden also nur einmal pro Note berechnet.

In Csound wird jedes Signal von Variablen, also temporär angelegten Bereichen im Arbeitsspeicher des Computers, repräsentiert. Variablen sind beliebige Zeichenfolgen, die einen bestimmten, variierbaren Wert repräsentieren. Eine Variable in Csound kann z.B. **asig** heissen, oder auch **ipferd**, **kspinat** usw⁴. Entscheidend ist ausschliesslich der erste Buchstabe der Variablen. Anhand dieses Buchstaben identifiziert Csound die Variable als a-rate, k-rate oder i-rate-signal.

Darüber hinaus wird zwischen *lokalen* und *globalen* Variablen unterschieden. Lokale Variablen gelten ausschliesslich für ein einzelnes Instrument und können von einem anderen Instrument in keiner Weise beeinflusst werden. So können in zwei verschiedenen Instrumenten unterschiedliche Variablen mit dem selben Namen verwendet werden.

Globale Variablen hingegen stehen allen Instrumenten zur Verfügung. Sie werden benutzt, damit verschiedene Instrumente miteinander kommunizieren können und somit Steuer- oder Audiosignale von allen Instrumenten verarbeitet werden können (z.B. zur Effektbeschickung oder zum Zusammenmischen mehrerer Instrumente). Globale Variablen werden von lokalen Variablen durch ein vorangestelltes »g« unterschieden; z.B. bezeichnet die Variable **gasig** ein globales a-rate-signal.

⁴Bestimmte reservierte Namen wie *kr*, *sr* usw. dürfen nicht zur Bezeichnung von neu definierten Variablen verwendet werden.

1.3 Funktionstabellen

Funktionstabellen (auch: *lookup tables*) sind elementare Bestandteile der Klangprogrammierung in Csound. Sie bestehen aus einer vom Benutzer festgelegten Anzahl Speicher-Adressen, in denen jeweils ein bestimmter Wert gespeichert ist. Vom Aufbau her gleichen Funktionstabellen Audio-Samples, mit dem Unterschied, dass die Werte der Tabellen normalerweise durch mathematische Funktionen erzeugt werden⁵. Zum Beispiel werden Wellenformen, Hüllkurven, Zufallsverteilungen usw. in Funktionstabellen definiert.

Durch die Verwendung von Funktionstabellen wird Rechenzeit gespart, da die verschiedenen Funktionen nur einmal berechnet und anschliessend nur noch aus dem Arbeitsspeicher gelesen werden. Eine Sinusschwingung wird also einmal berechnet und die berechneten Werte im Arbeitsspeicher gespeichert. Wenn ein Oszillator nun z.B. einen Sinuston mit 2 kHz ausgeben soll, liest er 2000 mal pro Sekunde sämtliche gespeicherten Werte aus. Welche Funktion in der Tabelle dargestellt werden soll, wird mit den sogenannten GEN-Routinen festgelegt; doch dazu später mehr.

⁵In bestimmten Fällen können auch Werte aus einer Datei, z.B. einer Audio-Datei oder einer Text-Datei, in eine Funktionstabelle importiert werden.

2 Die Programmierung der Orchestra-Datei

In den folgenden Erläuterungen werden Variablen mit $\langle \dots \rangle$ gekennzeichnet.

2.1 Header und Instrumentensektion

Jede Orchestra-Datei besteht aus zwei Teilen, dem *Header* (der »Kopf« der Datei) und der *Instrumentensektion*.

Im Header werden folgende Parameter festgelegt:

- die *Samplingrate* sr in Hz
- die *Kontrollrate* kr in Hz
- die Anzahl der Audiosamples pro Kontrollperiode $ksmps$ (entspricht $\frac{sr}{kr}$)
- die Anzahl der Kanäle $nchnls$

sr , kr , $ksmps$ und $nchnls$ sind reservierte, globale Variablen. Ihre Werte können während der Ausführung der Orchestra-Datei nicht mehr verändert werden.

Ein Beispiel für einen Orchestra Header:

```
sr      = 44100
kr      = 4410
ksmps   = 10
nchnls  = 1
```

In diesem Header ist die Samplingrate auf 44100 Hz festgelegt; die Kontrollrate beträgt 4410 Hz, also ein Zehntel von 44100; somit muss $ksmps$ auf 10 festgelegt werden, da $44100/4410 = 10$. Die Anzahl der Kanäle ist auf einen festgelegt. Speichert man das Ergebnis der Klangberechnung in einer Datei, wird dies also eine Mono-Datei mit einer Abtastfrequenz von 44,1 kHz sein.

