Technical Report No. 7
National Science Foundation Project No. GK-14191, May, 1973

MUSICOL MANUAL, VERSION 1

(MUSical Instruction Composition Oriented Language)
for the 6400 Digital Computer

by Peter Gena

State University of New York, Buffalo, N.Y.

Abstract: This is a manual for MUSICOL, a computer language for composing music. The composer need not have any knowledge of computer programming to use it, nor must he concern himself with organizing various amounts of data-cards with intricate, precisely placed, nemerical values which serve as input to data-processing routines. MUSICOL uses conventional musical mnemonics in a free card-format. When and how instruments, controls, chance processes, etc., act can be specified at will. With freedom to program his own musical ideas, the composer's personal style emerges in the resultant composition.

TABLE OF CONTENTS

I. MUSICOL OVERVIEW

1.1 General Description

MUSICOL (MUSical Instruction Composition Oriented Language) is a language for composition using musical mnemonics. It is a block-structured language with a free format, constructed in such a way that a great degree of flexibility is available to the composer as he sets desired musical parameters in successive time-blocks. This allows total control over the composition by either the composer or the computer, or any degree of control in between these two extremes.

The selection of musical parameters (pitch, duration, range, attack, timbre, and dynamics) is generated by setting up probability distributions, or by Zipf's law, a stochastic principle originally developed to express the frequency of occurrence of syntactical elements in languages in terms of their rank orders. Each successive rank order is selected at a probability equal to the inverse of the rank order, unless a zeroth-ordered stochastic process is specified, in which case total randomness is generated.

Programming a composition in MUSICOL involves the logical implementation of continuous sections where the rank orders are used or not used in such a way as to create the desired parametric content. If no rank order is specified (zeroth-order), the elements are selected randomly by a random integer generator. If the composer decides to select some parameters exactly, without references to probability distributions, literals also exist. With such parametrical versatility, a composer should be able to implant his own personal style on a composition, eliminating the "salt and pepper" effect thus far common in computer compositions that depend heavily upon stochastic processes to the exclusion of other controls.

A total of sixteen different instruments can be used simultaneously. These are chosen from an operation-code list of common instruments, including non-pitched percussion. In addition, there are extra codes available (INSTR1, INSTR2, etc.) for use of instruments not included in the list, or others (XTRA1, XTRA2, etc.) for timbres and attacks not contained in the list. Special routines exist for generating lists of parametric elements.

1.1.2 MUSICOL Compiler

This is written mostly in FORTRAN IV. Each instruction (in free format) is assembled into a word, which is divided into five or six fields depending on the instruction type. Parametric strings (order listings) are stored in successive words containing five fields. A field occupies four octal digits of the CDC 60-bit word.

Example 1.1

(PPP PP P MP MF FT FF FFF) is represented octally (see Appendix A) by:

```
  1    2    3    4    5
/7042/7121/7122/7123/7124/
/7125/7126/7127/7130/7043/
```

The first digit (7) is masked onto each field to identify parameter storage.

For most other instructions, the word is divided into six fields of unequal length, the last always being filled with zeros.

<u>Example 1.2</u>

VIOLIN = PIZZ * 3 is represented as:

```
  1    2    3     4     5     6
/0040/44/0025/40/0003/0000/
```

Instructions are stored as they appear in their successive time-blocks for entry into the simulator. A complete error-detecting system providing diagnostics is available at the compilation stage, as is a word dump, to facilitate debugging. A compilation map is given after the MUSICOL source-code listing. It contains assembly information and an assigned mnemonic cross-reference list. (See Appendix B)

1.1.3 <u>MUSICOL Simulator/Output</u>

The simulator constructs a memory core for the program's MUSICOL source instructions, so that the execution of the compositional process is determined by the programmed specifics. The compiled code for each time-block is not removed from the simulation memory until the composing is completed for the set length, as specified in the block.

Output is stored on a file as each block is processed. The file is dumped by an outputting program which converts the octal representation to display code (see Appendix A), and orders the musical events according to their place in time (measures, beats, etc.).

An error system at the execution stages provides diagnostics for faulty program logic. If an error is detected, the compositional process aborts for the remaining time-blocks. Simulation proceeds however, at the loader level so that all loader errors can be detected on one program-run. Any special execution conditions are listed in the loader-execution map, which also provides the total execution time (excluding output). (See Appendix B)

1.2 MUSICOL Design/Memory Requirements

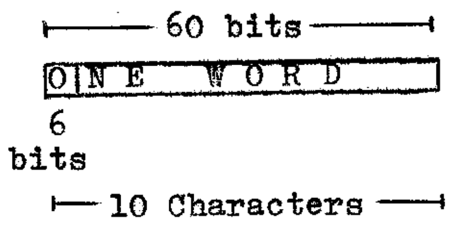
The MUSICOL language is designed to run on the CDC-6400 computer, but can be altered to run on any computer with a FORTRAN IV compiler. The source program is constructed by means of four overlays: the main-line, the compiler, the simulator/executor, and the output program. The majority of the routines are written in FORTRAN IV, excepting a number of assembly language routines which can be translated to other assembly languages or to FORTRAN IV (see Appendix E). The MUSICOL software program requires only 33,000 octal locations when loaded as binary files on the CDC.

The bit configuration of the CDC word is as follows:

1 word contains 60 bits.
10 characters are contained in each word.
Each character consists of 6 bits.

Example 1.3

A typical word:



MUSICOL operation codes contain from one to ten characters. For computers with less than ten characters per word, those op-codes that exceed the limit can be reduced in length. Special routines and the compilation instruction lengths can also be changed to fit a more convenient configuration.

II. CODING PROCEDURES

2.1 MUSICOL Character Set

All the available FORTRAN IV characters (A-Z, 0-9, etc.) can be used in MUSICOL. Programming in MUSICOL involves the use of mnemonics, expressions, and integer constants.

2.1.1 Mnemonics

A mnemonic contains from one to ten characters. The use of blanks between characters is prohibited, since they act as delimiters. For a complete list of mnemonics, see Appendix A. If a mnemonic is incorrectly spelled, a compilation diagnostic is issued.

