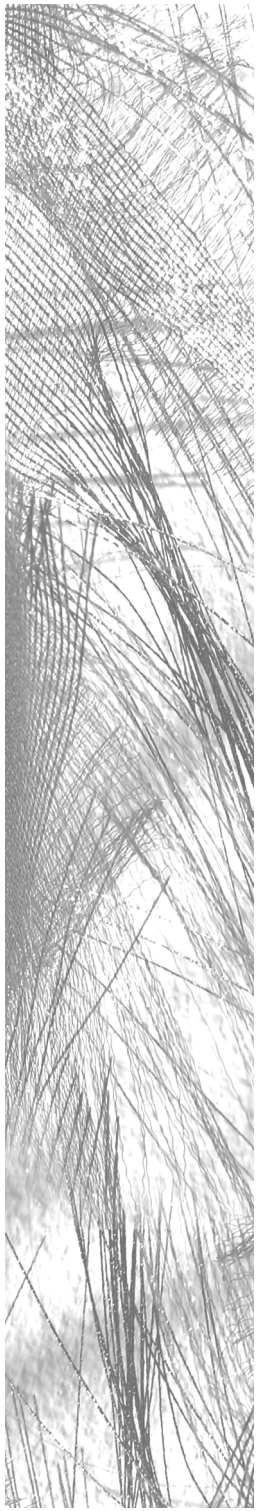


After Hindemith

For Piano Trio

Symbolic Composer score files, annotated by Phil Legard

Nigel Morgan

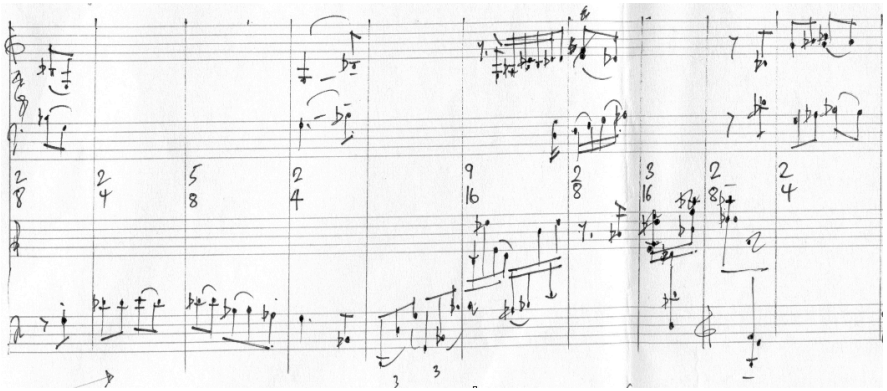
After Hindemith

Paul Hindemith was not only an extraordinarily prolific composer but an important theorist and teacher. In Eckart Richter's essay '*A Glimpse into the Workshop of Paul Hindemith'* there is a description of Hindemith's working method as presented to a class at Yale in 1951. Hindemith later 'converted' this procedure into a guide for listening in a lecture he gave at the University of Zurich in 1955. This 'working method' involved a 4-stage process:

> 1. The general determination of the character, medium and the basic purpose of the piece, as well as its expressive character, and even place of performance.
>
> 2. A master plan of formal design, including the overall shape, the number and character of sections, changes in mode and tempo, rhythmic character, texture and the degree of activity, the gauge being the amount of effort the listener must expend to comprehend.
>
> 3. Then 'came the tonal layout in which the basic tonalities of each section and their relative degrees of tonal stability and complexity, as well as the modulations, were mapped by means of a diagram.'
>
> 4. Specific thematic material.

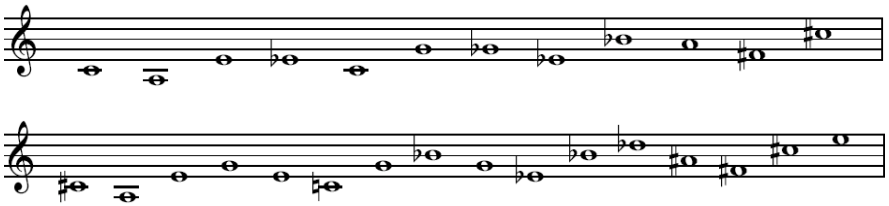
My piano trio adopts Hindemith's 4-stage process, but with a contemporary twist: the integration of algorithmic techniques with traditional methods of composition. The music is a sequence of playful sections (algorithmically composed) with more meditative episodes and interludes (freely composed on paper).