In der Instrumentensektion werden die einzelnen Instrumente definiert. Ein Instrument besteht aus miteinander verbundenen Modulen bzw. Funktionseinheiten (*Opcodes*), die jeweils ein Signal erzeugen oder bearbeiten. Jedes Instrument bekommt eine eigene Nummer und wird durch die Anweisungen `instr <Nummer>` und `endin` von den anderen Instrumenten abgegrenzt (der *Instrument Block*).

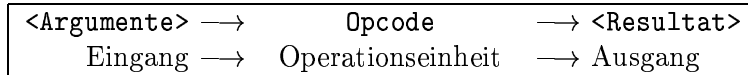


Tabelle 2.1: Die »Anschlüsse« eines Opcodes.

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

Es können beliebig viele Instrumente in einer Orchestra-Datei enthalten sein.

Die Bezeichnung »Instrument« soll nicht bedeuten, dass diese Einheit in Csound ausschließlich Klänge erzeugt. Jede Art von Signalbearbeitung, also auch Effekte, Steuervorgänge und ähnliches kann in Instrumenten zusammengefasst werden, wenn sie unabhängig von anderen Instrumenten gesteuert werden soll.

2.2 Syntax

Eine Anweisung (*statement*) hat in der Orchestra-Datei folgendes Format:

```
Label: <Resultat> Opcode <Argument1>, <Argument2>, ... ;Kommentar
```

Label und Kommentar sind optional. Ein Label kann als Sprungmarke verwendet werden, um die Zeile als Ziel einer goto-Anweisung zu markieren. Ein Kommentar kann an jeder Stelle mittels vorangestelltem Semikolon eingefügt werden und gilt bis zum Ende der Zeile; er dient zur Dokumentation des Codes. Label und Kommentare haben keine Auswirkung auf die Anweisung.

Die eigentliche Anweisung wird durch Resultat, Opcode und Argumente gebildet. Der Opcode bezeichnet die Funktionseinheit, also die durchzuführende Operation. Die Argumente bilden in Form von Konstanten, Variablen oder Ausdrücken die Eingabewerte für die Operation (sozusagen der Eingang der Operationseinheit). Argumente können auch aus mathematischen Konstrukten mit arithmetischen und logischen Operatoren wie +, −, *, /, =, AND, OR, % usw. bestehen. Der Ergebniswert der Operation ist das Resultat (also praktisch der Ausgang der Operationseinheit); er wird in eine Variablen geschrieben. Ein Beispiel¹:

```
instr 1 ;Beispiel 1
asig oscil 20000, 700, 1
out asig
endin
```

Diese vier Zeilen Code sind in etwa das einfachste Instrument, das in Csound programmiert werden kann. Zunächst wird Instrument 1 eingeleitet (`instr 1`). Die Variable `asig` bildet das Resultat, also den Ausgang des Opcodes. `oscil` ist ein einfacher Oszillator,

¹Der Header wird von nun an au weggelassen, da er in allen Beispielen immer der gleiche ist; siehe Beispiel-Header in Abschnitt 2.1.

2 Die Programmierung der Orchestra-Datei

der drei Argumente erwartet: die Amplitude als Absolutwert, im Beispiel 20000, die Frequenz, hier 700 Hz und schliesslich die Wellenform als Funktionstabellennummer. Nun muss noch eine Anweisung zur Ausgabe der Berechnung folgen; dazu wird der Opcode `out` mit der Variablen `asig`, die ja die berechneten Werte von `oscil` beinhaltet, verwendet. Schliesslich wird die Definition von Instrument 1 mit `endin` beendet.

Die Orchestra-Datei alleine wird keinen Ton von sich geben; das Instrument muss aus einer Score-Datei aufgerufen werden. Hier muss ebenfalls noch die Funktionstabelle generiert werden. Die passende Score-Datei, die den in Instrument 1 definierten Sinusoszillator für genau eine Sekunde spielt, sieht dann so aus:

```
f1 0 8192 10 1 ;Funktionstabelle Nr.1 (Sinusschwingung)
i1 0 1 ;Instrument 1 fuer 1 Sekunde aufrufen
```

Die Programmierung der Score Datei wird in Kapitel 3 erklärt.

