A Guide to Composing for Orchestra
with Symbolic Composer

Over the past 9 months I have been developing an approach to composing orchestral music using Tonality Systems *Symbolic Composer.* I'm now half way through the composition of a sequence of orchestral concertos titled *Six Concertos (Instrumentarium Novum).*

This Guide is written for composers familiar with SCOM and attempts to present as much as possible from illustrations and code rather than wordy explanation.

The rationale for working in this way came from a desire to bring together the composition of the music to be played by an orchestra with the business of the selection of instruments, in other words the orchestration. Was it possible, through using a computer environment like SCOM, to bring these two activities together? I've tried within SCOM code to device a mode of rough prototyping. The aim: to enable the composer to experience some of the essential elements of the composition before turning to the working in small detail that tends to overwhelm orchestral composition.

It was from this aim that the idea of I-Functions was born. Essentially, I-Functions is a collection of on-the-fly functions belonging to a separate (customisable) file, that may be applied using a two letter abbreviation and placed on what I have called a Scoresheet. With this in place it becomes possible for the composer to apply very quickly a processing action.

In all the concertos the music starts life as a stream of vectors converted to symbols. The symbol stream is processed to find phase boundaries and then becomes a list of phrases. This list becomes the core pitch material of the music which is applied to a seven layer orchestration of timbres types. The incidence of play and silence across this seven layer orchestration is controlled by an array-generated timesheet into which the phrases are 'poured' using the <symbol-swallow> mechanism. Very quickly it is possible to hear an 'orchestra' (of seven timbres) play a heterophonic rough prototype of the music. At this point the composer begins building the Scoresheet, processing each phrase in each of the seven layers using the I-Functions. To make composing easier the code is assembled in a series of files initially kept open on the desktop, some as reference (like the Timesheet), others as executable windows of data. Working this way enables, generally, much faster compilation and debugging.

Once a second-stage prototyping is done the music leaves Symbolic Composer and is ready to be worked on in a scorewriter (in my case Nightingale which reads zone-lengths as time-signatures). All rhythm, most part-writing, the harmonic structure, the basic orchestration is in place. It is expanded over two stages: in eleven layers, then sixteen layers.

The examples included here show page 1 of the first Concerto from the rough prototype, through the I-Function Scoresheet processing, to the eleven layer version, to the final sixteen layer version.

1. Example 1 - first rough prototype. Output from SCOM

2. Example 2 - second prototype showing how I-functions have processed the material. Output from SCOM

3. Example 3 - working now in Nightingale. Score expanded to eleven layers

4. Finished Score - Page 1

5. I-Function Tester - a demo score showing I-functions processing a single phrase sequencially.

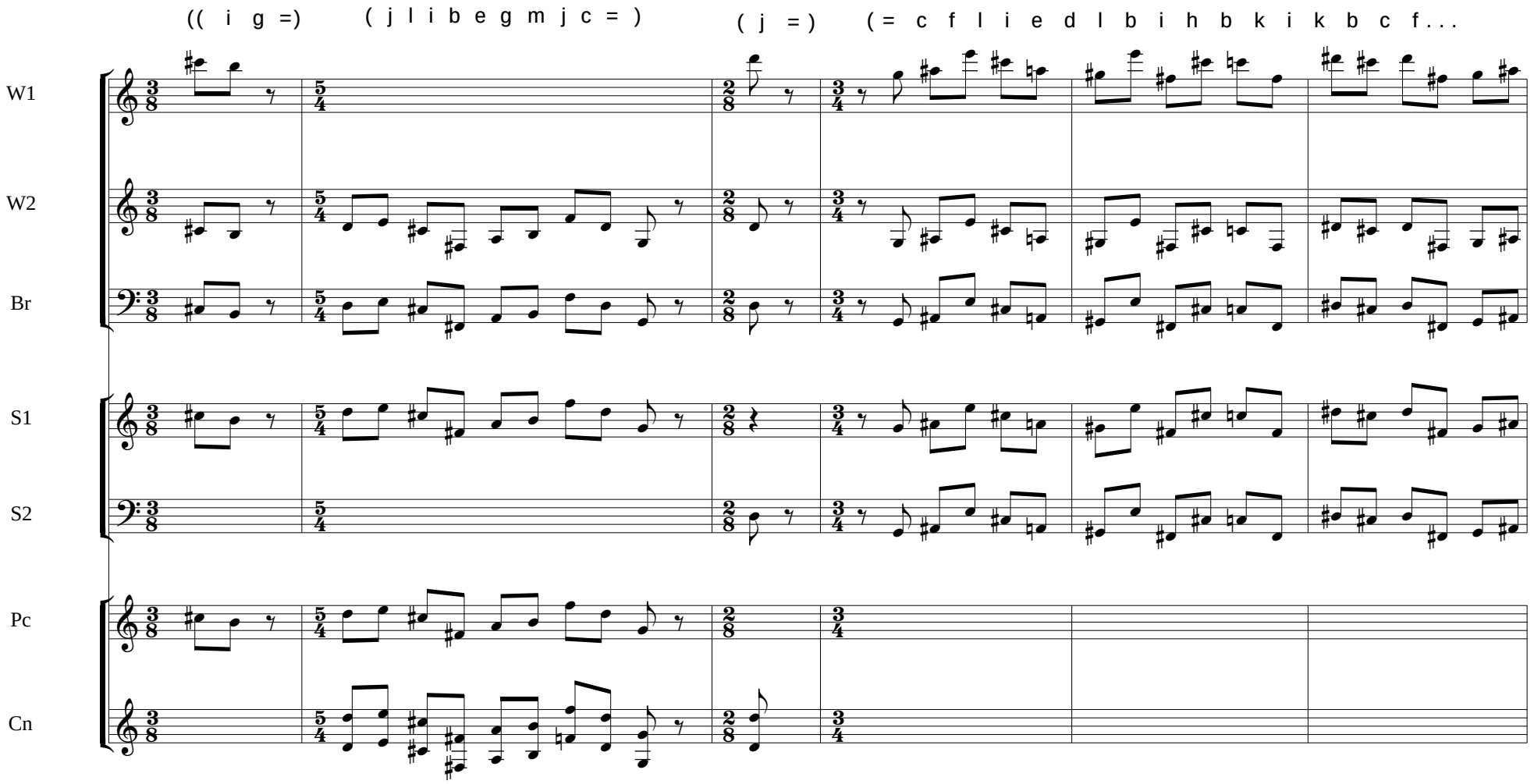
6. I-Function Tester Score - a notated and annotated output of the SCOM scorefile

7. A screen image showing how multiple windows are used to co-ordinate the process of the composition

8. The complete scorefile for Concerto 1, first movement

Nigel Morgan 29 September 2004

Concerto 1:1 (phrases 1 - 4)