2.1.2 Assigned Mnemonics

To specify unlisted instruments, or timbre and attack parameters, the special INSTR1,,,INSTR5; or XTRA1,,, XTRA10 op-codes **can** be replaced with mnemonics of the programmer's choice. The length of such assigned mnemonics cannot exceed ten characters. Any FORTAN IV characters except blanks and MUSICOL expressions can be used in the string.

Example 2.1

```
INSTR1 = BASSFLUTE
XTRA4 = SULTASTO
```

Assigned mnemonics cannot be replaced with already existing operation codes.

2.1.3 Expressions

The use of MUSICOL expressions denotes specific meaning to an instruction. The following is a list of expressions and their uses. A thorough explanation of contextual usages is given in Chapter 4.

-	denotes ranges
*	specifies rank orders and play/rest ratios
/	separates time signature values
(	parameter-string and voice delimiter
)	parameter-string and voice delimiter
=	sets all parametrical specifications for instruments; assigns op-codes; used for instrument deletion and replacement; used before all range instructions
,	renders all code to the right as comments
.	signifies multiple stop probabilities and literal lengths.

2.1.4 Integer Constants

Positive numbers or zero may act as operands after MUSICOL expressions.

Example 2.2

```
TIMSIG  = 3/4
REST *75
QUARTER = 60
VIOLIN(1)= ARCO * 5 = 3G -4F
CRESC = FF.232
```

2.2 Programming Format

All eighty columns of the standard "IBM" card can be used in MUSICOL. Only columns 1, and 73 - 80 have specific meanings when used.

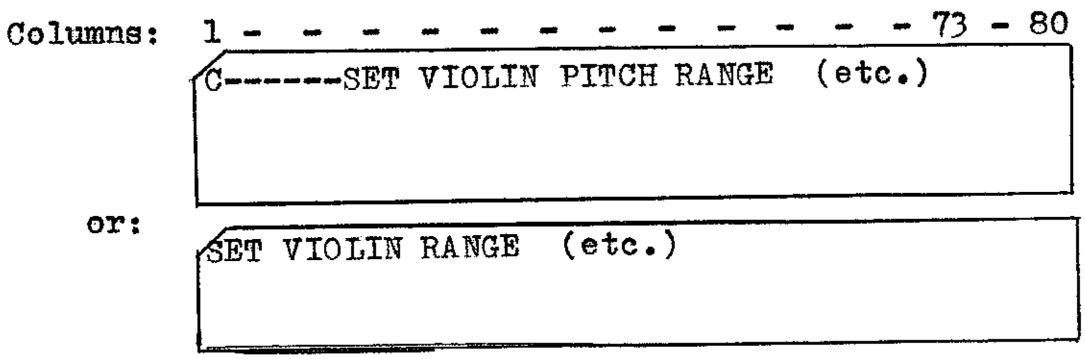
2.2.1 MUSICOL Statements

Statements (instructions) have a free format. A card may contain as many statements as desired by the programmer. An incomplete statement continues on the next card, providing no operation code is split. Expressions and blanks are recognized as delimiters (multiple blanks are ignored). MUSICOL statements are contained in columns 2 - 72.

2.2.2 Comments

Any character in the first column of a card will allow columns 1 to 72 to be treated as comments. Similarly, the use of a comma (,) in any column will render the remainder of the card free from compilation. Comments are used to outline program logic, etc.

Example 2.3



```
Columns: 1 - - - - - - - - - - - - 73 - 80
C------SET VIOLIN PITCH RANGE  (etc.)

or:
SET VIOLIN RANGE  (etc.)
```

```
Columns: 1 - - - - - - - - - - - - - 73 - 80
         ƀVIOLIN = 3G - 6F+ , SET VIOLIN RANGE
```

In the last example, an instruction is followed by a comment.

2.2.3 Identification Field

Columns 73 to 80 are ignored by the MUSICOL compiler, but are listed 10 places after column 72 in the source listing. This field may be used for card numbering, etc.

Example 2.4

```
Column: 1 - - - - - - - - - - - - - 73 - 80-
        ƀSTART    TEST PIECE (etc.)     TEST 001

        ƀƀORDERS (PPP PP MP  FT FF)     TEST 002
```

III. MUSICOL ELEMENTS

3.1 Control Statements

MUSICOL Control Statements operate at the compilation level only. They are not stored for simulation. Therefore, no locations are reserved in the simulated memory, and the location counter is not incremented.

3.1.1 START

To signify the beginning of MUSICOL source code, a START instruction must appear after column 1 on a single card, before all other instructions except control statements. The remainder of the card (to column 72) cannot contain MUSICOL instructions or control statements, as it is reserved for the title field (see 4.1). If the START-card is not properly located, a fatal error will result and a diagnostic will be printed, aborting the program listing.

Example 3.1

```
Columns: 1 - - - - - - - - - - - 72 - 80
        ┌──────────────────────────────────┐
        │b̸STARTb̸' Title (optional)          │
        │                                  │
        └──────────────────────────────────┘
```

3.1.2 END

An END statement follows the last MUSICOL source-card. Any statements in the column following the END instruction (to 72) will be rendered as comments. An error is triggered if the compiler fails to encounter an

END statement as the last instruction.

Example 3.2

```
Columns: 1 - - - - - - - - - - - - - 72 - 80
         ┌─────────────────────────────────────┐
         │b̸ENDb̸  Comments (optional)           │
         │                                     │
         └─────────────────────────────────────┘
```

3.1.3 DUMP

For octal instruction listings (see Examples 1.1 and 1.2) and debugging purposes, a DUMP instruction may be placed at any point in the program. When a compilation error is incurred, the dump is automatically switched on, and the octal instructions appear to the right of the location counter, with the remaining source-code instructions. The fields in which errors occur are filled with 7's.

In the simulator, the contents of all the simulated-memory locations which store the results of MUSICOL instructions are dumped for each time block in which simulation errors occur (see 5.1.3). A compiler dump listing is shown in Appendix D, as well as the simulator dump.

3.1.4 NOLIST

A NOLIST option is available to be used once the MUSICOL program is debugged. Part or all of the source-listing can be suppressed to conserve paper and compilation time. Placement of the NOLIST statement before the appropriate card will result in a suppression of the ensuing source-listing.