In seiner jetzigen Form ist der Sinuston kaum zu gebrauchen; eine Hüllkurve würde ihn durchaus lebendiger klingen lassen. Dies kann z.B. mit folgendem Code erreicht werden:

```
instr 2 ;Beispiel 2
kenv linen 20000, .1, 1, .2
asig oscil kenv, 700, 1
out asig
endin
```

Hier wird der Opcode `linen` verwendet, der eine einfache lineare AD-Hüllkurve (**linear envelope**) generiert. Als Argumente erwartet er zunächst die maximale Amplitude, als zweites die Attackzeit, als drittes die Dauer und schliesslich die Decayzeit der Hüllkurve. Im Beispiel wird also eine 1 Sekunde lange Hüllkurve mit 0,1 Sekunden Attackzeit und 0,2 Sekunden Decayzeit erzeugt, die eine Amplitude von 20000 hat. Die Hüllkurve in der Variablen `kenv` (ein k-rate-signal), wird anschliessend in das Argument für die Amplitude des Oszillators eingesetzt. Dadurch moduliert die Hüllkurve die Amplitude des Oszillators, wodurch der Sinuston mit der Hüllkurve ausgegeben wird.

Die beiden Beispiele zeigen das grundsätzliche Prinzip der Orchestra-Programmierung. Hat man dieses Prinzip verstanden, hat man schon die Hälfte von Csound verstanden. Nachzuschlagen gilt es allerdings immer wieder die Verwendung der gewählten Opcodes (d.h. Anzahl und Reihenfolge der Argumente und unterstützte Signaltypen); die insgesamt über 400 verschiedene Opcodes können unmöglich auswendig gelernt werden. Sämtliche Opcodes werden im Public Csound Reference Manual [2] erläutert.

3 Die Programmierung der Score-Datei

Die Score-Datei besteht aus den Funktionstabellen-Definitionen und der Notenliste.

3.1 Syntax

In der Score-Datei werden keine Opcodes wie in der Orchestra-Datei verwendet; die Anweisungen bestehen hier nur aus einzelnen Buchstaben, die die Anweisung charakterisieren, und den darauf folgenden Zahlenkolonnen, den Parametern.

Anweisung <p1> <p2> <p3> ...

Die Parameter (*p-fields*) werden von der Anweisung ausgehend durchnummeriert; das p-field direkt nach der Anweisung ist p1, das darauf folgende p2 usw. Jedes dieser p-fields kann entweder aus einem numerischen Wert oder aus einer arithmetischen Operation (in eckigen Klammern: [...]) bestehen. Die p-fields werden in der Score-Datei (anders als die Argumente in der Orchestra-Datei) nicht durch Kommata, sondern durch (beliebig viele) Leerstellen getrennt. Kommentare können wie in der Orchestra-Datei mit vorangestelltem Semikolon eingefügt werden.

Ein Beispiel: In Beispiel 1 in Abschnitt 2.2 wurde in der Score-Datei eine Funktionstabelle mit dem f-statement aufgerufen:

```
f1 0 8192 10 1 ;Funktionstabelle Nr. 1
```

Das f ist die Anweisung; die 1 nach dem f entspricht p1; die 8192 entspricht p2, die 10 p3 und die 1 p5.

3.2 Die GEN-Routinen

Funktionstabellen werden mit dem *f-statement* (Funktionstabellen-Anweisung) definiert. Ein f-statement beginnt mit einem »f« und hat folgende Parameter:

f <Nummer> <Zeitpunkt> <Größe> <GEN-Routine> <p5> <p6> ...

Die Nummer der Funktionstabelle dient der Bezeichnung.

Der Zeitpunkt der Berechnung spielt dann eine Rolle, wenn der Arbeitsspeicher nicht für alle im Stück benötigten Tabellen ausreicht. Erstellt man eine Funktionstabelle mit der gleichen Nummer zu einem späteren Zeitpunkt, so wird die alte Funktionstabelle gelöscht und der zuvor von ihr beanspruchte Speicherplatz für die neue Tabelle freigegeben.