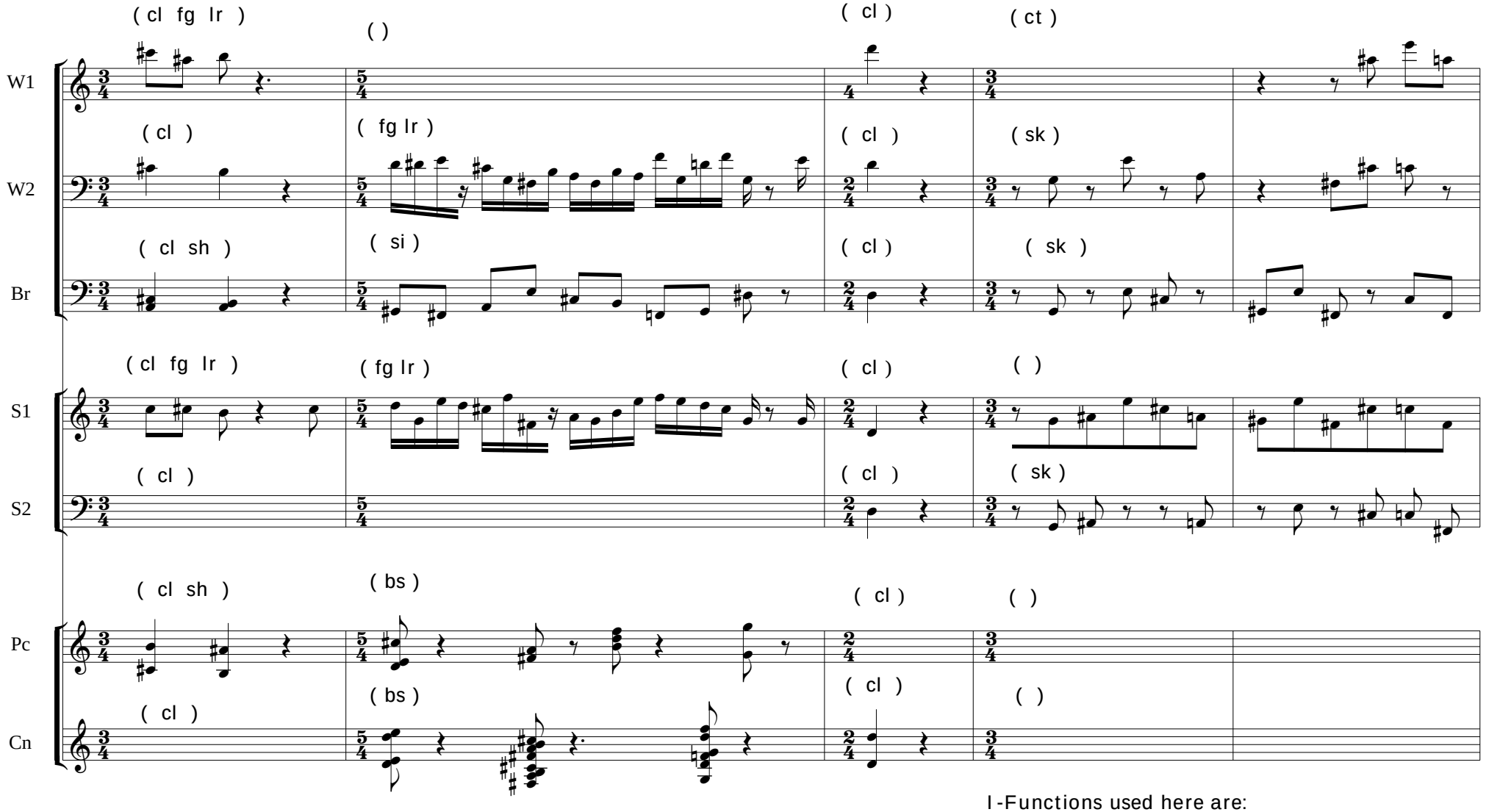
Process of Pitch Composition

* Vector output from gen-noise-white
* conversion into Symbols (scaled between a l)
* division into Phrases (using create-lists)
* Phrases picked and randomised into new order
* new order includes possibility of Repeats

Orchestration

* Stage 1 orchestration in 7 timbres
* Wind 1 & 2, Brass, Strings 1 & 2, Percussion, Continuo
* Array-based processing of instrumentation possibilities
* Production of timesheet to display incidence of play and silence

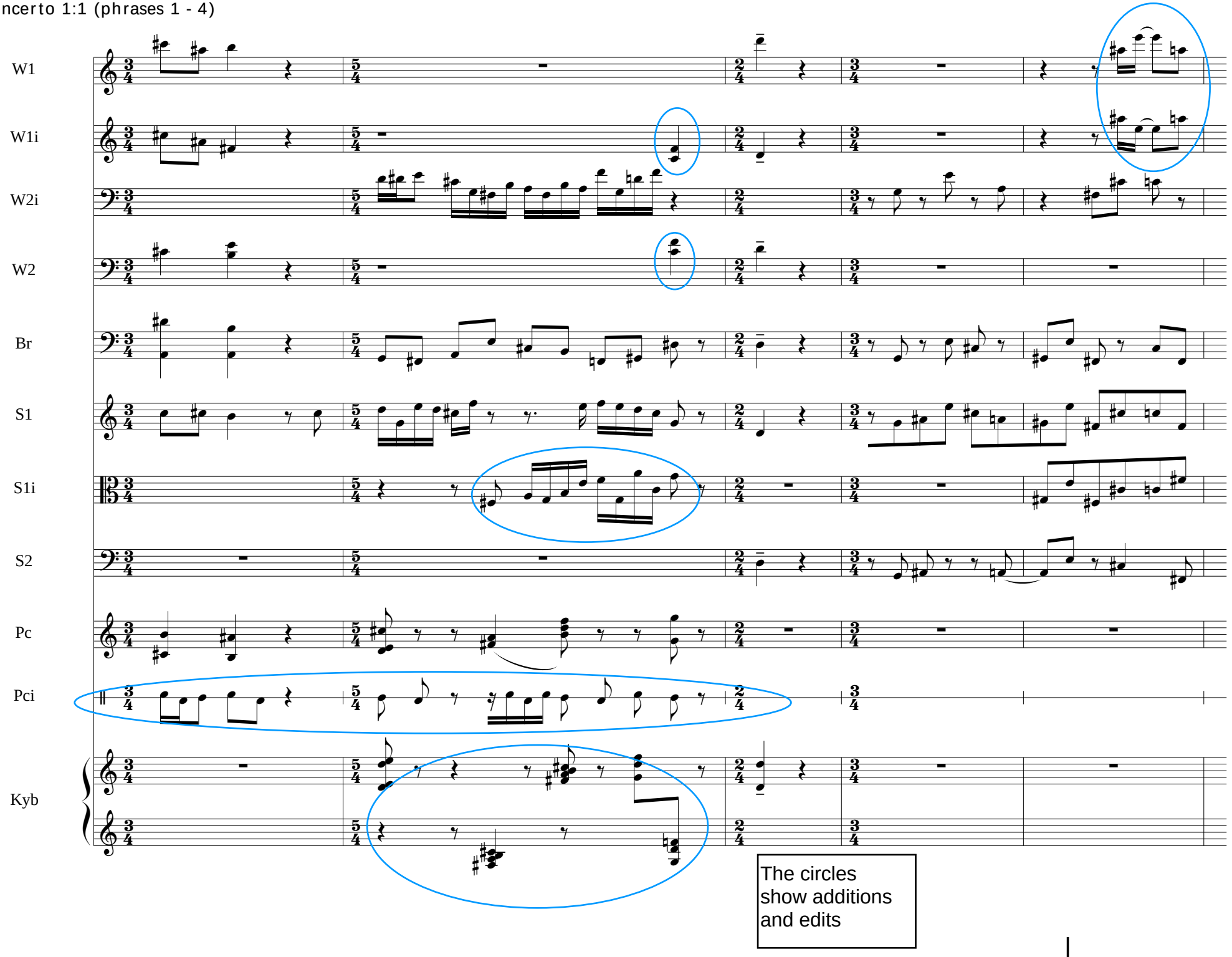
Concerto 1:1 (phrases 1 - 4)



This example shows the effect of processing each part using I-Functions.

I-Functions used here are:
cl - change-length
fg / lr - symbol-figurate (plus length-repeat)
sh - symbol-harmonize
si - symbol-inversion (@ symbol g)
sk - symbol-skip
ct - symbol-cut
bs - symbol-bundle-i

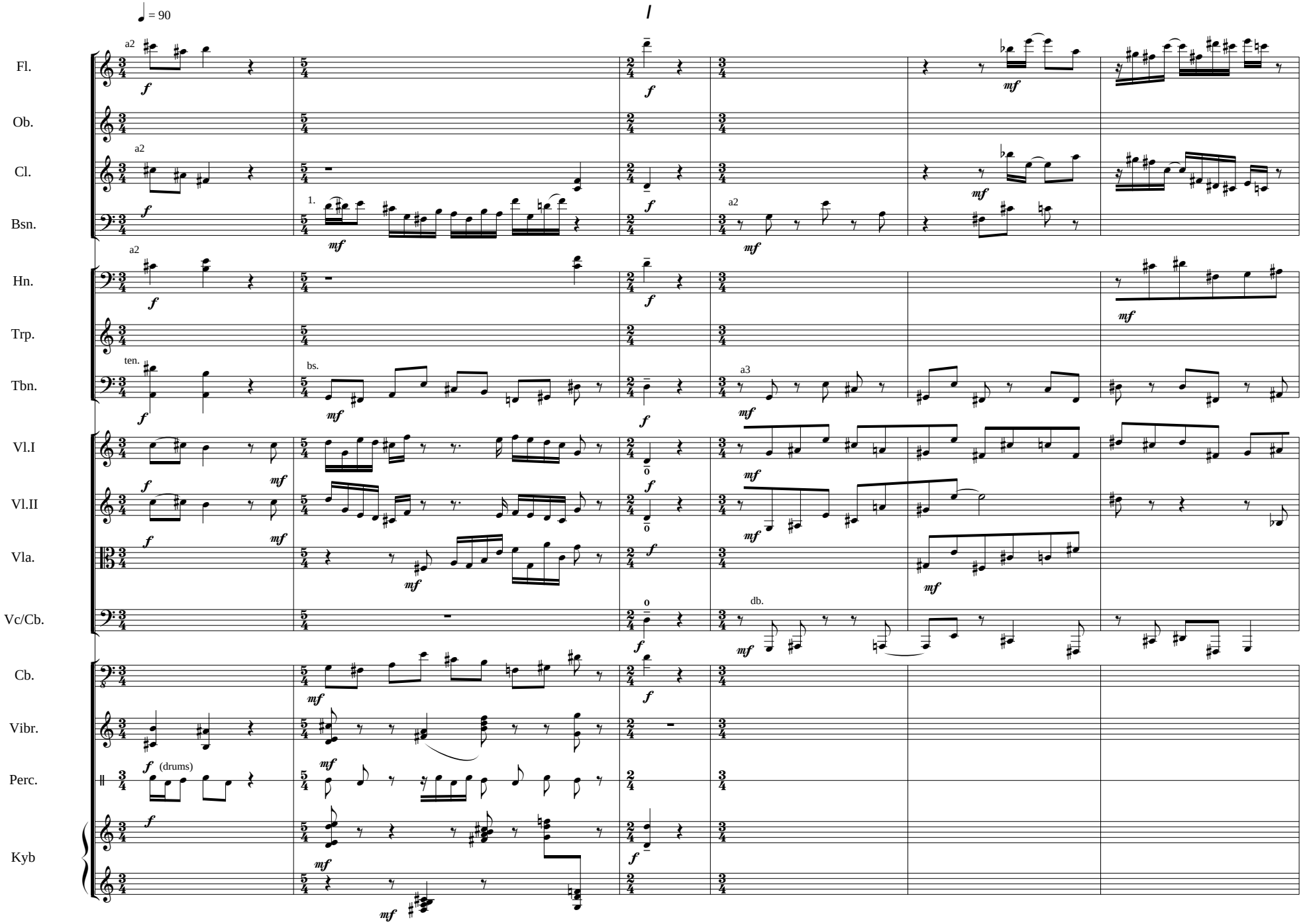
Concerto 1:1 (phrases 1 - 4)



for Susan

Concerto No.1 (Instrumentarium Novum)