3.1.5 PUNCHI

To duplicate a source deck, the PUNCHI instruction is inserted. PUNCHI can also be put into effect at any point in the program, so that only the desired amount of cards are produced.

DUMP, NOLIST and PUNCHI need not be located on separate cards. They are admissable alongside any MUSICOL instructions.

3.2 Mnemonic Types

A basic explanation of the various MUSICOL mnemonics is presented here. Their correct usage in MUSICOL programming is detailed in Chapter 4. Each of the following headings contains the numbers of the mnemonics referred to, corresponding to the complete listing in Appendix A.

3.2.1 Instrument (Nos. 38 - 67)

The instrument mnemonics are arranged according to family (strings, winds, brass, percussion). Any combination may be used, but no more than sixteen instruments can appear in a composition during one time block. The non-pitched percussion instruments (PERC1, PERC2, etc.) have no specific names (except CYMBAL). Any non-pitched instrument(s) can be represented by these mnemonics.

3.2.2 Pitch (Nos. 1 - 17)

There are seventeen pitch mnemonics, the twelve chromatic tones, and five enharmonic (black-key) spellings. The + signifies a sharped pitch, thus F+ = F sharp; similarly - denotes a flatted pitch (i. e. B- = B flat).

3.2.3 Attack (Nos. 18 - 28)

Only the most general attacks are given in the mnemonics listing. Any desired special attacks may be added to a program (see 2.1.2). The permanent list contains three mnemonics which pertain only to stringed-instruments (ARCO, PIZZ, COLLEG). These are never distributed to other instruments during the compositional process.

3.2.4 Duration (Nos. 68 - 78)

The duration scale runs from a dotted whole to a thirty-second-note. All durations are spelled normally, except sixteenth (16TH) and thirty-second (32ND). A + is used in place of a dot, therefore HALF+ denotes a dotted half-note.

3.2.5 Dynamics (Nos. 79 - 86)

All dynamics from PPP to FFF, including MP and MF are available. The standard abbreviations are used, with the exception of FT (forte), to distinguish from the pitch F. Crescendo (CRESC) and diminuendo (DIMIN) are also listed for individual usage.

3.2.6 Octave Range (Nos. 89 - 95)

The octave parameter is specified in the general sense by seven mnemonics which correspond to seven octaves of the piano (8VE1,....,8VE7). Each operation code encompasses twelve pitches (A - A flat) in its designated octave.

A related matter to octave range involves absolute pitches. These are indicated by prefixing the pitch with its octave number.

Example 3.3

```
6A- = 6th octave, A flat
4C  = Middle C (4th octave, C)
```

More examples for the correct usages of octave range and absolute pitch are explained in 4.3.1, 4.4.1, 4.5.2, and 4.5.3.

3.2.7 Timbre (Nos. 96 - 101)

Only the most common of tone color possibilities are given as permanent operation codes. All are abbreviated clearly in their mnemonic form. NORMALT (ordinaire) is used to distinguish from NORMAL in the attack parameter. Those permanent mnemonics that do not apply to all instrument-types will only be selected by the correct ones.

If additional timbres are desired, they can be employed by replacement of assigned mnemonics (XTRA1, etc., see 2.1.2).

3.2.8 <u>Special Action Instructions</u> (Nos. 87, 88, 102-105, 119, 120, 141)

For special musical actions, these operations codes are used to determine texture (PLAY, REST), random parametric content (ROWGEN, SHUFFLE, RANDOM), overall crescendi, diminuendi, accelerandi, ritardandi, etc. A full detailed explanation is given in 3.3.1 and Chapter 4.

3.2.9 <u>Overall Range Instructions</u> (Nos. 112 - 117)

To set limits on duration, dynamics, and frequency (pitch) ranges, and to set the time signature and metronomic markings, overall range instructions are available. The time signature and metronomic markings may be changed if desired, in successive time-blocks. The lengths of the time-blocks are set in amounts of thirty-second-notes.

3.3 <u>Parameter Generating Processes</u>

Three general methods are presently available to generate parametric content. Various aspects of their techniques can be used alone or simultaneously within a time-block, to achieve desired parametric configuration. The logic involved in utilizing the following procedures in a MUSICOL program, is fully discussed in Chapter 4.

3.3.1 Random Processes

The random generator (see Appendix F) is used to some extent in all parametric generation, with the exception of the literals declarations. If no rank order instructions are encountered in a time-block, or if any such instructions are cleared (RANDOM, see 4.3.7 and 4.5.4.1), total randomness is assumed (zeroth-order). Hence, the parameter(s) affected are selected randomly from a complete list of the pertinent mnemonics.

If a partial list is desired for random selection, those mnemonics may be listed within the time-block. In the compositional process, only those in the declared list will be eligible for selection by the random generator. In addition, the elements of a declared list can be generated randomly (see ROWGEN, 4.4.2). These elements would, in turn, be chosen by the random process. Furthermore, the elements of the declared list, whether they were determined by the composer, or the random generator, can be shifted about (see SHUFFLE, 4.4.3). This shuffling procedure can be called at any point in time, repeatedly, at the discretion of the composer.

3.3.2 Rank Order Distribution

A more sophisticated aspect of computer-assisted composition constitutes the application of stochastic methods for parametric selection. Each element belonging to a parameter-class is assigned a rank order, that is, a percentage representing the probability of its occurrence.

The most basic stochastic process in MUSICOL determines the texture of the composition by means of a play/rest ratio. Percentages of play/rest can be set inclusively for all instruments or for each individual one. If a play probability of 75 percent is assigned, a 25 percent chance of rest is automatic. To produce the percentages, the random processor selects integers from one to one hundred. Since all one hundred integers are equally probable in a random situation, the specified percentages are derived by adjusting the "play range" accordingly.

Example 3.4

Play = 75% (Rest = 25%)
$1 \leq I \leq 100$
if $I \leq 75$, Play option is triggered
if $I \geq 76$ and
$I \leq 100$, Rest option is triggered

The elements of the declared list (introduced in 3.3.1) can be subjected to stochastic selection by the use of rank order declarations. These proportional percentages are chosen according to probability distributions as shown in 3.3.2.1 and 3.3.2.2.