Corresponding to the first stage of Hindemith's process an initial 'imagined' statement was presented in the first twelve bars, handwritten by the composer.



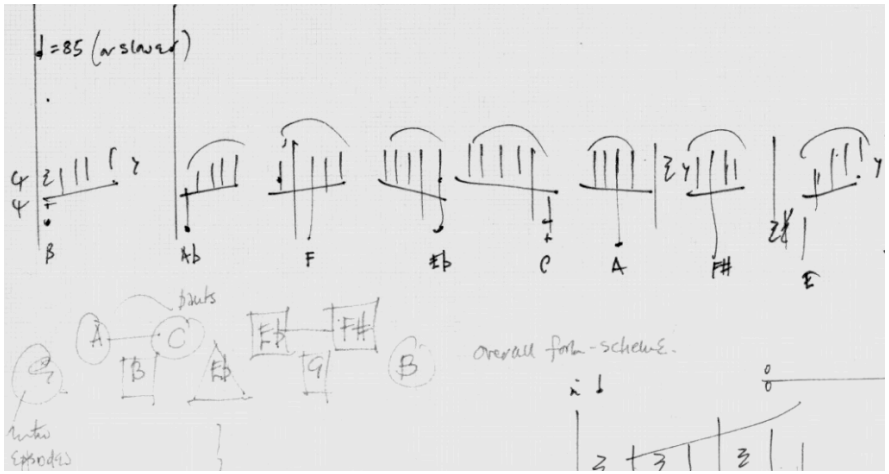
Original sketch of the opening statement.

A number of tonality patterns were devised and chosen from a 3-note figure: a combination of 4 ascending minor triads (in arpeggio), 4 ascending dominant seventh chords (in arpeggio), and two positions of the octotonic scale acting as a binding mechanism.



The two tonality patterns, referred to as min-seq1 and maj-seq1 in the code.

The whole rhythmic and instrumental design was then drafted on graph paper followed by a tonal layout.



Drafts of rhythmic and textural design.

Title	Reference	Tonality	Tempo	Composition
Preface	(1-12)	G	96	Handwritten
Section I	A (13-51)	A	96	SCOM
Episode I	B (52-69)	B	84	Handwritten
Section II	C (70-108)	C	84	SCOM
Episode II	D (109-123)	Eb	72-60-72	Handwritten
Section III	E (124-173)	Eb	84	SCOM
Episode III	F (174-177) G (178-194)	G	60-72-84	Handwritten
Section IV	H (195-249)	F#	96	SCOM
Coda	I (249-254)	B	96	Handwritten

The final 'master plan', showing the sections of the piece, their compositional method and tonal scheme.

Then, finally, specific thematic material was created. The means of creating this material varied according to the method of composition.

Regarding the sections composed on paper: Episode I is a bell-like passage for solo piano in which falling bass notes (in minor thirds and major seconds) set off a five note arpeggio-like figure. The phrase grouping is in five. The harmonic rhythm is unambiguous: from bar to bar, moving with intent towards the next harmonic area.

Episode II begins with three chords that accumulate notes from a minor triad to produce an extended chord, building harmonic tension and reaching a period of stasis. The episode is in two parts with a central pivot point around which the whole movement rotates.

Episode III is a variant on Episode I but opens with an echo of the chordal stasis of Episode II. In this variant the material is moved onto violin and cello with harp-like figurations from the piano surrounding the arpeggio figures. This time the bass notes are reversed to rise rather than fall.

The Coda shares similar material with the preface, but reverses melodic direction downwards.

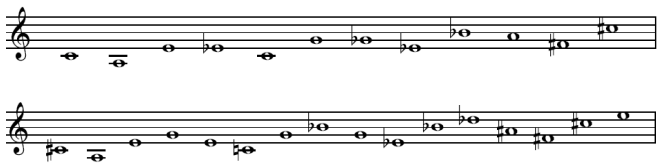
There now follows an annotation of the Symbolic Composer code that generated the music for Sections I and III. Here the tonality of each algorithmically generated section corresponds to the tonic of each of the four arpeggiated triads in min-seq1. In each section a triad is focused upon, with occasional interventions from the other values in the tonality pattern. This thematic material is then subjected to further processing in each voice, as well as distortions and transformations caused by subjecting it to tonal mappings outside the min-seq1 tonality pattern.