I



```
;;; I-function-tester 1 6.07.04

(setq motif '((j l i b e g m j c =))) ;; from phrase 2 of Concerto 1:1
(setq r-sym (gen-repeat 36 motif))
(setq w1-sym r-sym)

(include-file "I-functions")

#|
rv - reverse
gp - gen-palindrome **
gi - g-intro **
gc - g-coda **
pr - play-registers
bs - symbol-bundle-i (x)
su - symbol-upward
sd - symbol-downward
fl - symbol-floating
or - symbol-ornamentation
cc - create-chords
sr - symbol-repeat
se - symbol-list-expand *
dt - distort-transpose
fd - filter-delete
f1/f2 - filter-delete (extracting whole-tone tonalities)
si - symbol-inversion (@ 'g)
st - sequence-transpose *
ts - symbol-thin ; incorporating symbol-cut and symbol-skip
sm - symbol-mask-i
sh - symbol-harmonize
fg - symbol-figurate
ii - interval-edit-i
ij - interval-edit-j
ik - interval-edit-k
il - interval-edit-l
lr - length-repeat (div 2)
li - length-repeat (get-random 2 4)
cl - change-length (times 2) *
xl - change-length (divide 2) *
yl - change-length (divide 4) *
zl - change-length (times 4) *
lv - length-variate
lw - length-variate(2)

|#
```

```
(setq output-all/w1
      '(()
        (rv)(rv gp xl)
        (gi lr)(gc lr)
        (pr)(bs)(su)(sd)
        (fl)(or)(cc)(sr lr)
        (se)(dt)(fd)
        (f1)(f2)
        (si)(st lr)
        (sm)(ts)(sh)(fg lr)
        (ii)(ij)(ik)(il)
        (lv)(lw)
        (lr)(li) (cl)(xl)
        (yl)(zl)))
```

This is an example of a Scoresheet

```
#| ;; output of processing <r-sym> with a sequence of processing i-functions contained in <output/w1>

((j l i b e g m j c =) ; master phrase
(= c j m g e b i l j) (= c j m g e b i l j l i b e g m j c =) ;  (rv)(rv gp lr)
(-d -b -e -l -i -g a -d -k j l i b e g m j c = =) (j l i b e g m j c v x u n q s y v o = =) ; (gi lr)(gc lr)
(v x i b q s m j o =) (= jl ib = eg = mj = = c) (b c e g i k j l m =) (m l i j i g e c b =) ;  (pr)(bs)(su)(sd)
(g j = c g l j l i =) (j l i b a e g m j c =) (pr = = = le = = = oq =) (j j l l i i b b e e g g m m j j c c = =) ; (fl)(or)(cc)(sr lr)
(b i l j l i b e g m j c =) (j m k e i l s q k =) (j l i b = g m j c =) ; (se)(dt)(fd)
 (j l = b = = = j = =) (= = i = e g m = c =) ; (f1)(f2)
(d b e l i g a d k =) (j l i b e g m j c = -e d g a -c -f c f d =) ; (si)(st lr)
(j l = = e = m = = =) (j l i b e g = j = =) (jn lp ik bc ei gb mr jp cb =) (j i l = i c b m e j g c m h i j c b = i) ; (sm)(ts)(sh)(fg lr)
(j l i b e = m j c =) (= = = = = g = = = =) (= l i b = = m j c =) (j = = = e g = = = =) ; (ii)(ij)(ik)(il)
(j l i b e g m j c =) (j l i b e g m j c =) ;  (lv)(lw)
(j l i b e g m j c =) (j l i b e g m j c =) (j l i b e g m j c =) (j l i b e g m j c =) ;  (lr)(li) (cl)(xl)
(j l i b e g m j c =) (j l i b e g m j c =)) ; (yl)(zl)

|#

(setq r-len (gen-process '(symbol-repeat x y)  (mapcar 'length r-sym) '(1/8) :list)
      z-len (zone-ratio-sc r-len))


(init-rnd 0.4468)

(include-file "process-list-w1")
```

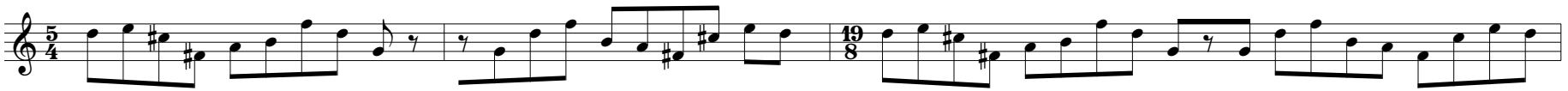
```
;; score

(def-tonality
w1 (activate-tonality (chromatic f 5))
)

(def-symbol
w1  w1-sym-o
)

(def-length
w1 w1-len-o
)


(def-velocity
w1 '(64)
)

(def-zone
default (zone-ratio-sc w1-len-o)

)

(def-channel
w1  8 ; clarinet
)

(def-tempo 90)

(compile-instrument-p "ccl;output:"  "I-function-tester"
w1
)
```

Original Phrase (from Concerto 1:1 Phrase 2)

reverse

gen-palindrome (plus change-length xl)

W1

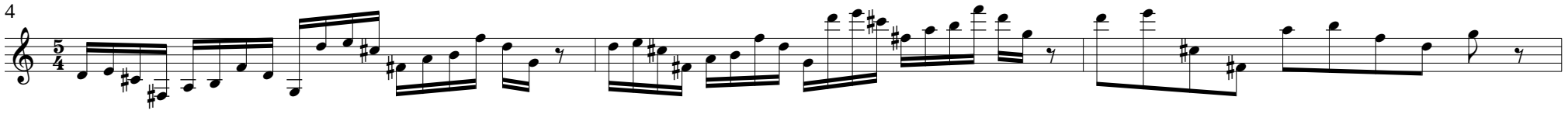


g-intro (plus length-repeat)

g-coda (plus length-repeat)

play-registers

4



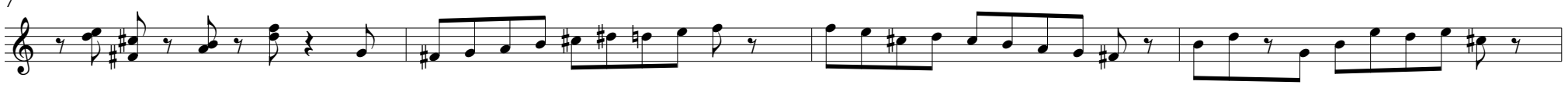
symbol-bundle-i

symbol-upward

symbol-downward

symbol-floating

7



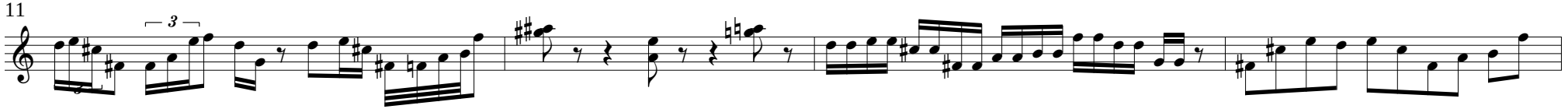
symbol-ornamentation

create-chords

symbol-repeat (plus length-repeat)

symbol-list-expand

11



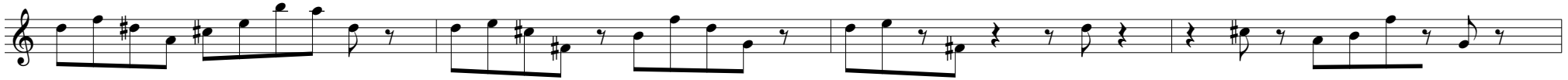
distort-transpose

filter-delete

fliter-delete (f1)

filter-delete (f2)