3.3.2.1 Basic Probability Distribution

To generate probabilities for parametric elements in a string, the random process (3.3.1) can be used by repeating elements in accordance with their desired rank distribution.

Since each element in a string receives equal distribution, the re-occurrence of the same element will increase its probability. In the following list:

(B A B- B C D- F B C B A C)

the pitch B (4 occurrences) will have the greatest probability, 4/12 or 1/3, followed by C (3 occurrences) with 3/12 or 1/4, then A (2 occurrences) 2/12 or 1/6. B-, D-, and F (one occurrence each) will have the same probability, 1/12 each.

This process, of course, can be used successfully on all MUSICOL parameter strings. The maximum list size of 24 elements allows a wide range of probability manipulations.

3.3.2.2 Zipf's Law

Statistical analyses of spoken and written languages have always received the attention of linguists, psychologists, and telecommunication engineers. It is clear that in the evolution of paradigmatic languages, the syntactical changes were influenced by the frequency of occurrence of words, phrases, etc. Subsequently, those words or phrases used more often eventually became shorter.

One of the more simplistic approaches to the statistical aspect of language economy is the "dot - dash" code of Samuel Morse of 1832. Morse merely tabulated the quantities of the type letters found in a printer's office. He then constructed his code so that there was a direct relationship between the brevity of the symbolic representation and the quantity of type for each letter. By establishing this hierarchy, Morse codified the English language in messages using, on the average, the least amount of symbols.[1]

Observing Morse's original code and the way that languages evolved, G. K. Zipf asserted that languages and other behavioral forms were governed by what he called the "Principle of Least Effort".[2] He believed that man, as a goal-seeking being, will always attempt to minimize the amount of work necessary to complete a task. Thus, studies measuring redundant elements in various systems should result, hypothetically, in relatively uniform results. Although Zipf experimented with this principle in many areas of behavior, his work with languages stands out as the most relevent for musical parallels.

Zipf collected data comprising of a statistical word-count of James Joyces' Ulysses as well as that of samples from American newspapers. The elements (words) were listed in the order of their frequency of occurrence, or rank - ordered. The results of these rank orders, plotted against the frequency, show two relatively linear graphs.[3] Ideally, such a graph is the basis for the generation of rank order probabilities in MUSICOL. Hence, each successive rank order is selected at a probability equal to the inverse of its number.

Example 3.5

Rank orders:	1, 2, 3, 4,,,,,,,,n
Probability (P):	1/1, 1/2, 1/3, 1/4,,,,1/n

To calculate the relative probabilities (P), it is necessary to multiply the inverse of each rank order number by a constant (K), equal to the reciprocal of the sum of elements 1/1 through 1/n, that is:

<u>Example 3.6</u>

$$K = 1\Big/\sum_{i=1}^{n} P_i \qquad (3.1)$$

if n = 5:

$$\frac{60}{60} + \frac{30}{60} + \frac{20}{60} + \frac{15}{60} + \frac{12}{60} \quad x \quad K = 1.0 \quad (100\%)$$

$$K = \frac{60}{137}$$

for individual probabilities:

$$P_r = 1/r \quad x \quad K \qquad (3.2)$$

P_1 = .438	or	43.8%
P_2 = .219	or	21.9%
P_3 = .145		14.5%
P_4 = .11		11.0%
P_5 = .088		8.8%

In MUSICOL, the declared lists can be assigned rank order numbers. The programmer decides which element is to be the first. In the following:

```
(PPP FT MP FF MF PP P)
```

if PPP is the first rank order then:

```
       (PPP FT MP FF MF PP P)
rank:   1   2  3  4  5  6  7
```

The first element of the string need not always be the first order. Thus:

```
        (PPP FT MP FF MF PP P)
```

if FF is the first order, then:

```
        (PPP FT MP FF MF PP P)
  rank:   4   3  2  1  2  3 4
```

In the last case, elements sharing the same rank order number will also share the probability on a 50/50 basis.

<u>Example 3.7</u>

```
        (PPP FFF MP MF FF PP P FT)
  order:  4   3  2  1  2  3 4  5
```

Probabilities: P(MF) = .438

P(MP) = .11

P(FF) = .11

P(FFF) = .0725

P(PP) = .0725

P(PPP) = .055

P(P) = .055

P(FT) = .088

To establish desired probabilities for elements, it may be necessary to repeat elements as needed.

<u>Example 3.8</u>

```
         (MF FT MF PP P MP P MF)
  orders:  4  3  2  1 2  3 4  5
```

Probabilities: P(PP) = .438

P(MF) = .2525

P(P) = .1645

P(FT) = .0725

P(MP) = .0725

A complete, detailed explanation of the implementation, declaration and use of rank orders is contained in Chapter 4.

3.3.3 Literals

To increase programming flexibility, exact selection of parametric elements is possible in MUSICOL. If a declared list contains only one element, that element will always be chosen. In a string, an element can be declared as the first rank order of one possible order. Therefore, its probability is 1.0 or 100%. Specific durational lengths within time-blocks can be specified for literals of any parameter for any individual instrument. This frees the remaining time for that particular instrument to be controlled by other declarations pertaining to the affected parameter. (For programming information, see 4.41, 4.5.1.1, 4.5.4, 4.6.2 to 4.6.4.)

FOOTNOTES, CHAPTER 3

[1] Colin Cherry, _On Human Communication, A Review, A Survey, and A Criticism_, The M.I.T. Press, Cambridge Mass., 1968, pp. 36 - 37.

[2]Cherry, _loc_. _cit_., p. 103.

[3]Cherry, _loc_. _cit_., pp. 103 - 108. The graph, and a clear explanation are found here to clarify the discussion.

IV. MUSICOL PROGRAMMING

Programming a musical composition in MUSICOL requires the proper and orderly implementation of the operations described in this chapter. The operation types are discussed in a suggested order for consideration, ranging from the general to the specific.

4.1 Title Field

As explained in 3.1.1, the title field follows the START control statement. The composer has the option of submitting a title, which will then appear at the heading of the output.