The code for Sections II and IV has been omitted. Section II follows the same program structure as Section I, but concentrates on the data symbols (d e f), as opposed to (a b c). Similarly, Section IV draws on (j k l) as opposed to (g h i).

```
;;; Piano Trio (section I)
;; functions

;; material

(create-tonality min-seq1 '(c 5 a 4 e 5 e& 5 c 5 g 5
;                           a    b    c   d   e   f
                            g& 5 e& 5 b& 5 a 5 g& 5 d& 6))
;                           g    h    i    j    k   l


(create-tonality maj-seq1 '(c# 5 a 4 e 5 g 5
;                            a   b    c   d
                            e 5 c 5 g 5 b& 5
;                           e   f   g    h
                            g 5 e& 5 b& 5 d& 6
;                            i   j    k    l
                            b& 5 g& 5 d& 6 f& 6))
;                            m    n   o    p
```

The material generated by this code comprises sections A1-A3 (bars 13-51) of *After Hindemith*.

Throughout the four algorithmically composed sections of *After Hindemith* these tonal patterns are employed. Note that they are not simply ascending scalic sequences, but pre-composed structures based on ascending transpositions of an arpeggiated minor triad (in min-seq1) and ascending dominant seventh chords (in maj-seq1). Here are the two core tonalities expressed in notation:



```
(setq var-sym1
      (gen-collect 0.14 20 :list
       '(symbol-shuffle
         (append (pick-random (list '(a b c) '(a b) '(c b) '(a c)))
           (gen-random nil (get-random 1 3) '(d e f g h i j k l))) nil)))
```

Each section concentrates on a particular area of the tonality sequences for its thematic material. In this section (a b c) is the focus. 20 groups of symbols are created by first picking one of the lists following possibilities (a b c) (a b) (c b) or (a c). To this are appended 1 to 3 notes picked from the rest of tonality pattern. Each group is then shuffled, resulting in groups that will always have elements of the group (a b c), along with interventions from elsewhere in the tonal continuum, for example:
`(l` **`a b`** `f k) (i` **`a`** `i` **`b c`**`) (`**`c`** `f` **`b a`**`)`
- a bold symbol indicates the core triad.

```
(setq zones (make-zone-list var-sym1 (build-list '(1/16) (length var-sym1))))
```

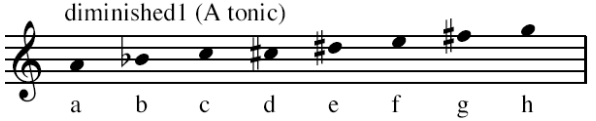
This creates zones of an appropriate duration to allow 1/16 note for each item in the symbolic groups generated above.

```
(setq mask (gen-template 0.12 1 2 (length var-sym1)))
(setq mask1 (gen-template 0.13 1 2 (length var-sym1)))
```

Create two templates for processing each group in different ways.

```
;; score

(def-tonality
      violin (append (gen-repeat 10 (activate-tonality (min-seq1 c 6)
                                                       (diminished1 a 5)))
                     (gen-repeat 10 (activate-tonality (maj-seq1 c# 6)
                                                       (diminished1 a 5))))
      cello (append (but-last (gen-loop '((1 3 6) (1 2 1))
                                (activate-tonality
                                 (min-seq1 c 4)(diminished1 a 4) (diminished1 a 3))))
                    (but-last (gen-loop '((1 3 6) (1 2 1))
                                (activate-tonality
                                 (maj-seq1 c# 4)(diminished1 a 4) (diminished1 a 3)))))
      piano-r (append
                 (gen-repeat 10 (activate-tonality (min-seq1 c 6) (diminished1 a 5)))
                  (gen-repeat 10 (activate-tonality (maj-seq1 c# 6) (diminished1 a 5))))
      piano-l (append (but-last (gen-loop '((1 3 6) (1 2 1))
                                  (activate-tonality
                                  (min-seq1 c 3)(diminished1 a 4) (diminished1 a 3))))
                      (but-last (gen-loop '((1 3 6) (1 2 1))
                                  (activate-tonality
                                 (maj-seq1 c# 3)(diminished1 a 4) (diminished1 a 3)))))
)
```

There is, in addition to min-seq1 and maj-seq1, a third tonality – diminished1 – the tonic of which changes in each of the four algorithmically composed sections according to the predetermined scheme tonic of the triad being focussed on in the section: A C Eb F#.