3 Die Programmierung der Score-Datei

GEN-Routine	Funktion
1	importiert Audio-Datei in Funktionstabelle
2	manuell gesetzte Werte
3	Polynomfunktion
4	Normalisierungsfunktion
5,7,16,25,27	lineare und exponentielle Segmente
6,8	kubische und spline Polynomfunktionen
9,10,19,33,34	Summierung von Sinusfunktionen
11	Summierung von Cosinusfunktionen
12	Besselfunktion
13,14,15	Tschebyscheff-Polynome
17	Sprungfunktion
18	setzt aus existierenden Funktionstabellen eine Neue zusammen
20	Fensterfunktionen (Hamming, Hanning, Blackman, ...)
21,40,41,42	verschiedene Arten von Zufallsverteilungen
23,28	liest Zahlenwerte aus einer ASCII-Datei
24	skaliert Werte aus einer anderen Funktionstabelle
30	extrahiert Harmonische aus einer Funktionstabelle
31,32	mischt Wellenformen aus anderen Funktionstabellen

Tabelle 3.1: Überblick über die verschiedenen GEN-Routinen; die Angaben beziehen sich auf Version 4.19.

Die Größe der Funktionstabelle entspricht der Menge der Adressen, denen Werte zugeordnet werden können. Die hier anzugebende Zahl muss eine Potenz von zwei (2^n) oder, in bestimmten Fällen, eine Potenz von zwei plus eins ($2^n + 1$) sein. Je größer dieser Wert gewählt wird, desto größer ist die Anzahl der Werte, mit denen die Funktion dargestellt wird; je mehr Werte, desto genauer wird die Funktion dargestellt. Allerdings wird bei großen Werten auch mehr Arbeitsspeicher benötigt.

Die Unterprogramme zur Berechnung der Funktionstabellen werden *GEN-Routinen* genannt. Jede GEN-Routine ist auf die Generierung eines bestimmten Typs von Funktion ausgelegt (siehe Tabelle 3.1).

Ein Beispiel: In einer Funktionstabelle soll eine Sinusschwingung generiert werden, die dann in der Orchestra-Datei von einem Oszillator abgespielt werden soll (siehe Beispiel 1 in Kapitel 2.2).

```
f1 0 8192 10 1 ;Funktionstabelle Nr. 1
i1 0 1 ;Instrument 1 fuer 1 Sekunde aufrufen
```

Hier wird die Funktionstabelle Nr. 1 ($p1=1$) zum Zeitpunkt 0 ($p2=0$) mit einer Größe von 8192 Punkten ($p3=8192$) definiert. Die GEN-Routine Nr. 10 ($p4=10$) wird mit dem Parameter 1 ($p5=1$) aufgerufen. Wie in Tabelle 3.1 ersichtlich, erzeugt die GEN-Routine Nr. 10 eine Summe von Sinusschwingungen. In [2] erfährt man, dass die Parameter ab $p5$ die relativen Amplituden der einzelnen Sinusschwingungen angeben, deren Frequenzen

automatisch ganzzahlige Vielfache der Grundfrequenz sind. Pro zusätzlichem Parameter wird also eine weitere Sinusschwingung addiert. Im Beispiel ist lediglich die Amplitude für die erste Sinusschwingung angegeben, also erzeugt die GEN-Routine genau eine Sinusschwingung mit der Amplitude 1.

3.3 Die Notenliste

In der Notenliste stehen die Noten bzw. die Noten-Anweisungen. Diese Anweisungen rufen zu einer bestimmten Zeit und für eine bestimmte Dauer ein Instrument auf; daher werden sie *i-statements* (instrument-statements) genannt.

Ein i-statement beginnt mit einem »i« und hat folgendes Format:

```
i <Nummer> <Startzeit> <Notendauer> <p4> <p5> <p6> ...
```

Die ersten drei Parameter sind für jedes i-statement festgelegt. p1 gibt an, welches Instrument aus der Orchestra-Datei aufgerufen werden soll. Die Startzeit und die Notendauer (p2 und p3) geben an, wann und wie lange das Instrument erklingen soll. Sofern kein anderes Bezugstempo definiert ist, ist die Zeiteinheit die Sekunde.

Die Score-Datei aus Beispiel 1:

```
f1 0 8192 10 1 ;Funktionstabelle Nr. 1 (Sinusschwingung)
i1 0 1 ;Instrument 1 fuer 1 Sekunde aufrufen
```

Das i-statement spielt Instrument Nr. 1 (p1=1) zum Zeitpunkt 0 (p2=0) für 1 Sekunde (p3=1) ab.

Weiterhin kann jedes i-statement beliebig viele Parameter an das Instrument übergeben; im Instrument werden die p-fields wie Variablen behandelt, die den Wert des p-fields im i-statement repräsentieren. Mit anderen Worten: Die p-fields eines i-statements sind i-rate-signals, die durch ihre Verwendung im Instrument definiert werden. So können in der Score-Datei für jede Note neue Parameterwerte übergeben werden, wodurch sich der Klang entsprechend Note für Note ändert.