15



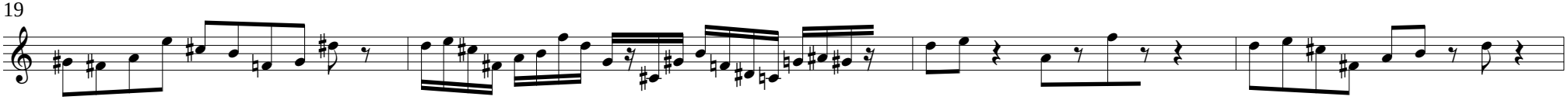
symbol-inversion (@ symbol g)

sequence-transpose (with length-repeat)

symbol-mask-i

symbol-thin

19



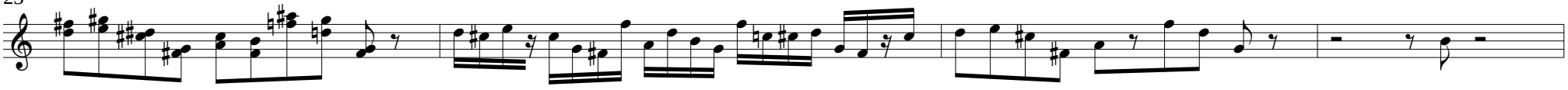
symbol-harmonize

symbol-figurate (plus length-repeat)

interval-edit-i

interval-edit-j

23



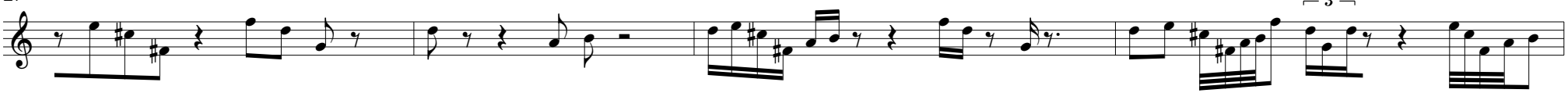
interval-edit-k

interval-edit-l

lv: length-variate

lw: length-variate

27

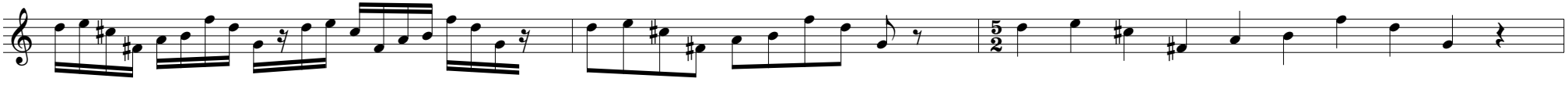


lr: length-repeat (div 2)

li: length-repeat (get-random 2 4)

cl: change-length (times 2)

31

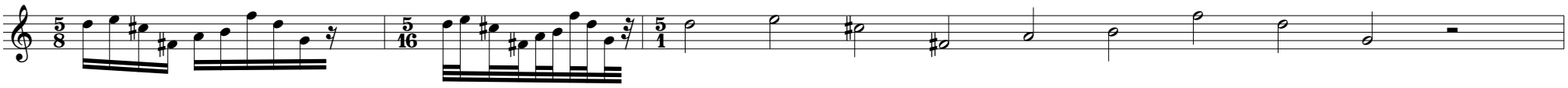


xl: change-length (div 2)

yl: change-length (div 4)

zl: change-length (times 4)

34



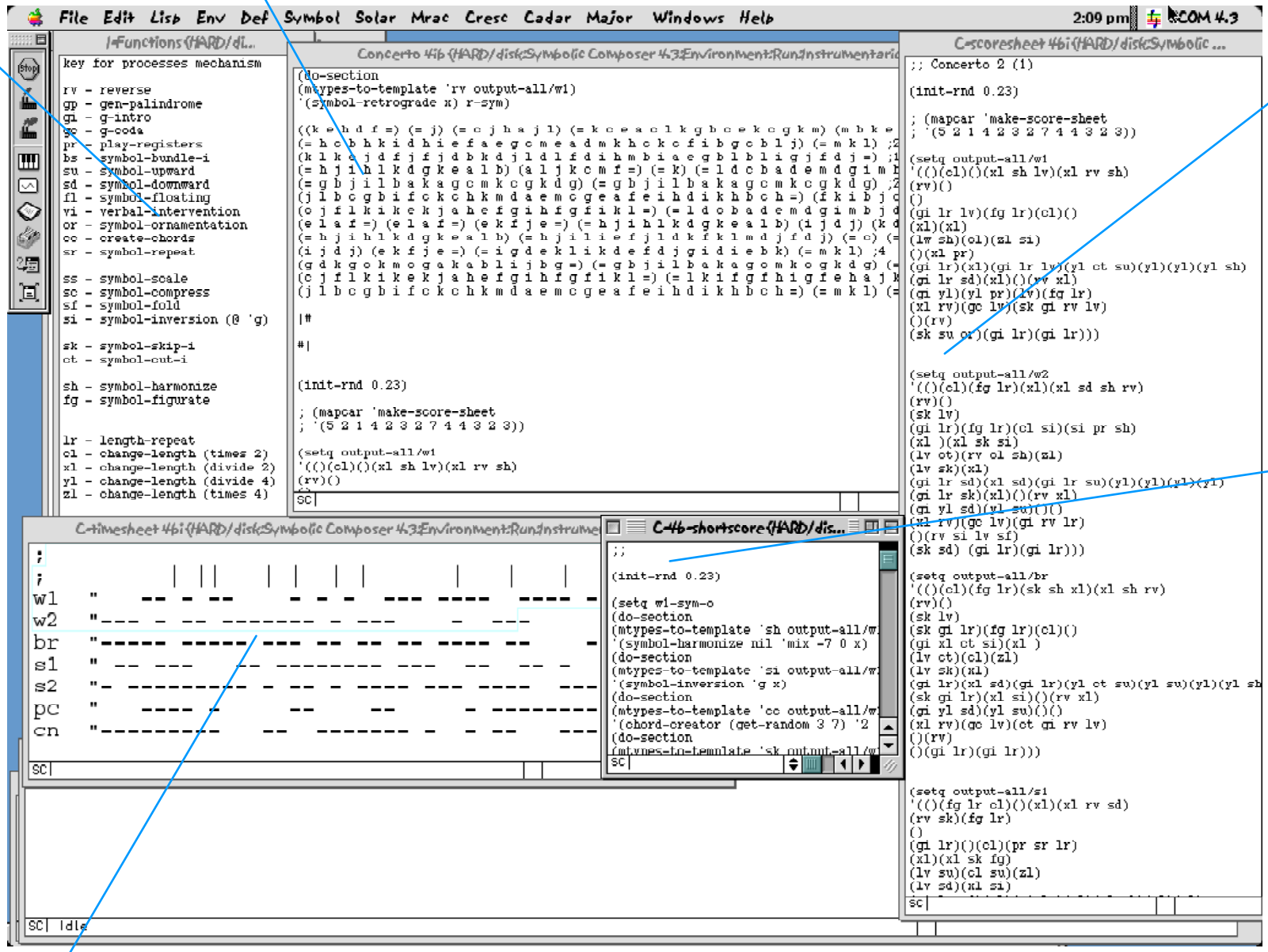
Here is the master list of symbol phrases which the Scoresheet commands process across the 7 orchestral layers

The I-Functions file contains all the processing functions needed to drive the Scoresheet. Most of the functions are adapted versions of standard SCOM tools with their parameters already set and defined

This is the Scoresheet displaying the complete processing lists of I-Functions for wind1 & 2, brass and Strings 1 parts

Here is the Process-list window which brings together the Scoresheet commands with the I-Functions. At the end of this file you'll find the standard scoring definitions: def- tonality, def-symbol etc

