MICROCOMPUTERS AND MUSIC

by Gary E. Wittlich, John W. Schaffer, and Larry R. Babb

Englewood Cliffs, Prentice-Hall, 1986

Reviewed by

Alexander R. Brinkman

Considering the proliferation of microcomputers in all aspects of the educational experience, it is not surprising that books specializing in specific subject areas should begin to appear. Although the literature on computer music composition and synthesis has seen a great deal of activity during recent years, the present text is one of the first intended as an introduction to general music processing on microcomputers. The primary objective of the authors is "to demonstrate how microcomputers can be used effectively in the solution of problems of interest to musicians," using three commonly available microcomputers—the Apple II family, the Commodore 64, and the IBM PC. The text specifically avoids the area of music composition, and although some applications in music theory are explored, the primary emphasis is on techniques that are useful in computer assisted instruction (CAI) in music.

The text uses the BASIC computer language, and presumes that the reader already has at least an elementary knowledge of

BASIC programming, control structures, and subroutines, or is willing to obtain these from other sources. More advanced techniques—string functions, programmer defined functions, and arrays—are defined in the text and illustrated by programming examples, as are the use of graphic and sound producing functions. The book is organized into seven chapters and five appendices:

1. Introduction to Top-Down Design and Structured Programming
2. Data Representation and Manipulation
3. Data Structures for Music Applications
4. Structured Programming for Computer-Assisted Instruction Lessons
5. Microcomputer Graphics
6. Computer-Generated Sound
7. Top-Down Design Examples

Appendix A: A Guide to Basic
Appendix B: An Apple Shape Maker
Appendix C: Memory Problems on the Apple II
Appendix D: Machine Codes for Screen Scroller Portion of Melody Maker Programs
Appendix E: Solutions to Exercises

The first chapter is a cogent discussion of the process of designing well-structured programs. Throughout the book the authors recommend and use a style of programming called structured programming, a technique that includes top-down design, stepwise refinement, and encoding the program in well-defined modules, each of which performs a specific task and has only one point of entry and one exit. These techniques lead to clearer program structure, and the resulting programs are generally easier to debug, maintain, and

extend. Certain computer languages, such as Pascal, were designed with these objectives in mind. Although BASIC is not one of these and lacks many features that facilitate the technique, the authors do an excellent job of implementing these goals. They present a systematic approach for developing programs that includes a clear statement of the problem in English, the use of pseudo-code and flowcharting, solving large problems by breaking them down into successively smaller steps, and the use of subroutines to implement modules. The authors also use indentation, blank lines, and well-delineated comments to help clarify program structure.

Chapter 2 introduces several music representations that are useful for data entry and storage. The authors give concise summaries of DARMS and MUSTRAN, the two most complete and widely used alphanumeric music encoding schemes. In each case the subset described is sufficient for encoding simple single-line (melodic) excerpts, and the chapter includes references to sources with more complete information. Presumably the section on DARMS is included for information only, since this excellent code is not used elsewhere in the book.[1] These codes are followed by a coding scheme more typical of microcomputer applications, which

[1] There are some errors in the DARMS code description. For example, the duration and the optional dot, as stated in the book. Also, the double barline (||) is encoded '//'. The symbol '!/' signifies a heavy (wider) barline, thus the double bar at the end of a piece (||) would be encoded '/!/'.

the authors refer to as LTTR/ACC/OCT/LNTH to indicate the encoding order. The code uses note names [C,D,E,F,G,A,B], accidentals [+ (sharp), ++ (double sharp), - (flat), -- (double-flat)], and octave number [0-8]. Durations are encoded using proportional integer values (12 = quarter note, 6 = eighth note, 4 = eighth-note triplet, etc.). This code and derivatives are heavily used in the rest of the book. This chapter also includes a description of BASIC string functions and illustrates their use in parsing LTTR/ACC/OCT/LNTH code. The last part of the chapter presents numeric systems for representing pitch, and illustrates various musical operations using them. The systems used are pitch class integers (0-11 in a modulo 12 system), diatonic pitch classes (0-6 in a modulo 7 system), and a system that combines chromatic and diatonic pcs and octave numbers in a manner that allows arithmetic manipulation of pitch data but retains precise spelling and registral information.[2]