Wie die Anweisungen in der Orchestra-Datei werden die Noten-Anweisungen Zeile für Zeile gelesen; in jeder Zeile kann ein i-statement stehen. i-statements müssen nicht in der Reihenfolge stehen, in der sie erklingen sollen. Es können zwei verschiedene Noten zur selben Zeit beginnen, also gleichzeitig (mehrstimmig) erklingen, indem einfach für p2 die gleichen Werte eingegeben werden. Hierbei gilt es, das in Abschnitt 1.1 erläuterte Clipping wohlbedacht zu vermeiden.

Mit diesen Mitteln wird Beispiel 3 nun um Amplituden- und Frequenz-Steuerung aus der Score-Datei ergänzt; außerdem wird die statische Hüllkurve (die bisher immer genau eine Sekunde lang war) an die Notenlänge angepasst:

```
instr 4 ;Beispiel 4
kenv linen 1, p3*.2, p3, .2
asig oscil p4, p5, 1
out asig*kenv
endin
```

3 Die Programmierung der Score-Datei

Die Hüllkurve passt ihre Länge jetzt immer der Dauer (p3) der Note an; außerdem wird die Attackzeit aus der Notendauer errechnet ($p3 \cdot 2$), sie beträgt ein Fünftel der Notendauer. Amplitude und Frequenz des Oszillators werden von p4 und p5 gesteuert.

In der dazugehörigen Score-Datei spielen wir eine mehrstimmige, dynamische Melodie:

```
f1 0 8192 10 1 ;Funktionstabelle Nr. 1
;erste Stimme
;Instr Start Dauer Ampl. Frequenz
i4 0 .8 .8 18000 350
i4 .8 .8 .8 10000 800
i4 1.6 .6 .6 10000 200
i4 2.2 .4 .4 5000 100
i4 3.1 1 1 10000 700
i4 4 1 1 10000 1050
;zweite Stimme
i4 .8 1.6 1.6 10000 400
i4 3.1 1 1 8000 350
i4 3.1 1 1 8000 175
;dritte Stimme
i4 .8 1 1 10000 600
i4 2.2 1.8 1.8 12000 1050
i4 4 1 1 10000 1400
```

Da in Instrument 4 anstatt Amplituden- und Frequenzwerten die Variablen p4 und p5 stehen, wird aus der Score-Datei der Wert von p4 und p5 im i-statement zu diesem Instrument gesendet. So erklingt als erstes ein 0.8 Sekunden langer Ton mit einer Amplitude von 18000 und einer Frequenz von 350 Hz (siehe erste Zeile in der Notenliste). Als nächstes erklingen drei Töne gleichzeitig mit unterschiedlichen Notendauern und Frequenzen (siehe unter zweite Stimmen und dritte Stimme). Dann wird in der ersten Stimmen zum Zeitpunkt 1,6 der nächste Ton gespielt usw.

Shortcuts

Zur Vereinfachung der Erstellung von Notenlisten existieren einige Hilfs-Symbole oder Shortcuts, von denen hier nur zwei angesprochen werden sollen.

- Das sogenannte Carry-Symbol besteht aus einem einfachen Punkt (.); es kopiert den Wert des gleichen p-fields vom vorherigen i-statement; sofern sich in der Zeile kein Parameter mehr verändert, kann die Zeile einfach leer gelassen werden, die Parameter des vorigen i-statements werden automatisch übernommen.

Exklusiv für die Startzeit der Note bzw. p2 kann das Symbol + verwendet werden; es entspricht Startzeit plus Dauer des vorherigen i-statements.

3 Die Programmierung der Score-Datei

Zum Beispiel werden diese Zeilen

```
i4      0      1      20000   700
i4      +      1.5    .      600
i4      +      1
```

in diese Zeilen übersetzt:

```
i4      0      1      20000   700
i4      1      1.5    20000   600
i4      2.5    1      20000   600
```

- Das Ramp-Symbol (<) interpoliert linear zwischen zwei Parametern.