The timesheet is generated from the <build-array> function. Each beat/space column is equivalent to a symbol phrase in the master list. Note that the resolution of each beat space matches the length of the phrase itself (see the full code on page 17)



```
;; Concerto for Instruments 1a (Instrumentarium)  7.11.03


;; instrumental permutations


#|

w1 - wind ensemble 1 (pc,fl, ob, cl)
br - brass (trp1/2 tb bs.tb)
w2 - wind ensemble 2 (bsn, hn)
s1 - strings 1 (vn 1/2)
s2 - strings 2 (va, vc, bs)
pc - percussion (tuned/untuned)
cn - continuo (kb, bs, cn)

|#

(setq moments
(build-array
 ; column 0    1     2    3    4    5   6
      '(( w1   br    w2   s1   s2   cn  pc)     ; 0
        (cl   s1    w1   br   cn   s2  pc)     ; 1
        (s1   w1    s2   cn   w2   br  pc)     ; 2
        (s2   cn    br   w1   s1   w2  pc)     ; 3
        (cn   s2    w1   s1   br   w2  pc)     ; 4
        (br   s1    w1   w2   cn   s2  pc)     ; 5
        (pc   cn    s2   s1   w2   br  w1)     ; 6
)))

(setq lisn '(0 1 2 3 4 5 6))

(setq output-all
     (mapcar 'remove-duplicates
              (delete 'nil
                      (gen-collect 0.1922 48 :list
; 48  is number of sections created from a generation of 512 symbols
                                   '(pick-array
                                     (pick1 nil lisn)
                                     (pick1 nil lisn) 5 moments
                                     (pick1 nil '(:left
                                                  :right
                                                  :up
                                                  :down
                                                  )))))))
```

```
; output of output/all/m

#|
((pc w2 br s1 w1) (s1 pc br w2 cn) (br w2 cn s2 w1) (w1 br w2 s1 s2) (s1 br cn w2 s2)
(w1 s1 pc br w2) (w1 pc) (br w2 s1 s2 cn) (w2 s1 w1 br cn) (br s1 w2 cn s2) (s2 cn w2 br pc)
(s1 br cn w1) (cn pc w1 br w2) (pc w2 s1 w1 br) (cn s2 br w2) (pc br cn s2 s1) (cn br s2 w2)
(pc br s1 w1 w2) (w1 cn s2 s1) (w1 s2 cn w2 br) (w2 s1 s2 cn pc) (cl s1 w1 br cn)
(cl pc s2 cn br) (s2 cn w1 s1) (s2 cn pc w1 br) (cl w1 pc br cn) (pc br s1 w1 w2)
(cn s2 pc br s1) (s1 w2 pc s2 cn) (s2 cn br w1 s1) (br w1 s1 cl pc) (w1 s1 br w2 pc)
(br w2 s1 s2 cn) (w1 s1 br pc s2) (w1 s1 w2 pc s2) (s1 br cn w2) (s1 w1 w2 cn s2)
(w2 s1 br cn) (br s1 s2 cn) (cn pc w1 br w2) (cn br s1 w2) (s2 w1 s1 pc br) (cn pc w1 br w2)
(w1 pc cn s2 s1) (w2 s2 cn br) (br cn s2 pc w2) (s2 cn br w1 s1) (br cn w1 s1))
|#

;; instrumentation per section

(instrument-to-string 'w1 output-all)
;- -- -- -  ---   --- - ----  --- -- -  - ---  --
(instrument-to-string 'br output-all)
;------ ---------- - -- ---- ----- - ------ ----
(instrument-to-string 'w2 output-all)
;------ ---- --- -- --     - -  -- ---- -- - --
(instrument-to-string 's1 output-all)
;-- --- --- - - - -- -- -  ------------ -- -  --
(instrument-to-string 's2 output-all)
;  ---  - --   --- --- ---  ---  --- - -  - ----
(instrument-to-string 'pc output-all)
;--   --   - -- - -  - - ----- -- --    - --- -
(instrument-to-string 'cn output-all)
; -- -  ------ --- -------- ---  -  ------ ------

#|

;
;
w1   "- -- -- -  ---   --- - ----  --- -- -  - ---  --"
w2   "------ ---- --- -- --     - -  -- ---- -- - --  "
br   "------ ---------- - -- ---- ----- - ------ ----"
s1   "-- --- --- - - - -- -- -  ------------ -- -  --"
s2   "  ---  - --   --- --- ---  ---  --- - -  - ---- "
pc   "--   --   - -- - -  - - ----- -- --    - --- -  "
cn   " -- -  ------ --- -------- ---  -  ------ ------"

|#
```

```
(setq tsheet-lengths (mapcar 'length r-sym))
; (3 10 2 33 11 32 17 23 3 8 21 2 5 2 19 32 1 3 10 8 2 18 7 5 3 11 11 8 5 4 17 4 19 10 15 8 4 11 13 3 8 10 8 5 5 2 3 11)

; material

(setq source (gen-noise-white 512 1.0 0.2))

(setq sym (vector-to-symbol a m source))

(setq f-sym (find-change sym)
      a-sym (find-anacrusis sym)
)

(setq c-sym (c-list-rotate 0.1 (create-lists f-sym)))

(symbol-divide (mapcar 'length c-sym) 'setq 'x (flatten c-sym))

(setq x0 '(m e g e))
(setq x1 '(= i l))
(setq x2 '(= e a b g i h i d g i g d b l))
(setq x3 '(= g e h j g l))
(setq x4 '(= g i c d f e g h b d k i c l a f l d j l))
(setq x5 '(k b c a c h j k b h =))
(setq x6 '(k c a j k a d j c e c f c k l i =))
(setq x7 '(k g h c h j c =))
(setq x8 '(=))
(setq x9 '(= e k l f b f g d k))
(setq x10 '(= b c e))
(setq x11 '(m a g f l h k b d k i j l d h g h =))
(setq x12 '(=))
(setq x13 '(d f m a h g f k d g e i e h b d k l i b i l h c d i h b c j i j g d i =))
(setq x14 '(j =))
(setq x15 '(= g m f d i b j f))
(setq x16 '(d a k j e b h c j g m l a g d g h c h i k j =))
(setq x17 '(b d =))
(setq x18 '(i c g f k l i f a l =))
(setq x19 '(= a k m k h e))
(setq x20 '(= d g a c b c e))
(setq x21 '(j l i b e g m j c =))
(setq x22 '(h g l j e g i l b l =))
(setq x23 '(d b l g c m k j f h f =))
(setq x24 '(= c f l i e d l b i h b k i k b c f e i c k i d k d h j c b d l g))
(setq x25 '(j i k l i b h e a i h d =))
(setq x26 '(g =))
```

```
(setq x27 '(= i d b i g f l e i c))
(setq x28 '(g e d a m f l =))
(setq x29 '(k j e d l k c =))
(setq x30 '(h g j k e b f j d k i c l =))
(setq x31 '(g e c h e d i m b h k b d g i e i j =))
(setq x32 '(= h l h k h l k h b j l b i g f l f h e j g k f i k g d c b c d))
(setq x33 '(= b j e l g i l e f j b l e d k j h f))
(setq x34 '(= c i l f b h l c b f j))
(setq x35 '(= g e l b))
(setq x36 '(= f d b k e a d k d b k b k m k f i e))
(setq x37 '(i g =))
(setq x38 '(= f c d g))
(setq x39 '(= f c))
(setq x40 '(a c h l =))
(setq x41 '(e c i =))
(setq x42 '(= f i))
(setq x43 '(j =))
(setq x44 '(= e i c g b a d k f c k b g l f b d c i l e j))
(setq x45 '(= k))
(setq x46 '(j h b j h l j b f e c i =))
(setq x47 '(= k))


(setq n-list (gen-random 0.1 48 (list-a-scale 0 48)))
(setq r-sym (eval-section-integer n-list 'x 'list))

(setq tsheet-lengths (mapcar 'length r-sym))
; (3 10 2 33 11 32 17 23 3 8 21 2 5 2 19 32 1 3 10 8 2 18 7 5 3 11 11 8 5 4 17 4 19 10
; 15 8 4 11 13 3 8 10 8 5 5 2 3 11)


#|
(sort-by-length 33 r-sym)

((= c f l i e d l b i h b k i k b c f e i c k i d k d h j c b d l g)
(= h l h k h l k h b j l b i g f l f h e j g k f i k g d c b c d)
(= h l h k h l k h b j l b i g f l f h e j g k f i k g d c b c d)
(d a k j e b h c j g m l a g d g h c h i k j =)
(= g i c d f e g h b d k i c l a f l d j l)
(= f d b k e a d k d b k b k m k f i e) (= f d b k e a d k d b k b k m k f i e)
(m a g f l h k b d k i j l d h g h =)
(k c a j k a d j c e c f c k l i =) (k c a j k a d j c e c f c k l i =)
(= e a b g i h i d g i g d b l)
(j i k l i b h e a i h d =)
(h g l j e g i l b l =)
```

```
(= i d b i g f l e i c)
(i c g f k l i f a l =) (i c g f k l i f a l =) (i c g f k l i f a l =)
(j l i b e g m j c =) (j l i b e g m j c =) (j l i b e g m j c =)
(= e k l f b f g d k)
(k g h c h j c =) (k g h c h j c =)
(= d g a c b c e) (= d g a c b c e)
(g e d a m f l =) (g e d a m f l =)
(= g e h j g l)
(a c h l =) (a c h l =) (a c h l =)
(= g e l b) (= f c d g)
(m e g e) (m e g e) (m e g e)
(= f c) (= f c)
(= i l) (= i l) (i g =) (i g =)
(= k) (= k) (= k) (= k)
(j =) (=))
|#


#| ; master list

((i g =) (j l i b e g m j c =) (j =) (= c f l i e d l b i h b k i k b c f e i c k i d k d h j c b d l g) ; 1
(i c g f k l i f a l =) (= h l h k h l k h b j l b i g f l f h e j g k f i k g d c b c d) ;2
(k c a j k a d j c e c f c k l i =) (d a k j e b h c j g m l a g d g h c h i k j =) (i g =)  ;3
(g e d a m f l =) (= g i c d f e g h b d k i c l a f l d j l) (= k) (= f c d g) (= k) ; 4
(= f d b k e a d k d b k b k m k f i e) ; 5
(= h l h k h l k h b j l b i g f l f h e j g k f i k g d c b c d) (=) (= i l) ; 6
(j l i b e g m j c =) (= d g a c b c e) (= k) (m a g f l h k b d k i j l d h g h =) ; 7
(= g e h j g l) (a c h l =) (= i l) (i c g f k l i f a l =) (i c g f k l i f a l =) ; 8
(g e d a m f l =) (= g e l b) (m e g e) (k c a j k a d j c e c f c k l i =) (m e g e) ; 9
(= f d b k e a d k d b k b k m k f i e) (= e k l f b f g d k) (= e a b g i h i d g i g d b l) ;10
(k g h c h j c =) (m e g e) (= i d b i g f l e i c) (j i k l i b h e a i h d =) (= f c) ; 11
(= d g a c b c e) (j l i b e g m j c =) (k g h c h j c =) (a c h l =) (a c h l =) ; 12
(= k) (= f c) (h g l j e g i l b l =)) ; 13

|#
```

```
;; scoresheet - I-function scoring (see list of I-functions)
```