Chapter 3 discusses data structures for music applications. The authors discuss arrays in one and two dimensions, and illustrate their use with a number of musical problems (e.g., melodic imbrication, note counting, and 12-tone matrices). Other useful techniques such as indirect reference of array elements and the use of

[2] This system was described by the present author in a recent article. See Alexander R. Brinkman, "A Binomial Representaton of Pitch for Computer Processing of Musical Data," *Music Theory Spectrum* 8 (1986): 44-57.

arrays as look-up tables and storage structures are also introduced and illustrated in an informal but clear manner. The discussion of arrays culminates with a procedure for calculating prime forms of pc sets. The algorithm used is essentially Allen Forte's, with an interesting computational twist.[3]

The latter portion of Chapter 3 presents simple linked lists, which are preferable to arrays in some applications because of the ease with which new items can be inserted or deleted. Since BASIC does not provide a pointer data type, the authors show how to implement linked lists using arrays. By way of illustration, they implement a linked list of notes using a matrix (NTE), in which each row constitutes a *node* in the list. Each column in the matrix represents a separate *field* for representing one item of information in the nodes. As an aid to clarity in the program code, the columns are referenced by mnemonic variables that have been assigned integer values, e.g., LTTR (letter name), ACC (accidental), OCT (octave), LNTH (length), LL (left link), RL (right link), etc. Thus, if the variable CRNT is the index (row number) of the current node, the various attributes of each note are referenced as NTE(CRNT,LTTR),

[3] See Allen Forte, *The Structure of Atonal Music*, (New Haven: Yale University Press, 1973). The modified algorithm given in the book is as follows: after sorting the set, all circular permutations with the smallest span are extracted and transposed to zero. For each of these, the intervals are compared symmetrically from both ends, and if the smaller interval is on the right, the set is inverted by reversing the interval sequence. The prime form is the set for which the *sum* of the resulting integers has the least value.

NTE(CRNT,OCT), etc. The next node in the list is obtained by CRNT = NTE(CRNT,RL). This implementation is possible because all of the values are stored in the matrix using the integer representations introduced in Chapter 2. A logical extension that would allow different data types in each field would be to use parallel one-dimensional arrays to represent the fields. Thus, the fields would be referenced LTTR(CRNT), OCT(CRNT), etc., and the next node obtained by CRNT = RL(CRNT).

Chapter 4 introduces design principles for computer-assisted instruction and reinforces the concepts introduced in the first chapter by working through the design of a CAI drill program using the top-down approach. The program is designed to drill interval spelling. The authors present two algorithms based on the circle of fifths that can be used to generate intervals or chords. This chapter also includes the use of sequential text files for permanent storage of data generated by programs.

The first four chapters are general, i.e., they do not depend on any particular microcomputer. Chapters 5 and 6 explore microcomputer graphics and sound production. Since implementation of these techniques differs from computer to computer, and even among different BASIC interpreters on the same computer, much of the discussion must necessarily deal with characteristics of specific systems.

Chapter 5 begins with a good overview of microcomputer graphics, then moves on to specific music problems such as drawing staves, defining music symbols (clefs, notes, etc.), and positioning them on the computer screen. Whenever possible the discussion is generalized; details necessary for implementation on specific computers are set off in grey-tone highlight boxes. The authors cover two different methods of input: alphanumeric music codes, and cursor-driven input in which the user uses arrow keys to choose items from a menu. The culmination of the chapter is a design for a "music editor" that utilizes the linked list implementation from Chapter 3. As music code is entered or changed, the notation appears or is updated on the computer screen. The editor allows for insertion or deletion of notes at any point in the melody, as well as for changing values (duration, accidentals, etc). It also scrolls the graphics left or right to accommodate writing a melody longer than one screen. The text discusses and lists each required procedure.