There are two forms of relationship between the instruments and the three tonalities in Sections I and II:

1) The violin and piano-rh move through a tonal scheme made up of 10 repetitions of (min-seq1 c) (dimished1 a), followed by 10 repetitions of (maj-sec1 c#) (diminished a) – at total of 40 tonal zones.

2) The tonal scheme for the cello and piano-lh is created using `gen-loop` to generate repeats of a series of 3 tonalities for example: (min-seq1 c 4) (diminished1 a 4) (diminished1 a 3). The three elements of the series are repeated 6 times (1 3 6) and then the first two elements repeated once (1 2 1). However, since this does not result in the function terminating at the end of the series, the final element is appended anyway (since the function creates complete loops of material). Therefore the `but-last` function is used to trim the final element, resulting in a total of 40 tonality zones.

Since the piece is palindromic in nature the divisions between min-seq1 and maj-seq1 mark a half-way 'pivot' point in the music.

```
(def-symbol
      violin (do-section '(21 :end) '(symbol-compress -35 x)
                  (do-section '(38 :end) '(reverse x)
                  (gen-palindrome
                   (do-section mask1 '(append (build-list '= (length x)) x)
                  (do-section  mask '(filter-delete '(b) x) var-sym1)))))
      cello (do-section '(25 :end) '(symbol-scale '(-e l) x)
                 (do-section '(38 :end) '(reverse x)
                 (gen-palindrome
                  (do-section mask1 '(append x (build-list '= (length x)))
                  (do-section  mask '(filter-delete '(a c) x) var-sym1)))))
      piano-r (do-section '(21 :end) '(symbol-chordize 0.12 nil -3 nil x)
                   (do-section '(21 :end) '(symbol-scale '(a p) x)
                   (do-section '(38 :end) '(reverse x)
                   (gen-palindrome
                    (do-section  mask '(filter-delete '(e h k) x) var-sym1)))))
      piano-l  (do-section '(25 :end) '(symbol-scale '(-e l) x)
                    (do-section '(38 :end) '(reverse x)
                   (gen-palindrome
                    (do-section  mask
                                 '(filter-delete '(d f g i j l) x) var-sym1))))
)

(def-length
violin '(1/16)
cello '(1/16)
piano-r (gen-palindrome (gen-random 0.129 (length var-sym1)
                  '((-1/16 1/16 1/16 1/16) (-1/8 1/8) (1/16) (1/8))))
piano-l (gen-palindrome (gen-random 0.139 (length var-sym1)
                  '((-1/16 1/16 1/16 1/16) (-1/8 1/8)(1/16) (1/8))))
)
```

Finally, the symbolic and length data is defined. In this section the material that defines the theme of the section (var-sym1) is subjected to different types of algorithmic processing for each instrumental voice.

For example, in the violin the material has the symbol ‘(b) filtered out according to the templates generated earlier. The second template is also used to create rest symbols at the beginning of selected symbol lists, which are of an equal length to the number of symbols contained therein. For example: (a e f f c) becomes (= = = = = a e f f c), creating unique phrase boundaries and pauses.

Certain symbol lists are also reversed and compressed and (in the piano) chordized, adding further shape to the material. Note too that the material is also subjected to `gen-palindrome`, creating a mirror image of the symbol lists - additionally, that the content of the lists is not reversed, just their order so that: (a b c) (d e f) (g a) would become (a b c) (d e f) (g a) (d e f) (a b c).

The use of `symbol-compress` and `symbol-scale` in the latter half of the piece relates to the difference in the length of tonalities min-seq1 and maj-seq1. Since maj-seq1 is based on 4 note sequences the range will be a lot smaller when symbols are mapped to it. This strategy helps avoid the music getting lost in the lower reaches of tonalities that contain a greater number of small intervals.