4.2

4.2 Time-Block Structure

The fundamental structure of a MUSICOL program is the time-block. Whenever a change in a permanent declaration is desired, a new block must begin. Any number of time-blocks may be declared in a sequence during the course of a MUSICOL program. Time is specified in the number of the measure plus the amount of 32nd notes. A decimal point separates the two fields which are preceded by the mnemonic TIME. Therefore:

TIME = 3.8

denotes the beginning of the second quarter (after 8 - 32nd notes), of the third measure. Likewise:

TIME = 4.0

specifies the beginning of the fourth measure.

The first block of a program starts as either 1.0 or 0.0. The duration of the block is equal to the difference of the two TIME declarations. All MUSICOL instructions pertaining to that block must be listed within the area between the TIME statements. The last time-block simply ends with a TIME declaration. Any code, other than control statements, located between the last TIME declaration and the END statement will be ignored by the MUSICOL simulator.

Example 4.1

```
                  ┌ TIME = 1.00  TIMSIG=4/4 QUARTER =60
   duration       │  FRQRANGE = 2C - 6F+ FLUTE =4C -6C
  11 1/2 bars    <   etc...............
                  └ TIME = 11.16  (more code) etc........
   duration       ┌   ....................
  19 1/4 bars    <    ...............
                  └ TIME = 30.24 etc....................
                      ....................
                      ...............
                    TIME = 60.00
                    END
```

4.3 Permanent Declarations

Permanent declarations are those that remain in effect until changed, that is, they need not be reinstated in each new time-block. All the instructions discussed in 4.3, 4.4, and 4.5 are of the permanent type. Permanent declarations may be changed as frequently as needed.

4.3.1 TIMSIG

To set the time signature the mnemonic TIMSIG is used, followed by the desired numerical values separated by a slash.

Example 4.2

```
TIMSIG = 3/2
TIMSIG = 2/4
TIMSIG = 6/8      etc.
```

Since the smallest existing note value in MUSICOL is the 32nd, the "denominator" cannot exceed 32. Any number but zero is acceptable as the "numerator". If TIMSIG is not present in the first blokk, 4/4 is assumed as the initial meter.

4.3.2 Metronomic Marking

Metronomic markings are specified by the note value and the number of beats per minute.

Example 4.3

```
QUARTER = 56
HALF = 38
EIGHTH = 144      etc.
```

If no initial metronomic marking is declared, If no initial metronomic marking is declared, QUARTER = 60 is assumed.

4.3.3 FRQRANGE

The frequency range is the overall pitch range, governing all instruments. The lowest and highest absolute pitches (see 3.2.6) are separated by a dash.

Example 4.4

```
FRQRANGE = 2C - 6A+
FRQRANGE = 3G- - 7B
FRQRANGE = 4D+ - 5A       etc.
```

If a pitch for any instrument is generated beyond the limits designated in a FRQRANGE instruction, it is transposed by octave shifts until it complies with the range. FRQRANGE is not a mandatory instruction.

4.3.4 DYNRANGE

An optional DYNRANGE instruction is available for overall dynamic range declaration. The limits are represented by dynamic markings. When no DYNRANGE instruction instruction is present, total availability of all dynamics is assumed.

Example 4.5

```
DYNRANGE = PP - FF  (includes PP, P,
                     MP, MF, FT, FF)
DYNRANGE = MP - FT
Dyn
```

```
DYNRANGE = PPP - P        etc.
```

Any dynamic marking selected out of range is changed to the nearest dynamic limit.

4.3.5 NTRANGE

The optional overall durational range is implemented exactly like DYNRANGE.

Example 4.6

```
NTRANGE = HALF - 16TH  (includes HALF,
                        QUARTER+, etc.
                        to 16TH)

NTRANGE = WHOLE - QUARTER+
NTRANGE = QUARTER+ - 32ND    etc.
```

4.3.6 Play/Rest

PLAY and REST are used in determining the play-rest ratio, as described in 3.3.2. Either mnemonic may be used to set the overall play/rest ratio. An asterisk precedes the percentage number.

Example 4.7

```
PLAY * 75  (play: 75%, rest: 25%)
REST * 60
PLAY * 100          etc.
```

When no initial play/rest ratio is set, 50/50 is automatic.

4.3.7 RANDOM

To clear instruments of all previous specific instructions except the voice number declaration (to be described in 4.5.1), a RANDOM instruction can be used. This randomizes all parametric selection, unless new declarations are made after the random-clear operation.

The overall RANDOM instruction is somewhat limited in that it can clear from one to all of the instruments involved, but only in the order in which the instruments were originally declared. Thus, if there are six instruments and two are to be cleared, only the first two in the order will be cleared. The assigned number (greater than zero) is preceded by an asterisk in the instruction.

Example 4.8

```
RANDOM * 2
RANDOM * 16
RANDOM * 1           etc.
```

It is obvious that the general random instruction is most useful for bulk random-clearing. A more versatile usage of RANDOM is described in 4.5.4.1.

4.4 Parameter Strings

In order to select from picked groupings of parametric elements, it is necessary to construct lists. These strings are set up and changed liberally throughout a program and may serve many functions (see 3.3.2.1). There are three basic operations used to manipulate them. All require the use of parentheses as delimiters. Once the operation is defined, parametric strings may be positioned adjacent to one another. When code other than the three following operations is interspersed, the correct operation code must be repeated before continuing with more lists.

4.4.1 ORDERS

When parametric strings are to be determined by the composer, the ORDERS declaration is used. The declared lists will enter the simulator exactly as compiled.

Example 4.9

```
ORDERS(PPP MP FT MF P PP FF) (C D+ F G- A B-
 F+) (SLUR SFFZ ARCO PIZZ NORMAL) (8VE1 8VE2
 8VE3 8VE4 8VE5) (WHOLE HALF QUARTER HALF+
 QUARTER HALF+ EIGHTH QUARTER+)
```

If only one element is placed in a list, that element is the sole choice for the parameter represented (probability of 1.0 or 100%).

4.4.2 ROWGEN

ROWGEN introduces a random process (see 3.3.1) to the selection of elements. It is used to generate strings randomly in accord with the type and number of elements specified. After the mnemonic ROWGEN, one of any of the representative elements belonging to the chosen parameter class is inserted. This is done merely to identify the class and has no influence on the selection order, nor does it insure the selection of that particular element. A number denoting the length of the list follows the representative element, both of which are enclosed in parentheses.