Die Zeilen

```
i4      0      0.5    20000   200
i4      +      .      .      <
i4      +
i4      +      .      .      500
```

entsprechen also den den Zeilen:

```
i4      0      .5    20000   200
i4      .5    .5    20000   300
i4      1      .5    20000   400
i4      1.5  .5    20000   500
```

Es können auch exponentielle Interpolation mit den Klammern (und) sowie Zufallsverteilung mit der Tilde ~ veranlasst werden. Außerdem existiert ein einfaches Makro-System, das nahezu beliebigen Coden mit einer Zeichenfolge repräsentiert; dies kann die Arbeit mit der Score-Datei erheblich vereinfachen.

3.4 Sonstige Score-statements

Zur Komposition braucht es mehr als nur Noten; es existieren eine Menge weiterer Anweisungen, die professionelles Komponieren möglich machen.

f 0-statement: Mit dem f 0-statement wird Stille in das Stück eingefügt.

t-statement (tempo): Mit dieser Anweisung wird das Tempo des Stücks in Beats-per-Minute festgelegt. Auch Verlangsamungen und Beschleunigungen (musikalisch gesprochen ritardandi und accelerandi) können hiermit veranlasst werden.

a-statement (advance): Mit dem a-statement können bestimmte Teile der Score-Datei bei der Berechnung übersprungen werden. Dies ist besonders nützlich, wenn nur bestimmte Teile der Score-Datei berechnet und abgehört werden sollen, also z.B. in der Entwicklungsphase eines Stücks.

3 Die Programmierung der Score-Datei

b-statement (base time): Mit dem b-statement kann die intern gezählte Zeit auf einen bestimmten Wert gestellt werden.

v-statement (time varying): Mit diesem statement kann ein Multiplikator für die Startzeiten und Dauern angegeben werden. So wird ein Tempowechsel erzeugt, indem die Startzeiten und Notendauern entsprechend angepasst werden.

s-statement (section end): Das s-statement markiert das Ende einer Sektion. In der neuen Sektion werden die gezählten Beats auf Null zurückgesetzt.

r-statement (repeat): Mit dem r-statement können die darauf folgenden Anweisungen bis zum nächsten s-, r- oder e-statement beliebig oft wiederholt werden.

m,n-statement (mark): Das m-statement setzt eine benannte Markierung; mit dem n-statement wird ein Sprung zu einer solchen Markierung veranlasst.

x-statement: Mit dem x-statement wird der Rest der Sektion übersprungen.

e-statement (end): Das e-statement markiert das Ende der Score-Datei. Sämtliche Anweisungen nach dem e-statement werden ignoriert.

4 Die Kommandozeile

Csound wird über die Kommandozeile aufgerufen, auch die in manchen Versionen vorhandenen grafischen Oberflächen setzen nur auf sie auf. Der Aufruf von Csound erfolgt in folgendem Format:

```
csound [-Optionen] Orchestra-Datei Score-Datei
```

Die Optionen (*Flags*) sind Voreinstellungen für den Programmablauf. Man kann so z.B. das Format der Ausgabedatei einstellen, oder aber bestimmte Präprozessoren aufrufen. Sie müssen nicht unbedingt eingegeben werden. Lässt man sie weg, werden bestimmte Voreinstellungen benutzt. Wichtig bei den Flags ist die Groß- und Kleinschreibung.

Einige wichtige Flags sind z.B.:

-o<Dateiname>: Das Ergebnis der Berechnung wird in der angegebenen Datei gespeichert (also nicht in Echtzeit ausgegeben); hier muss die Datei-Endung auch angegeben werden. Außerdem sollte ein weiteres Flag gesetzt werden, das den Dateityp festlegt, da sonst eine Audio-Datei ohne Datei-Header geschrieben wird, die von den meisten Audio-Programmen nicht abgespielt wird.

Soll das Ergebnis in Echtzeit abgehört werden, ist anstatt dem Dateinamen der Begriff »dac« (für Digital/Analog Converter) anzugeben.

-W: Das Ergebnis wird als Wave-Datei gespeichert.

-A: Das Ergebnis wird als AIFF-Datei gespeichert.

-F<MIDI-Datei>: Liest MIDI-Daten aus der angegebenen MIDI-Datei.

Weitere Flags sind in [2] nachzuschlagen.

Literaturverzeichnis

- [1] Richard Boulanger, editor: *The Csound Book – perspectives in software synthesis, sound design, signal processing and programming*. MIT Press, Cambridge (Massachusetts), second edition, 2001.
- [2] John ffitch, Richard Boulanger, Jean Piché und David Boothe (editors): *The Public Csound Reference Manual*. MIT Press, Cambridge (Massachusetts), April 2002.