The following section of code is arranged in 2 columns to make the score-file easier to study.

```
(setq output-all/w1
'((cl fg lr) () (cl) (ct)
(cl) (ct)
(pr or) () (cl fg lr)
(xl) () () () ()
()
() (cl) (cl)
() (xl) (cl) (or)
() () () () (cl)
(xl) (xl) (cl) () (cl)
()(or)(or pr)
(xl) (cl fg lr) () () (cl fg lr)
(xl) (or) (xl) () ()
() (cl) (pr)))

(setq output-all/w2
'((cl ) (fg lr) (cl) (sk)
(cl) (sk)
() (sk) (cl sh)
(xl) (sk bs) () (sh) ()
(pr)
(ct is) (cl) (cl fg lr)
() (xl) (cl) (or)
() () () () (cl)
(xl) (xl) (cl) (sk) (cl)
()()()
(xl) (cl) () () (cl)
(xl) () (xl) () ()
() (cl) ()))

(setq output-all/br
'((cl sh ) (si) (cl) (sk)
(cl si) (sk cc)
() (sk) (cl)
(xl) (sk cc) () () ()
(pr)
(ct) (cl) (cl)
() (xl) (cl) (sd)
() () () () (cl)
(xl) (xl) (cl) (ct) (cl)
(su pr sk)(or)(ct bs)
(xl) (cl) () () (cl)
(xl) () (xl) () ()
() (cl) (fg lr)))

(setq output-all/s1
'((cl fg lr) (fg lr) (cl) ()
(cl su) (pr)
() (or) (cl fg lr)
(xl) () () () ()
(pr)
(or) (cl) (cl )
(or) (xl) (cl) (or)
() (fg lr) () () (cl)
(xl) (xl) (cl) () (cl)
(sk)(sk pr)(sk)
(xl) (cl) () () (cl)
(xl) () (xl) () ()
() (cl) (su)))

(setq output-all/s2
'((cl ) () (cl) (sk)
(cl sk lr fg) ()
() () (cl)
(xl) () () () ()
(pr)
(sk) (cl) (cl fg lr)
() (xl) (cl) ()
() () () () (cl)
(xl) (xl) (cl) () (cl)
(si or)(sk sd)(sk su)
(xl) (cl sd) () () (cl)
(xl) (sd) (xl) (sd) (sd)
() (cl) (su)))

(setq output-all/pc
'((cl sh ) (bs) (cl) ()
(cl) ()
(cc) () (cl)
(xl) (ct or) () (sh) ()
(pr)
() (cl) (cl)
() (xl) (cl) ()
() () () () (cl)
(xl) (xl) (sk ct cl) (ct) (cl ct sh)
()(sk cc)(sk cc)
(xl) (cl) () () (cl)
(xl) (cc) (xl) () ()
() (cl) ()))
```

```
(setq output-all/cn
'((cl ) (bs) (cl) ()
(cl bs ct) ()
() (bs) (cl fg lr)
(xl) () () () ()
(pr)
(sk bs) (cl) (cl)
() (xl) (cl) ()
() () () () (cl)
(xl pr) (xl pr) (cl) () (cl)
(bs)()()
(xl) (cl) () () (cl)
(xl) () (xl) () ()
() (cl sk sh) (sk cc)))
```

```
(defun zone-convertor (list-of-zones timesheet-string)
"maps beat/space values onto a list of zones"
(do-quietly
 (get-ratio-sc (mapcar (function (lambda (x y) (*  (get-ratio-cl x) y)))
        (mapcar 'get-ratio list-of-zones)
         (string-to-length '1 timesheet-string)))))

; (zone-convertor '(11/8 12/8 4/4 7/8 11/8 12/8 5/8 7/8) "- - -- -")
; (11/8 -3/2 1/1 -7/8 11/8 3/2 -5/8 7/8)
```

```
;; note-length, symbol-mapping and zone processing

(setq r-len (gen-process '(symbol-repeat x y)  (mapcar 'length r-sym) '(1/8) :list)
      z-len (zone-ratio-sc r-len))

(setq w1-zone (zone-convertor z-len (instrument-to-string 'w1 output-all))
      w2-zone (zone-convertor z-len (instrument-to-string 'w2 output-all))
      br-zone (zone-convertor z-len (instrument-to-string 'br output-all))
      s1-zone (zone-convertor z-len (instrument-to-string 's1 output-all))
      s2-zone (zone-convertor z-len (instrument-to-string 's2 output-all))
      pc-zone (zone-convertor z-len (instrument-to-string 'pc output-all))
      cn-zone (zone-convertor z-len (instrument-to-string 'cn output-all)))
;
```

This expression generates a timesheet string for a particular instrumental part

```
(setq w1-temp (do-section :all '(calculate-rests x)  w1-zone)
      w2-temp (do-section :all '(calculate-rests x)  w2-zone)
      br-temp (do-section :all '(calculate-rests x)  br-zone)
      s1-temp (do-section :all '(calculate-rests x)  s1-zone)
      s2-temp (do-section :all '(calculate-rests x)  s2-zone)
      pc-temp (do-section :all '(calculate-rests x)  pc-zone)
      cn-temp (do-section :all '(calculate-rests x)  cn-zone)
)
```

<calculate-rests> creates a template showing incidence of rests in a length or zone list

(calculate-rests '(1/8 -1/8 1/8 -1/8 -1/8))

```
(setq w1-len (do-section w1-temp '(lengths-to-rests x)  r-len)
      w2-len (do-section w2-temp '(lengths-to-rests x)  r-len)
      br-len (do-section br-temp '(lengths-to-rests x)  r-len)
      s1-len (do-section s1-temp '(lengths-to-rests x)  r-len)
      s2-len (do-section s2-temp '(lengths-to-rests x)  r-len)
      pc-len (do-section pc-temp '(lengths-to-rests x)  r-len)
      cn-len (do-section cn-temp '(lengths-to-rests x)  r-len)
)

(setq w1-sym (mapcar (function (lambda (x y) (symbol-swallow x y))) w1-len r-sym)
      w2-sym (mapcar (function (lambda (x y) (symbol-swallow x y))) w2-len r-sym)
      br-sym (mapcar (function (lambda (x y) (symbol-swallow x y))) br-len r-sym)
      s1-sym (mapcar (function (lambda (x y) (symbol-swallow x y))) s1-len r-sym)
      s2-sym (mapcar (function (lambda (x y) (symbol-swallow x y))) s2-len r-sym)
      pc-sym (mapcar (function (lambda (x y) (symbol-swallow x y))) pc-len r-sym)
      cn-sym (mapcar (function (lambda (x y) (symbol-swallow x y))) cn-len r-sym)
)
```

These expressions output the basic note-length data for each instrumental part

The <symbol-swallow> function is the key to the whole process which allows instrumental pitches (symbols) and rhythms (note-lengths) to synchronize when scored on a timesheet