<u>Example 4.10</u>

```
ROWGEN (E 12)          (pitch)
ROWGEN (SFZ 3)         (attack)
ROWGEN (MP 5)          (dynamics)
                   (etc.)
```

4.4.3 <u>SHUFFLE</u>

Yet another usage of the random process (see 3.3.1) is found in the SHUFFLE operation. SHUFFLE simply shifts the order of elements in a parametric string that was previously created by an ORDERS or ROWGEN operation. The format of the SHUFFLE operation is exactly that of ROWGEN. The representative element which defines the class is followed by a number indicating the amount of elements in the list to be shuffled.

<u>Example 4.11</u>

```
SHUFFLE (B  12)
SHUFFLE (ARCO 2)
SHUFFLE (8VE2 5)             etc.
```

It is not necessary to shuffle all of the elements in a list, but shifting begins at the left. Thus, in the following operations:

```
                              ORDERS (FFF MP FT MF P PP FF)
                                 SHUFFLE(P  4)
```

only the first four elements will be shuffled, allowing the final three to remain intact.

When a ROWGEN or SHUFFLE operation is encountered by the simulator, the operation is executed immediately, and a list of the new order of elements is printed out under the "EXECUTION CONDITIONS" portion of the MUSICOL listing (see Appendix B).

Although it is allowable to proceed with ORDERS strings directly after a ROWGEN or SHUFFLE declaration without inserting the ORDERS mnemonic, ROWGEN and SHUFFLE operations must be redeclared with each usage.

Example 4.12

```
                              ROWGEN (C 12) (PPP MF FT P MP FF)
                    (8VE1 8VE2 8VE3 8VE4) SHUFFLE (8VE1 4) ROWGEN
                    (SLUR 6) (NORMALT SULPONT MUTED GLISS FLUTTERTGE)
```

A zero can be inserted as the numerical value in a ROWGEN or SHUFFLE instruction. This clears the designated parametric string from simulation memory, and control of that parameter falls to random process until respecified.

Example 4.13

```
ROWGEN (C 0)
SHUFFLE (SLUR 0)
ROWGEN (PP 0)          etc.
```

4.5 Instrument Declaration

A maximum of sixteen instruments is admissible at any given time. This total includes those specified in the operation code list (Appendix A) as well as those assigned (see 2.1.2). An instrument is declared when the amount of possible voices is assigned.

4.5.1 Voice Number

The voice number specification must be in the first operation of any instrument declaration. Failure to do so will result in an error, with the issuance of a diagnostic. In the case of assigned instrument mnemonics, the voice number may follow the replacement code. Any number of voices from one to four can be assigned for each instrument. The voice number is enclosed in parentheses.

Example 4.14

```
VIOLA (2)
TRUMPET(1)
INSTR1 = BASSFLUTE(1)        etc.
```

For instruments capable of more than four simultaneous attacks, another name for the instrument may be assigned in addition. For keyboard instruments one declaration can be initiated for each hand, counting as two instruments in the simulator. The pitch ranges are adjusted for each hand (see 4.5.2).

Example 4.15

```
                    PIANO(4)
                    INSTR1 = PIANOL(4)
          or
                    INSTR1 = HPSCHDL(4)
                    INSTR2 = HPSCHDR(4)
```

4.5.1.1 Voice Number Rank Orders

On multiple voiced instruments, rank order numbers may be assigned for selection of the amount of voices per instance, according to Zipf's Law (see 3.3.2.1). The voice number to be the first order is preceded by a period, and the amount of possible orders follows an asterisk. If any of these values exceed the declared number of voices, an error is detected.

Example 4.16

```
FLUTE.1*2  (Selection: one voice is the
            first order out of two
            possible voices (1 and 2))
VIOLIN .2 *2
CELLO. 3 * 1          etc.
```

Declared parametric lists are not possible for voice rank orders. Hence, rank orders are in numerical order, as seen in Example 4.16.

4.5.2 Pitch Range

Another mandatory declaration for each instrument is the pitch range specification. The intended range of the instrument is given in absolute pitches (see 3.2.6), as in FRQRANGE (see 4.3.3). Of course, here the range is prefaced by the instrument name.

Example 4.17

```
CLARINET(1) = 3E - 6G
VIOLA = 3C - 6C+
TUBA = 2B- - 4A          etc.
```

No pitch range is declared for non-pitched instruments.

Care should be taken when using instrumental range declarations in conjunction with FRQRANGE to avoid cancellation of the resultant pitch range. In the following operations:

```
FRQRANGE = 2C - 3F+
VIOLIN = 3G - 7A
```

the narrowest range prevails in order to comply with both statements. That is to say that any pitch above 3F+ or below 3G cannot be selected, an impossible situation. If this type of error is encountered, a diagnostic is provided.

4.5.3 Instrument Play/Rest Ratio

The play/rest ratio for specific instruments is employed in the identical manner of the general play/rest (see 4.3.6). The statement however, is linked to the instrument by an equal sign. The absence of this optional statement will assign control to the general ratio for the instrument.

Example 4.18

```
OBOE = PLAY * 75
VIOLA = REST *60
TROMBONE = PLAY * 100         etc.
```

4.5.4 Individual Rank Order Specifications

The ability to set rank orders for individual parameters of each instrument exists in MUSICOL as a voluntary instruction. Rank orders of parametric elements can be specified only when a declared list for that particular parameter is present (see 3.3.2.1, 4.4 - 4.4.3). The element chosen for the first order is entered with the amount of possible rank orders following. An asterisk separates the two values.

Example 4.19

(in connection with Ex. 4.12)

```
OBOE = 8VE4*1 = FT*3 =FLUTTERTGE * 1
CONTRABASS = D+ *7 =PPP*2=8VE1*2
           = MUTED*2
                                 etc.
```

When an element is declared as the only rank order (such as 8VE4 * 1) its probability is 1.0 or 100%. Thus, the programmer can assert total control over selection if desired.

Since individual rank order declarations are permanent, they remain in effect until changed. If a respecification of orders is not desired, a zero may be inserted as the numerical value, erasing the governing rank orders instruction from simulation memory. A random process then prevails in the selection of elements from the parametric string.