The length of notes is a constant 1/16 for the violin and cello. For the piano the lengths are selected 20 times from a group of four possibilities. These are then turned into palindromic form.

```
(def-duration
      piano-r '(1/16)
      piano-l '(1/16)
)
```

Although the length of the notes was determined above, the actual sounding durations of the notes in the piano part are a constant 1/16. This means that when a 1/8 note occurs it will be interpreted as a staccato note.

```
(def-zone
      violin (gen-palindrome (change-length :times 2 zones))
      cello (gen-palindrome (change-length :times 2 zones))
      piano-r (gen-palindrome
         (change-length :times 2 (length-syncopate 0.121 3 1 zones)))
      piano-l (gen-palindrome
         (change-length :times 2 (length-syncopate 0.121 2 1 zones)))
)
```

Zone-lengths for the cello and violin are multiplied. This allows for repetitions of length and symbolic material to occur in those zones where material has not been preceded by silences. Therefore the symbolic data (a b c) (= = = d e f) would be heard as (a b c a b c) (= = = d e f). This also happens in the piano part, with the addition of `length-syncopate`, which has the effect of creating zones of silence.

```
(def-velocity
      violin  (do-section :all '(gen-accent (get-random 15 30) (length x) x)
                          (do-section :all '(build-list '64 (length x))
                  (get-symbols-of 'violin)))
      cello (get-velocities-of 'violin)
      piano-r (do-section :all '(gen-accent (get-random 15 30) (length x) x)
                          (do-section :all '(build-list '72 (length x))
                                      (get-symbols-of 'violin)))
      piano-l (do-section :all '(gen-accent (get-random 15 30) (length x) x)
                          (do-section :all '(build-list '80 (length x))
                                      (get-symbols-of 'violin)))
)

(def-channel
      violin  1
      cello   2
      piano-r 3
      piano-l 3
)
```

For each voice a velocity list is created corresponding to the length of the symbolic material. For example, the violin has a base velocity of 64. `Gen-accent` is used to create slight accents (in the range of 15-30, additional to the base velocity).

```
(def-program gm-sound-set
      violin violin
      cello cello
      piano-r acoustic-grand-piano
      piano-l acoustic-grand-piano
)

(def-tempo 96)

(compile-instrument-p "ccl;output:" "trio ii"
      violin
      cello
      piano-r
      piano-l
)
```

```
;;; Piano Trio (part 6 - section III)
;; functions

(defun make-rhy-tie (tic notes)
  (setf note-list (list (car notes))
        rhy-list (list tic))
  (do ((step 1 (+ step 1))) ((equal step (length notes)) note-list rhy-list)
      (cond ((not (equal (nth step notes) (nth (- step 1) notes)))
             (push (nth step notes) note-list)
             (push tic rhy-list))
             ((equal (nth step notes) (nth (- step 1) notes))
             (setf rhy-list
             (p-replace nil 'first (change-length :add tic (list (first rhy-list)) :ratio)
              rhy-list)))))
  (setf note-list (reverse note-list)
        rhy-list (reverse rhy-list)))


(defun find-repeats (symbol-list)
"finds and removes any repeated symbols from a list:
use with <make-rhy-tie> and <tie-repeat>"
   (prog (out el1 el2)
     loop
     (cond ((null symbol-list) (return out)))
     (setq el1 (car symbol-list))
     (setq el2 (cadr symbol-list))
     (setq out (append out (cond ((eql el1 el2)
                                  nil)
                                  (t (list el1)))))
     (setq symbol-list (cdr symbol-list))
     (go loop)))
```

This function constructs a rhythmic figure from a set of symbols. A basic rhythmic value is provided (e.g. 1/8) and for each duplication of a symbol that value is incremented. So: (a a a) (b f f) (b i b f) would result in the rhythm (3/8) (1/8 1/4) (1/8 1/8 1/8 1/8).

This function removes duplications of symbols so that only the first instance remains. The previous example (a a a) (b f f) (b i b f) becomes (a) (b f) (b i b f).

```
;; material

(create-tonality min-seq1 '(c 5 a 4 e 5 e& 5 c 5 g 5
;                           a    b    c   d   e   f
                            g& 5 e& 5 b& 5 a 5 g& 5 d& 6))
;                           g    h    i    j    k   l


(create-tonality maj-seq1 '(c# 5 a 4 e 5 g 5
;                             a   b    c   d
                            e 5 c 5 g 5 b& 5
;                           e   f   g    h
                            g 5 e& 5 b& 5 d& 6
;                            i   j    k    l
                            b& 5 g& 5 d& 6 f& 6))
;                            m    n   o     p


(setq var-sym1
(gen-collect 0.1412 5 :list
 '(symbol-shuffle
   (append (pick-random (list '(g h i) '(g h) '(i h) '(g i))); e& minor arpeggio
           (gen-random nil (get-random 1 4) '(a b c d e f j k l))) nil)))


(setq vc-sym
      (reverse (apply 'append  (gen-evolve 9
                               '(cf-noise-white (get-random 2 6) 1.0 nil x)
                                                var-sym1 :list 0.12))))


(setq vn-sym
      (reverse (apply 'append  (gen-evolve 9
                              '(cf-noise-white (get-random 2 6) 1.0 nil x)
                                                var-sym1 :list 0.13))))
```