The following section of code is arranged in 2 columns to make it easier

```
;; process-lists / I-functions

(setq w1-sym-o
(do-section
(mtypes-to-template 'sh output-all/w1)
'(symbol-harmonize nil 'mix -7 0 x)
(do-section
(mtypes-to-template 'si output-all/w1)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'cc output-all/w1)
'(chord-creator '(2) '2  x)
(do-section
(mtypes-to-template 'sk output-all/w1)
'(symbol-skip-i 7 12 x)
(do-section
(mtypes-to-template 'ct output-all/w1)
'(symbol-cut-i 7 12 x)
(do-section
 (mtypes-to-template 'fg output-all/w1)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/w1)
  '(play-registers 2 '(-12 12 -7 -5) x)
  (do-section
 (mtypes-to-template 'bs output-all/w1)
'(symbol-bundle-i 2 x)
  (do-section
 (mtypes-to-template 'su output-all/w1)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/w1)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/w1)
 '(symbol-floating x)
(do-section (mtypes-to-template 'or output-all/w1) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/w1)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 w1-sym))))))))))))))


(setq w1-len-o
(do-section (mtypes-to-template 'xl output-all/w1)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'cl output-all/w1)
'(change-length :times 2 x :ratio)
(do-section (mtypes-to-template 'lr output-all/w1) ; length-
repeat
```

```
'(length-repeat 2 x)
(p-replace-sections (mtypes-to-template 'or output-all/w1)
(do-section (mtypes-to-template 'or output-all/w1) '(flatten x)
(do-section
 (mtypes-to-template 'or output-all/w1)
 '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 w1-sym))
 w1-len)))))

(setq w2-sym-o
(do-section
(mtypes-to-template 'sh output-all/w2)
'(symbol-harmonize nil 'mix -7 0 x)
(do-section
(mtypes-to-template 'si output-all/w2)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'sk output-all/w2)
'(symbol-skip-i 7 12 x)
(do-section
(mtypes-to-template 'ct output-all/w2)
'(symbol-cut-i 7 12 x)
(do-section
(mtypes-to-template 'cc output-all/w2)
'(chord-creator '(2) '2 x)
(do-section
 (mtypes-to-template 'fg output-all/w2)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/w2)
  '(play-registers 2 '(-12 12 -7 -5) x)
  (do-section
 (mtypes-to-template 'bs output-all/w2)
 '(symbol-bundle-i 2 x)
  (do-section
 (mtypes-to-template 'su output-all/w2)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/w2)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/w2)
 '(symbol-floating x)
(do-section (mtypes-to-template 'or output-all/w2) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/w2)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 w2-sym))))))))))))))

(setq w2-len-o
(do-section (mtypes-to-template 'xl output-all/w2)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'cl output-all/w2)
'(change-length :times 2 x :ratio)
(do-section (mtypes-to-template 'lr output-all/w2) ; length-
repeat
'(length-repeat 2 x)
(p-replace-sections (mtypes-to-template 'or output-all/w2)
(do-section (mtypes-to-template 'or output-all/w2) '(flatten x)
(do-section
 (mtypes-to-template 'or output-all/w2)
 '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 w2-sym))
 w2-len)))))

(setq br-sym-o
(do-section
(mtypes-to-template 'sh output-all/br)
'(symbol-harmonize nil 'mix -7 0 x)
(do-section
(mtypes-to-template 'si output-all/br)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'sk output-all/br)
'(symbol-skip-i 7 12 x)
(do-section
(mtypes-to-template 'ct output-all/br)
'(symbol-cut-i 7 12 x)
(do-section
(mtypes-to-template 'cc output-all/br)
'(chord-creator '(2) '2 x)
(do-section
 (mtypes-to-template 'sf output-all/br)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/br)
  '(play-registers 2 '(-12 12 -7 -5) x)
  (do-section
 (mtypes-to-template 'bs output-all/br)
 '(symbol-bundle-i 2 x)
```

```
  (do-section
 (mtypes-to-template 'su output-all/br)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/br)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/br)
 '(symbol-floating x)
(do-section (mtypes-to-template 'or output-all/br) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/br)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 br-sym))))))))))))))

(setq br-len-o
(do-section (mtypes-to-template 'xl output-all/br)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'cl output-all/br)
'(change-length :times 2 x :ratio)
(do-section (mtypes-to-template 'lr output-all/br) ; length-
repeat
'(length-repeat 2 x)
(p-replace-sections (mtypes-to-template 'or output-all/br)
(do-section (mtypes-to-template 'or output-all/br) '(flatten x)
(do-section
 (mtypes-to-template 'or output-all/br)
 '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 br-sym))
 br-len)))))

(setq s1-sym-o
(do-section
(mtypes-to-template 'sh output-all/s1)
'(symbol-harmonize nil 'mix -7 0 x)
(do-section
(mtypes-to-template 'si output-all/s1)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'cc output-all/s1)
'(chord-creator '(2) '2 x)
(do-section
(mtypes-to-template 'sk output-all/s1)
'(symbol-skip-i 7 12 x)
(do-section
(mtypes-to-template 'ct output-all/s1)
'(symbol-cut-i 7 12 x)
(do-section
 (mtypes-to-template 'fg output-all/s1)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/s1)
  '(play-registers 2 '(-12 12 -7 -5) x)
  (do-section
 (mtypes-to-template 'bs output-all/s1)
 '(symbol-bundle-i 2 x)
  (do-section
 (mtypes-to-template 'su output-all/s1)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/s1)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/s1)
 '(symbol-floating x)
(do-section (mtypes-to-template 'or output-all/s1) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/s1)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 s1-sym))))))))))))))

(setq s1-len-o
(do-section (mtypes-to-template 'xl output-all/s1)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'cl output-all/s1)
'(change-length :times 2 x :ratio)
(do-section (mtypes-to-template 'lr output-all/s1) ; length-
repeat
'(length-repeat 2 x)
(p-replace-sections (mtypes-to-template 'or output-all/s1)
(do-section (mtypes-to-template 'or output-all/s1) '(flatten x)
(do-section
 (mtypes-to-template 'or output-all/s1)
 '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 s1-sym))
 s1-len)))))

(setq s2-sym-o
(do-section
(mtypes-to-template 'sh output-all/s2)
```

```
'(symbol-harmonize nil 'mix -7 0 x)
(do-section
(mtypes-to-template 'si output-all/s2)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'cc output-all/s2)
'(chord-creator '(2) '2 x)
(do-section
(mtypes-to-template 'sk output-all/s2)
'(symbol-skip-i 7 12 x)
(do-section
(mtypes-to-template 'ct output-all/s2)
'(symbol-cut-i 7 12 x)
(do-section
 (mtypes-to-template 'fg output-all/s2)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/s2)
  '(play-registers 2 '(-12 12 -7 -5) x)
  (do-section
 (mtypes-to-template 'bs output-all/s2)
 '(symbol-bundle-i 2 x)
  (do-section
 (mtypes-to-template 'su output-all/s2)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/s2)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/s2)
 '(symbol-floating x)
(do-section (mtypes-to-template 'or output-all/s2) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/s2)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 s2-sym))))))))))))))

(setq s2-len-o
(do-section (mtypes-to-template 'xl output-all/s2)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'cl output-all/s2)
'(change-length :times 2 x :ratio)
(do-section (mtypes-to-template 'lr output-all/s2) ; length-
repeat
'(length-repeat 2 x)

(p-replace-sections (mtypes-to-template 'or output-all/s2)
(do-section (mtypes-to-template 'or output-all/s2) '(flatten x)
(do-section
 (mtypes-to-template 'or output-all/s2)
 '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 s2-sym))
 s2-len)))))

(setq pc-sym-o
(do-section
(mtypes-to-template 'sh output-all/pc)
'(symbol-harmonize nil 'mix -7 0 x)
(do-section
(mtypes-to-template 'si output-all/pc)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'sk output-all/pc)
'(symbol-skip-i 7 12 x)
(do-section
(mtypes-to-template 'ct output-all/pc)
'(symbol-cut-i 7 12 x)
(do-section
(mtypes-to-template 'cc output-all/pc)
'(chord-creator '(2 3) '3 x)
(do-section
 (mtypes-to-template 'fg output-all/pc)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/pc)
  '(play-registers 2 '(-12 12 -7 -5) x)
  (do-section
 (mtypes-to-template 'bs output-all/pc)
 '(symbol-bundle-i '(2 3) x)
  (do-section
 (mtypes-to-template 'su output-all/pc)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/pc)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/pc)
 '(symbol-floating x)
(do-section (mtypes-to-template 'or output-all/pc) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/pc)
```

```
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 pc-sym)))))))))))))


(setq pc-len-o
(do-section (mtypes-to-template 'xl output-all/pc)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'cl output-all/pc)
'(change-length :times 2 x :ratio)
(do-section (mtypes-to-template 'lr output-all/pc) ; length-
repeat
'(length-repeat 2 x)
(p-replace-sections (mtypes-to-template 'or output-all/pc)
(do-section (mtypes-to-template 'or output-all/pc) '(flatten x)
(do-section
 (mtypes-to-template 'or output-all/pc)
 '(cdr (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 pc-sym))
 pc-len)))))

(setq cn-sym-o
(do-section
(mtypes-to-template 'sh output-all/cn)
'(symbol-harmonize nil 'mix -7 0 x)
(do-section
(mtypes-to-template 'si output-all/cn)
'(symbol-inversion 'g x)
(do-section
(mtypes-to-template 'sk output-all/cn)
'(symbol-skip-i 7 12 x)
(do-section
(mtypes-to-template 'ct output-all/cn)
'(symbol-cut-i 7 12 x)
(do-section
(mtypes-to-template 'cc output-all/cn)
'(chord-creator '(2) '4 x)
(do-section
 (mtypes-to-template 'fg output-all/cn)
 '(symbol-figurate x)
(do-section
 (mtypes-to-template 'pr output-all/cn)
  '(play-registers 2 '(-12 12 -7 -5) x)
  (do-section
 (mtypes-to-template 'bs output-all/cn)
 '(symbol-bundle-i '(2 4 3 4) x)
  (do-section
 (mtypes-to-template 'su output-all/cn)
 '(symbol-upward x)
  (do-section
 (mtypes-to-template 'sd output-all/cn)
 '(symbol-downward x)
 (do-section
 (mtypes-to-template 'fl output-all/cn)
 '(symbol-floating x)
(do-section (mtypes-to-template 'or output-all/cn) '(flatten x)
 (do-section
 (mtypes-to-template 'or output-all/cn)
 '(car (symbol-ornamentation 2 3 0.52 20 x '(1/8)))
 cn-sym)))))))))))))

(setq cn-len-o
(do-section (mtypes-to-template 'xl output-all/cn)
'(change-length :divide 2 x :ratio)
(do-section (mtypes-to-template 'cl output-all/cn)
'(change-length :times 2 x :ratio)
(do-section (mtypes-to-template 'lr output-all/cn) ; length-
repeat
'(length-repeat 2 x)
 cn-len))))
```

```
;; score-template


(def-tonality
w1 (activate-tonality (chromatic f 6))
w2 (activate-tonality (chromatic f 4))
br (activate-tonality (chromatic f 3))
s1 (activate-tonality (chromatic f 5))
s2 (activate-tonality (chromatic f 3))
pc (activate-tonality (chromatic f 5))
cn (activate-tonality (chromatic f 4))
)


(def-symbol
w1  w1-sym-o
w2  w2-sym-o
br  br-sym-o
s1  s1-sym-o
s2  s2-sym-o
pc  pc-sym-o
cn  (do-section :all '(make-octave 12 x) cn-sym-o)
)

(def-length
w1 w1-len-o
w2 w2-len-o
br br-len-o
s1 s1-len-o
s2 s2-len-o
pc pc-len-o
cn cn-len-o
)


(def-velocity
w1 '(64)
w2 '(64)
br '(75)
s1 '(76)
s2 '(76)
pc '(64)
cn '(70)
)
```

```
(def-zone
default (zone-ratio-sc cn-len-o)
;(3/4 5/4 1/2 33/8 11/4 4/1 17/8 23/8 3/4 1/2 21/8 1/4 5/8 1/4 19/8 4/1 1/4 3/4 5/4
;1/2 1/2 9/4 7/8 5/8 3/8 11/8 11/4 1/2 5/16 1/1 17/8 1/1 19/8 5/4 15/8 1/2 1/1 11/8
; 13/8 3/4 1/2 5/4 1/2 5/8 5/8 1/4 3/4 11/8)
)

(def-channel
w1  1 ; flute
w2  2 ; bassoon
br  3 ; trombone
s1  4
s2  5
pc  6
cn  7
)

#|
(def-program gm-sound-set
w1 flute
w2 bassoon
br trombone
s1 string-ensemble-1
s2 string-ensemble-1
pc vibraphone
cn electric-piano-1
)
|#

(def-controller gm-controllers
  (w1  panning  '((60)))
  (w2   panning  '((70)))
  (br   panning  '((80)))
  (s1   panning  '((20)))
  (s2   panning  '((110)))
  (pc   panning  '((120)))
  (cn   panning  '((0)))
)
```