Example 4.20

```
CONTRABASS = D * 0 = P*0
OBOE = NORMALT *0
HARP = GLISS *0            etc.
```

If the element being assigned as the first order does not exist in the parameter string, an error will be incurred. The diagnostic will locate the time-block containing the error and identify the instrument and erroneous element.

4.5.4.1 Random Clear

An easy, quick way to clear individual instruments of all permanent declarations, except voice number and pitch range, is to use the RANDOM-clear instruction. This renders random control to all parametric selection for the instrument. New declarations can be enlisted directly afterwards, where desired. When the RANDOM mnemonic is encountered by the simulator, it is immediately executed.

Example 4.21

```
                    (in connection with Ex. 4.19)
               OBOE = RANDOM
               CONTRABASS = RANDOM =8VE2 *1 =PPP *2
                          = PLAY * 33
```

In the preceding example, the simple statement: OBOE = RANDOM is equivalent to OBOE =8VE4 *0 = FT * 0 = FLUTTERTGE * 0 = PLAY *50, etc., a more cumbersome operation.

4.5.5 Instrument Deletion and Replacement

When the composer wishes to terminate the use of an instrument, or have it rest for a very lengthy period of time in a piece, a deletion statement may be employed. This saves MUSICOL execution and output time, as the compositional process for the instrument otherwise continues to output rests. The instruction for deletion simply incorporates the END mnemonic with the instrument-name.

Example 4.22

```
                    VIOLA = END
                    OBOE = END
                    CONTRABASS = END          etc.
```

The instrument can be redeclared later, if needed, by the standard procedure (as in 4.5 et segue).

In the case of replacing one instrument with another at the same point in time, a time-saver statement is available to eliminate statements of deletion and declaration. The new instrument inherits all the special instructions of its predecessor. The new name is placed to the right of the old one, in the instruction. An equal sign separates the two. All ensuing declarations must pertain to the new instrument-name.

Example 4.23

```
OBOE = CLARINET
VIOLA = CELLO = 2C - 4G = ...(etc.)
  ...............
  .........
CELLO = PP*1 = F*5       etc.
```

4.6 Temporary Declarations

A temporary declaration is one that remains effective only for a specified length or the duration of the time-block in which in occurs. As in all types of declarations, control by temporary instructions commences at the beginning of the block. When the designated time expires, control is resumed by permanent, or new temporary instructions.

4.6.1 General Rank Declarations

For overall rank-ordered designations the instruction is similar to the instrumental permanent declaration (see 4.5.4), however the name of an instrument is not needed. Also, the numerical order number is non-functional, so any integer may be inserted (1 is suggested). Hence, all elements available in a parametric string will be assigned order numbers, given the mnemonic of that which is to be the first.

Example 4.24

```
PIZZ * 1
MP *1
WHOLE * 1            etc.
```

The general rank order declarations govern all instruments having no permanent order specification for the featured parameter.

4.6.2 Accelerando and Ritardando (ACCEL, RITARD)

Gradual changes in tempi are easily implemented by the use of ACCEL and RITARD statements. The desired instruction includes the correct mnemonic, the eventual metronomic marking, and the length of the process in 32nd notes. A period is located in between the latter two values, to denote literal length.

Example 4.25

```
ACCEL = 120 .128   (16 Quarters)
```

```
RITARD = 30.64
ACCEL =144. 108          etc.
```

The length may be greater or less than the duration of the time block in which it is present, but in no case should exceed 4095. The output results are seen in Appendix B.

4.6.3 <u>Crescendo and Diminuendo (CRESC, DIMIN)</u>

Two types of instructions exist for crescendi and diminuendi, those for the entire group, and those for individual instruments.

4.6.3.1 <u>Overall Crescendi and Diminuendi</u>

The format of this instruction is identical to that of ACCEL and RITARD (see 4.6.2), the only difference being that the dynamic goal is placed in between the mnemonic and the length in 32nd notes.

<u>Example 4.26</u>

```
DIMIN = MP . 122
CRESC = FF.80
DIMIN = PP. 256          etc.
```

All other individual instructions concerning dynamics are nullified for the length of a CRESC or DIMIN stipulation.

4.6.3.2 <u>Individual Crescendi and Diminuendi</u>

The DIMIN or CRESC mnemonics in 4.6.3.1

are replaced by the instrument-name to individualize gradual dynamic change. Whether the operation is a crescendo or diminuendo is determined automatically by the simulator through a comparison of the existing dynamic and the projected one, for the instrument.

Example 4.27

```
VIOLA = MP.64
INSTR1= FF. 512
PERC1 = FT .38          etc,
```

4.6.4 Literal Lengths of Parametric Elements

Specific elements can be chosen at any time for any duration (not exceeding 4095) by the composer. Again, the format is identical to the previous instructions, the chosen element placed in the second field (after the equal sign).

Example 4.28

```
VIOLIN = COLLEG .8
FLUTE = FLUTTERTGE.32
INSTR2 =XTRA1 .88          etc.
```

4.7 Linking Together Instrument-Instructions

It is convenient to group together all instructions which pertain to the same instrument in a time-block. When this is done, the instrument-name need only be listed once as the heading of the group. This aids the composer during the programming and debugging stages by expediting the location of specific instructions. Also, compilation time is reduced.

Example 4.28

```
BASSOON(1)=2C-5C=PLAY*70=8VE*3=XTRA1
       *2 =NORMALT *1 = MP*4 =B- *12
       = QUARTER * 6
VIOLIN(2) .1*2 = 3G - 7C =PLAY *55
      = C*10=NORMALT * 1 =SULPONT.8
      =FT *5 = MP.32 = ARCO *4=HALF*3
                                 (etc.)
```

4.8 Programming Logic; Basic Compositional Layout

It is suggested that the sample programs listed in Appendix B be studied thoroughly before attempting to program a composition in MUSICOL. They can provide basic hints that will increase the programmer's accessibility to the features of the language. Carefully planned program-logic will curtail excess program runs, reduce redundancy, and hasten the realization of the composer's musical intentions.