The material that defines the character of this section is generated in a similar manner to that of sections I and II. The difference being that there are fewer groups (5, as opposed to 20), but each group has more interventions from outside the core triad (g h i).

For each instrumental voice `gen-evolve` is called to create a kind of feedback loop. At each step `cf-noise-white` is called to create a white-noise melody from elements in the list, between 2-6 symbols in length. The resulting list is then fed back into the function, the outcome being that eventually all but one element will be discarded. For example, the first element of the list would evolve as follows:

```
(b h i g d c) >>> (g c b) >>> (b g) >>>
(g g b b g g) >>> (g g g) >>> (g g g) >>> (g g) >>>
(g g g) >>> (g g g g g) >>> (g g g g g g)
```

This sequence is then reversed, giving the impression of a melody gradually expanding from a set of extended tones.

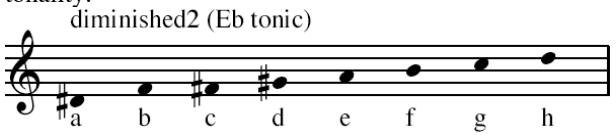
```
(setq pnr-sym
      (apply 'append  (gen-evolve 9
                                  '(cf-noise-white (get-random 2 6) 9.0 nil x)
                                  var-sym1 :list 0.13)))


(setq pnl-sym
      (apply 'append  (gen-evolve 9
                      '(cf-noise-white (get-random 2 6) 9.0 nil x)
                                       var-sym1 :list 0.12)))


(setq mask (gen-template 0.123 1 2 (length pnr-sym)))
(setq mask1 (fill-rest mask 'x))


(setq maskx (gen-template 0.124 1 2 (length vc-sym)))
(setq maskx1 (fill-rest maskx 'x))

(setq len-val (mapcar 'length (do-section :all '(find-repeats x) pnr-sym)))


(setq vn-programs (flatten (do-section  mask1 '(replace '(46) x
 :start1 (length x) :end1 (length x)
 :start2 (length x) :end2 (length x)) (gen-repeat 50 '((41))))))
```

Store the number of non-duplicate notes in each symbol-list of the piano right-hand. For example:
`((b h i g d c) (g a a h f l i) (h i k k j)) = (6 6 4)`

For each zone the violin may change its program (midi instrument) number, where 46 is the midi program number for *arco*, and 41 for *pizz*.

```
;; score

(def-tonality
      violin (append (gen-repeat 25 (activate-tonality (maj-seq1 c# 6)))
                     (gen-repeat 25 (activate-tonality (min-seq1 c 6))))
      cello  (append (gen-repeat 25 (activate-tonality (maj-seq1 c# 4)))
                     (gen-repeat 24 (activate-tonality (min-seq1 c 4)))
                     (activate-tonality (diminished2 e& 4)))
      piano-r (append (gen-random 0.1 25 (activate-tonality
                                          (diminished1 e& 5) (maj-seq1 c# 5)))
                      (gen-random 0.1 25 (activate-tonality
                                          (min-seq1 c 5) (diminished2 e& 5))))
      piano-l (append (gen-random 0.12 25 (activate-tonality
                                           (diminished1 e& 3) (maj-seq1 c# 3)))
                      (gen-random 0.12 25 (activate-tonality
                                           (min-seq1 c 3) (diminished2 e& 3))))
)
```