Chapter 6 deals with sound production on microcomputers, using the internal tone generators and BASIC commands that control them. The chapter begins with an overview of simple acoustic principles, and then shows how tones can be generated on microcomputers. Specific examples are given for each of the three target computers used in the text. The chapter concludes with a program for playing melodies that uses many techniques presented

earlier in the text and is implemented for each of the target machines.

The final chapter contains a series of complete programs which incorporate most of the routines and techniques developed throughout the text. This chapter also reviews the suggested steps for algorithm development and applies them to the programs presented. The programs are complete implementations of the designs covered in Chapters 3 through 6—an interval drill CAI program; a music editor program for writing, editing, and storing melodies; and a melodic analysis program. The latter is a menu-driven program that can calculate interval succession, melodic range, conjunctivity index,[4] and pitch inventory (note counts), as well as search for imbricated melodic patterns. The programs are designed to work together, i.e., melodies created and saved with the music editor can be played by the melody player (Chapter 6) or analyzed by the melodic analysis program. Complete program listings are included, with appropriate variations for each of the target computers.

The Appendices provide additional useful information. Appendix A is a summary of BASIC commands and syntax. Appendix B is a complete "shape maker" program for designing graphic symbols for the Apple II family of computers. In addition to

[4] Their conjunctivity index represents the relative amount of disjunct vs. conjunct motion in a melodic line.

the program listing, there are instructions for loading and initializing the shape table so that it can be utilized by user-written programs. Appendix C addresses memory problems on the Apple II. Because of the way high-resolution graphics are implemented on this system, BASIC programs cannot generally make use of much of the available memory. The authors present a solution to this problem and a program to implement it. Appendix D contains assembly language and machine code segments to implement the routines that scroll the graphics screen left and right in programs presented in the text. Versions are included for the Apple and Commodore computers (they are not required by the IBM PC). Appendix E contains solutions or partial solutions to most of the exercises in the text.

The question of audience for a book such as this is an interesting one. In the preface the authors target three groups: musicians who wish to use the microcomputer for such tasks as data management and analysis, students in a course using the book as a text, and professional and amateur musicians who are interested in applications of the computer to music and would like to learn something about them.

The book is well suited as a textbook for an introductory course in microcomputer applications in music. There are many exercises, which appear after each section of text. Especially in the beginning, the emphasis is on making changes to routines that are

shown in the text. It would be nice if more exercises called for applying creatively the program development techniques suggested in the text. The fact that answers are given to most exercises is useful to one who is teaching himself, but may necessitate designing additional assignments when the book is used as a text in a course.

Exercises might also have provided an opportunity to explore more advanced techniques that are absent from the text. One example is the binary representation of pc sets, which can be implemented with integers. In this representation, a set of n unique pitch-class integers P_i, $1 \leq i \leq n$ is represented by a single integer S, which is the sum of 2^{P_i} for each pitch class, P_i:

$$S = \sum_{i=1}^{n} 2^{P_i}$$

This could be implemented easily in BASIC using an array of pc integers, PC, the exponentiation operator (↑), and a FOR loop:

```
100 S = 0                          : REM set number
110 FOR I = 1 TO N                 : REM for n pcs
120     S = S + 2 ↑ PC(I)          : REM add 2↑pc
130 NEXT I
```

Since this algorithm yields a unique integer value for each possible pc set, its use would improve efficiency in problems such as finding invariant subsets in a twelve-tone matrix (exercise 3.5.2); it also

suggests a more elegant prime form algorithm.[5]

Perhaps because of the wide intended audience, the authors have adopted a fairly informal style, and have avoided technical explanations both in the areas of music and computer science. Some readers will object to a lack of preciseness in use of terms generally used in specific ways. For example, consider the following passage (page 52):

> Much analysis of twentieth-century music deals with sets of notes, that is, collections of notes in which there are no duplicates, such as the set (C,E,G). [Emphasis theirs]