The composer's personal compositional style is not threatened by MUSICOL programming, because the layout of the piece is flexible to individual concepts. If an overall length, texture, and instrumentation is originally planned, then a complete skeletal program can be constructed initially. On the other hand, if the compositional plan is a sectional one, the piece can be programmed and run section by section. Of course, a blending of both techniques is possible and expected. Increased programming sophistication is commensurate with the interpretation of MUSICOL output, and the achievement of musical goals.

V. MUSICOL PRINT-OUT INTERPRETATION

As seen in Appendix B, three stages of printed material result from the submission of a MUSICOL program. The compilation and simulation listings, briefly described in 1.1.2 and 1.1.3, are invaluable for reviewing the logical process, and for debugging. The output (musical realization) is the most important end result, and the final determinant of the composition's completion. The purpose of this chapter is to discuss the basics of debugging and output interpretation.

5.1 Debugging

The MUSICOL error systems are thorough and clear. Each error is issued in order of occurrence. All errors are fatal to the extent that the program will not proceed to the next level of execution. Thus, if errors are incurred at the compilation stage, the program is aborted, never entering the simulator. When simulation errors are detected, the compositional process terminates immediately, but the remaining preliminary simulation steps continue. Afterwards, the output file is dumped, producing the composed material, if any.

Although error messages are essentially self-explanatory, a concise explanation is given for each one in the error listings in Appendix C.

5.1.1 Compilation Errors

All compilation diagnostics are numbered and are printed immediately on the line following the erroneous code. In some cases the location is given as a reference. As stated in 3.1.3, the compiler-dump is automatically switched on for the remainder of the listing. The error fields in the octal-representations are filled with 7's. In most cases, one error may trigger a series of ensuing errors which would otherwise seem correct. Usually when an error cannot be logically rationalized, it is the result of one previously incurred. Appendix D contains a MUSICOL listing with compilation errors. (compilation dump is switched-on automatically)

5.1.2 Simulation Errors

Simulation diagnostics are not numbered like those of the compiler, but are similarly outlined with asterisks. The location of the infraction is always given by reference to the location-counter if it occurs in the simulation stage. The time-block is identified when an error is incurred during execution. Because simulation errors usually involve faulty programming logic, a closer scrutiny may be necessary to correct them. As in the compiler, confusing errors may stem from a previous one. Thus, a single correction of the initial error could clear subsequent ones.

5.1.3 Simulator Dump

In the event of a stubborn problem in MUSICOL logic, a simulator dump is triggered on by the occurrence of simulation errors. The contents of simulated memory-locations are dumped (listed) for each time-block in which diagnostics are issued. Specific (individual) declarations are listed first.

The instruments are represented octally in the order that they were declared, under the "Instrument" designation. All other control factors in effect for that time-block are similarly listed for each instrument under the appropriate heading (Voices, Pitch, Attack, Duration, Dynamics, Octave Range, Timbre, Special Actions, Instrumental Pitch Range, Play/Rest ratio, Literal Action). If rank orders or literal lengths, etc., are applicable, the octal words are divided into four-digit groups.

General declarations (i.e. Pitch Range, Dynamic Range, Time Signature, Play/Rest ratio, Special Literal Actions, and Metronomic Marking) are listed similarly, but the 16-digit words are sometimes split into groups of eight, instead of four. For example, 4/4 is represented as 0000000400000004 under the time signature heading. A complete structural breakdown of the words is listed in Appendix D.

Existing "Orders Strings" are labeled and listed separately. If rank order-numbers are in effect at the exact moment of the infraction, they will show in the rightmost four digits of the words representing the specific parameter strings. Appendix D contains a sample listing with a simulation-memory dump.

5.2 Composition-Output-Listing

The actual listing of the composition, programmed in MUSICOL, is clearly printed for easy transcription. The total duration is approximated by the time signatures in conjunction with the metronomic markings throughout the piece. The parameters and actions are listed under the appropriated columns. "SPEC 1" and "SPEC 2" list overall "crescendi" or "diminuendi" and "accelerandi" or "ritardandi" respectively. A mnemonic listed in a column remains in control until a new one appears. Dashes indicate that a "crescendo, diminuendo, accelerando", or "ritardando" is in operation.

The "POSITION" column conveniently designates at what point in time an event (articulation) begins. The bar, beat and subdivision numbers are listed chronologically. The subdivision values refer to the beginning of the 32nd-note location specified.

Example 5.1

BAR 1	BEAT 1 + 1	= Beginning of the first measure.
BAR 2	BEAT 2 + 9	= The beginning of the quarter-note after the 2nd beat of bar 2.

(etc.)

Often, near the end of a time-block, the chronology may be broken because an instrument, very close to being completed, is allowed to finish before the control returns to the next instrument furthest from completion.

VI. PROPOSED MODIFICATIONS (VERSION 2)

Although there is a great degree of flexibility in MUSICOL which is advantageous for a composer who programs in it, some obvious limitations do exist. In the future, an improved MUSICOL, Version 2, will further expand the versatility of the language by introducing new features, thus eliminating many of the present shortcomings. This chapter discusses a few of the more important limitations to be dealt with in Version 2.

Harmony in Version 1 is essentially a by-product of the specified linear structures taken vertically. A composer is able to influence the selection of types of harmonies by carefully manipulating pitch and rhythmic distributions, but a new procedure must be developed to allow direct access to controlling vertical simultaneities. Also, there is no initial relationship between parameters, that is to say, the choice of one element in a given parameter class (i.e. duration) will have no control over the choice in another type (i.e. attack). Such a feature is asthetically important, as this concept is quite relevent to successful musicality.

Desired probability distribution can be programmed quite easily in the present version, as shown in 3.3. However, more sophisticated principles like Zipf's Law, namely those like Xenakis' and Iching (as programmed by L. Hiller) could be implemented as a MUSICOL operation, to avoid the composer's involvement with complicated mathematics, should he attempt to program the distributions himself (i.e. as in 3.3.2.1).

There are several existing MUSICOL mnemonics that are not yet implemented. Most of them depend on manipulation of material already composed. A search and store program will be developed to make composed material accessible.

Finally, the "timbre" and attack functions will be redistributed into three groups, so that a more flexible selection of elements is possible.