Define tonality structures for each voice. The violin is composed of 25 repetitions of (maj-seq1 c#), followed by 25 of (min-seq1 c). The cello is similar, but diverges in the last zone to (diminished2 Eb).
The both hands of the piano make use of random combinations, firstly of (diminished1 Eb) and (maj-seq1 c#), followed by combinations of (diminished2 Eb) and (min-seq1 c). Note that this section introduces a new tonality:




```
(def-symbol
      violin (do-section :all '(find-repeats x) vn-sym)
      cello (do-section maskx '(replace  '(g h i c a b d e f j k l) x
                           :start1 (length x) :end1 (length x)
                            :start2 (length x) :end2 (length x))
                     (do-section :all '(find-repeats x) vc-sym))
      piano-r (do-section mask '(replace '(g h i c a b d e f j k l) x
                           :start1 (length x) :end1 (length x)
                           :start2 (length x) :end2 (length x))
                    (symbol-divide len-val nil nil
                                   (symbol-chordize-shift 50 nil nil
                              (flatten
                          (do-section :all '(find-repeats x) pnr-sym)))))
      piano-l (do-section :all '(find-repeats x) pnl-sym)
)
```

Generate symbolic data for each voice. The violin voice is processed by `find-repeats` to reduce instances of repeated symbols. The cello likewise, but also with certain groups being completely replaced by the sequence (g h i c a b d e f j k l), according to the template maskx.

The piano right-hand operates in a similar way to the cello, although a different template controls the substitution. `Symbol-chordize` is also employed to create new chords from the material. Because this function needs to work on a flattened list, the structure is then restored with len-val.

```
(def-length
      violin (do-section :all '(make-rhy-tie '1/8 x) vn-sym)
      cello (do-section maskx '(make-rhy-tie '1/16 x)
                  (do-section maskx1 '(make-rhy-tie '1/8 x) vc-sym))
      piano-r (do-section mask '(make-rhy-tie '1/16 x)
                  (do-section mask1 '(make-rhy-tie '1/8 x) pnr-sym))
      piano-l (do-section :all '(make-rhy-tie '1/8 x) pnl-sym)
)
```

Use `make-rhy-tie` to create differing note values from the distribution of duplicated symbols. The cello and piano right-hand also make use of occasional 1/16 based passages, allowing for the generation of quick, repeated phrases – for example, see the cello in bars 161-2.

```
(def-duration
      violin (do-section mask1 '(change-length :divide 2 x) (get-lengths-of 'violin))
      cello  (do-section mask '(change-length :divide 2 x) (get-lengths-of 'cello))
      piano-l (do-section maskx1 '(change-length :divide 2 x) (get-lengths-of 'piano-l))
)

(def-zone
      violin (do-section :all '(make-zone x) (get-lengths-of 'violin))
      cello (get-controls-of 'violin)
      piano-r (get-controls-of 'violin)
      piano-l (get-controls-of 'violin)
)
```

Create staccato passages by dividing the duration of notes by 2 in selected sections.

```
(def-velocity
      violin  (do-section mask '(gen-cresc 44 72 (length x))
                  (do-section :all '(gen-accent (get-random 5 15)
                                          (length x) x)
                        (do-section :all '(build-list '30 (length x))
                  (get-symbols-of 'violin))))
      cello  (do-section maskx '(gen-cresc 44 80 (length x))
                  (do-section :all
                              '(gen-accent (get-random 10 20) (length x) x)
                        (do-section :all '(build-list '30 (length x))
                  (get-symbols-of 'cello))))
      piano-r (do-section mask '(gen-cresc 44 96 (length x))
                  (do-section :all
                              '(gen-accent (get-random 5 15) (length x) x)
```

The previously-used templates are also employed to create areas of crescendo, for example where maskx is used in the cello to create rapid phrases based on 1/16 notes. These same areas are now also processed with `gen-cresc`, to create crescendos from 44-80.

```
                        (do-section :all '(build-list '44 (length x))
                    (get-symbols-of 'piano-r))))
      piano-l (do-section :all '(gen-accent (get-random 5 15) (length x) x)
                        (do-section :all '(build-list '40 (length x))
                    (get-symbols-of 'piano-l)))
)

(def-channel
      violin  1
      cello   2
      piano-r 3
      piano-l 3
)


(def-program gm-sound-set
      violin (absolutes vn-programs)
      cello cello
      piano-r acoustic-grand-piano
      piano-l acoustic-grand-piano
)

(def-tempo 84)

(compile-instrument-p "ccl;output:" "trio vi"
      violin
      cello
      piano-r
      piano-l
)
```