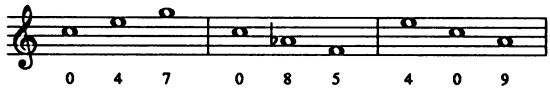
The program that illustrates the concept uses as input a MUSTRAN string representing a melody from a Chopin Prelude. The program extracts a set, also in the form of MUSTRAN code, that represents the following: quarter note E4, dotted-eighth C#5, sixteenth D5, quarter B4, half B4, quarter F#5, dotted eighth D#5, etc. The *implicit* definition is of a set of note-classes with class membership defined by spelling, register, and duration. The example might be more useful from the music-theoretic viewpoint, and perhaps more interesting as a program, if one property of each note, e.g., pitch class, had been extracted from the code.

The authors' efforts to make abstract concepts accessible to the inexperienced are occasionally misleading. After correctly

---

[5] See Daniel Starr, "Sets, Invariance and Partitions," *Journal of Music Theory* 22/1 (1978):1-42.

defining *pitch class* and modulo 12 arithmetic operations, they illustrate with the following (page 33):

> Finally, transposition can be applied to an inverted interval or note name. In the mod 12 system, a C major triad can be inverted to produce the F minor triad and be transposed up a major third to produce the A minor triad:
>
> C major triad: 0 4 7
> Invert mod 12: 0 8 5
> Transpose by 4: 4 0 9
>
> The following musical notation will help make transposed inversion easier to visualize:
>
> 
>
>
> Remember that inversion produces the *contour mirror* of the starting interval or chord, so that C E G is mirrored by C A- F, the latter of which is transposed to E C A in this example. [Italics mine]

While explanations such as this are obviously meant to help the novice understand theoretical concepts, the association of *contour* with *pitch class* may well have the opposite effect.

The fact that the book is geared toward three popular microcomputers makes it immediately useful to a large number of users. Users of other machines and other BASIC interpreters will still find the contents informative, although they will have to spend more time with technical manuals for their environment. However, targeting specific computers may make the book prone to obsolescence as the computers for which it was written become outmoded.

Finally, there is a great deal of literature on microcomputers, data structures, computer assisted instruction, and music theory. It would be helpful for many readers if more suggested readings and references were cited where appropriate. Suggestions such as "Our procedure is but one of a number of ways to find prime form, and we suggest that you check the recent literature on music theory for other possibilities" (page 61) will not be helpful to a large segment of the intended audience.

The minor criticism in the previous paragraphs should not detract from the utility of this book, and may result from the terseness of the text, which packs a great deal of useful information into about 300 pages. The authors have managed to express difficult concepts in language that is almost always clear and concise.

Microcomputers and Music is an excellent introduction to good programming style, using a language for which style and program structure have often been secondary considerations. In the preface, the authors defend BASIC on the grounds that it is widely available and many people already know how to use the language. They make the statement, "So long as programs are logically designed and well-structured, then the choice of a language becomes much less of an issue." In my classes in computer applications to music research, using Pascal and C, I have found that students who have learned programming in unstructured languages such as BASIC

sometimes have more difficulty developing good programming style than students with no experience at all. However, I suspect that students who learn to program in BASIC using programming techniques such as those described in this book will find the transition to more powerful languages quite natural, and will discover that the new control statements, named subroutines, and user-defined data types allow them better to express the program structures that they are already using.

Although this text is an excellent introduction to microcomputer applications in music, it must be considered in this light. The emphasis on graphics and simple sound production will not be particularly useful to serious researchers in music theory, and those who wish to write CAI applications for training professional musicians will need to master more sophisticated tone generators than those built into most popular microcomputers. While the emphasis on programming style and program structure are admirable in a text on programming in BASIC, the language, at least in the form presented in this text, is not an adequate medium for developing large systems and powerful programming tools. These applications are more easily constructed using languages that provide more flexible control structures, named subroutines (with symbolic arguments), separate compilation of modules into libraries that can be linked with other programs, user-defined data types, and other amenities of more powerful programming languages.