This list of zones is essentially the time-signature structure for the whole movement. Each zone correponds to the length of a symbol-phrase from the master list (see page 15)

```
(def-tempo 90)

(compile-instrument-p "ccl;output:"  "concerto-1b"
w1
w2
br
s1
s2
pc
cn
)
```

```
;; Concerto for Instruments 1bi (Instrumentarium) 19.11.03
;; movement 1 section 2

;; tonality of final chord of section 1

(setq chord-m '(c d j k m)
      chord-mx '(-cdcj ))

(setq scale-1 (pattern-to-scale chord-m))
; (d 4 d# 4 a 4 a# 4 c 5)

(create-tonality resonance-1 '(d 4 d# 4 a 4 a# 4 c 5))

(activate-tonality (resonance-1 g 4))
;((g 4 g# 4 d 5 d# 5 f 5))


(setq c-seq (g-chord 0.1 4 4 -5 7 (gen-repeat 11 chord-m))
; (-d-cef j-bag fh-d-c lmoe ekln hiop mcdj . . .
      m-seq (do-section :all '(symbol-melodize x) (mapcar 'list (append chord-mx c-seq)))
; ((-c d c j) (-d -c e f) (j -b a g) (f h -d -c) . . .
)
```

This is the Coda of Movement 1. It is organised quite differently from any other movement in the Concerto sequence. It takes the zone structure of the last twelve phrases of the first section and repeats it. Symbol-phrases are generated as before BUT these are set against a string of tonalities generated (using <g-chord>) from the final chord of section 1. This technique first appears in the second movement of my Conversations in Colour (2001)

```
(setq t-seq (transpose-chords (patterns-to-scales m-seq) 10))

#|
((g# 4 c 5 c# 5 g 5) (g 4 g# 4 d 5 d# 5) (a 4 a# 4 e 5 g 5) (g 4 g# 4 d# 5 f 5)
(d 5 a 5 a# 5 c 6) (d 5 g# 5 a 5 b 5) (f 5 f# 5 c 6 c# 6) (c 5 c# 5 g 5 a# 5)
(c 5 c# 5 g# 5 a# 5) (a 4 e 5 f 5 g 5) (f# 5 c 6 c# 6 d# 6) (f# 5 g 5 c# 6 d 6)
(g 5 g# 5 d 6 f 6) (f 5 f# 5 c# 6 d# 6) (g 5 g# 5 a# 5))
|#

(setq zone-1 (reverse '(11/8 13/8 3/4 1/2 5/4 1/2 5/8 5/8 1/4 3/4 11/8 2/1)))
; zones derived from the last part of section 1

(setq val (make-symbol-value zone-1 '1/16)
; 186
      val-x (do-section :all '(make-symbol-value x '1/16)  (mapcar 'list zone-1)))
; (32 22 12 4 10 10 8 20 8 12 26 22)

(setq source  (gen-noise-white 768 1.0 0.2)
      source-x (subseq source 512 (+ 512 val)))

(setq sym (vector-to-symbol a m source-x)
      d-sym (symbol-divide val-x nil nil sym))

#|
((i l i f i b h j m i j k g i g b e l f l h h k k h f h f g m h d)
 (f b l g l k d h a g c g h k h e f h i c h m) (i k l k a l f h j e k j) (i g l c)
(b a f b f l h f a b) (k j g l l l g e k k) (a g d i m e j d)
(h g h j c f h j i k g h k i l l m e g a) (j k l c f b a j) (j l j k f e d b b l e a)
(i m j j h k k d k f e a f k j l e a e j c g c c k i)
(c f f i m k i h g l f l j i b m f f f j g l))
|#


(setq d-sym-scale (do-section :all '(symbol-scale '(-d d) x) d-sym)


#|
((b c b -b b -d a b d b b c a b a -d -b c -b c a a c c a -b a -b a d a -c)
(a -c c a c c -c a -d a -c a a c a -b a a b -c a d) (b c d c -d d a b c -b c c) (b a d -d)
(-c -d a -c a d b a -d -c) (c b -b d d d -b -d c c) (-d a -c b d -b c -c) . . .
|#
       e-sym-scale (do-section :all '(symbol-scale '(a g) x) d-sym))
```

```
;; using skipping and cutting

(setq t-1  '(= x x = = = = x = = x =)
      t-2 (template-invert t-1))

(setq e-sym-w (do-section t-2
                          '(symbol-cut-i 5 9 x 0.121795)
                          (do-section t-1
                                '(symbol-skip-i 7 12 x 0.12 ) e-sym-scale)))
#|
((= = = = = = = = e f e c d g e e d e d a c f c d f d c d c d d b)
(= = g d = = = e = d b d = = e c = e = = e g) (e f g f a g d e f c f f) (e d g a)
(= = = = = = = b g a) (= = = = = c g g a f) (a d c e g c e c)
(e = e = = = e = e f d e = = g = = c d a) (f f g b d b a f) (= = = = = = = g c c g c)
(= g = = e = = = = c = a c f = g = a c = b d b = f e)
(= = = = = c e g f e d f c f e e g c c e d f))
|#

(setq d-sym-x (do-section t-2 '(symbol-cut-i 5 9 x 0.127) (do-section t-1
                               '(symbol-skip-i 5 9 x 0.126 ) d-sym-scale)))


; zones for reference  (2/1 11/8 3/4 1/4 5/8 5/8 1/2 5/4 1/2 3/4 13/8 11/8)

(setq s1-len '((-1/1 1/1) (-3/8 3/8 5/8)(1/4) (1/16) (1/8) (1/8) (1/8) (1/2. 1/2) (-1/4 1/4)
               (1/8) (4/8 -4/8 1/8 1/8 1/8 1/8 1/8) (5/8 6/8)))

(setq db-len '((-1/1 -1/4 1/8 1/8 1/4 1/4) (-3/8 1/4. 1/4. 2/8) (-1/4 1/8 1/8 1/8 1/8) (1/4)
              (5/8) (4/8 1/8) (1/8) (-1/4 1/4 1/4 1/4 1/4) (-1/4 1/4)(-1/4 1/8 1/8 1/8 1/8)
              (4/8 -6/8  1/8 1/8 1/8) (7/8 -1/8 1/8 1/8 1/8)))

(setq cn-len (do-section :all '(length-condense x) db-len))
; ((-5/4 3/4) (-3/8 1/1) (-1/4 1/2) (1/4) (5/8) (5/8) (1/8) . . .


(setq dt-1 '(x = x = = x x = = = = x)
      dt-2 (template-invert dt-1))

(setq db-sym (do-section '(= = = = = = = = = x x x) '(symbol-transpose -12 x)
                 (do-section dt-1 '(sort> x) (do-section dt-2 '(sort< x)  m-seq))))
; ((j d c -c) (-d -c e f) (j g a -b) (-d -c f h) (e l m o) . . .
```

```
;; score-template

(def-tonality
s1 t-seq
s2 t-seq
ob t-seq
tr t-seq
db (activate-tonality (chromatic f 3))
cn  (activate-tonality (chromatic f 4))
)


(def-symbol
s1 '(a b c d e f g)
s2 '(-d -c -b a b c d)
ob e-sym-w
tr d-sym-x
db db-sym
cn (mapcar 'list (append chord-mx c-seq))
)

(def-length
s1 s1-len
s2 s1-len
ob '(1/16)
tr '(1/16)
db db-len
cn  cn-len
)


(def-velocity
s1 '65
s2 '65
ob '70
tr '90
cn '64
db '70
)


(def-zone
default zone-1
; (2/1 11/8 3/4 1/4 5/8 5/8 1/2 5/4 1/2 3/4 13/8 11/8)
)
```

```
(def-channel
s1 4
s2 5
ob 12
tr 14
cn 7
db 16
)

#|
(def-program gm-sound-set
ob oboe
tr trumpet
tr trumpet
cn  electric-piano-1
)
|#

(def-tempo 90)

(compile-instrument-p "ccl;output:" "coda"
s1
s2
ob
tr
cn
db
)
```