Book Reviews

Alexander R. Brinkman. *Pascal Programming for Music Research.* Chicago: University of Chicago Press, 1990.

Reviewed by Eric J. Isaacson[1]

The computer has long been embraced by musicians and music researchers as a tool capable of providing valuable assistance in the collection and analysis of large amounts of data, of testing and developing musical theory, of providing instruction of various types, of composing and performing. The historical accounts of Harry Lincoln and Bo Alphonce provide useful overviews of history of computer-assisted music research.[2] Current computer-based research is reported

---

[1]UNIX is a registered trademark of AT&T. Macintosh® is a registered trademark of Apple Computer, Inc. VAX is a registered trademark of Digital Equipment Corporation. PS/2 is a registered trademark of IBM Corporation. MS-DOS is a registered trademark of Microsoft Corporation. Turbo Pascal is a registered trademark of Borland International.

[2]Harry B. Lincoln, "Uses of the Computer in Music Composition and Research," in *Advances in Computers*, vol. 12, edited by Morris Rubinoff (New York: Academic Press, 1972), 73-114. Bo Alphonce, "Computer Applications in Music Research: A Retrospective," *Computers in Music Research* 1 (1989): 1-74.

annually by Hewlitt and Selfeidge-Field.[3] The bibliographies of Deta Davis, Alphonce, and Gary Wittlich are also important references.[4] As these summaries demonstrate, the range of musical problems for which the computer has proved useful is vast, and is likely to expand further.

In computer-aided research, one fact is virtually inescapable: with few exceptions, the researcher must be able to write computer programs—algorithms that instruct the computer to manipulate musical data toward the desired end. For the scholar first coming to this realization, or for the graduate student entering the field, Alexander Brinkman's book, *Pascal Programming for Music Research*, will be an important resource. On the other hand, music scholars experienced in computer programming could often benefit from sharing ideas on how to accomplish fundamental tasks. Typically, researchers doing some form of computer programming (including this one) have to devise their own methods for representing and manipulating musical data—a case of the continuous reinvention of the wheel. Basic tasks, such as calculating the prime form of a pitch-class set to take just one example, have probably been implemented dozens of times by dozens of researchers in dozens of different ways. The time required to solve more complex problems, such as devising a representation of a full musical score that can be examined in many different ways by the computer, can be depressing. Brinkman's book will also be an important resource for this group. One of the chief merits—though only a secondary purpose—of *Pascal Programming for Music Research* is that it should help break the cycle of redundant development of basic tools. It raises the common denominator by providing in one place a

---

[3]Walter B. Hewlett and Eleanor Selfridge-Field, *Computing in Musicology* (Menlo Park, CA: Center for Computer Assisted Research in the Humanities, annually since 1985). This series was originally published as *Directory of Computer Assisted Research in Musicology*. Volume 7 is scheduled for 1991.

[4]Deta S. Davis, *Computer Applications in Music: A Bibliography* (Madison: A-R Editions, 1988). Bo H. Alphonce, "Computer Applications: Analysis and Modeling," *Music Theory Spectrum* 11 (1989): 49-59. Gary E. Wittlich, "Computer Applications: Pedagogy," *Music Theory Spectrum* 11 (1989): 60-65.

library of general-use algorithms for music study, a library that will save the researcher from having to craft what are merely tools whose purpose is to achieve a higher goal. But *Pascal Programming for Music Research* not only provides the library, it tells how to construct the library.

Brinkman's choice of the Pascal programming language was prudent. BASIC, though widely available on early microcomputers and still supplied with most IBM-compatible computers, is limited in power and, despite the fact that it is relatively easy to learn, is not well suited to the development of large, complicated applications. The other language Brinkman might have used is C. C is perhaps more widely used than Pascal in research applications and is viewed by its users as the more powerful of the two, but it is also more difficult to learn. Nicklaus Wirth created Pascal specifically with the teaching of computer programming in mind. Its emphasis on structured programming makes it a good pedagogical language. Mastery of Pascal would make easier mastery of C, yet Pascal is powerful enough to create complex applications and is not at all inadequate as a tool for serious research.

Brinkman's weighty contribution—the book numbers nearly 1000 pages—contains something for nearly everyone involved or interested in computer programming. For the programming novice it begins with several chapters which introduce the Pascal programming language. Those already familiar with Pascal programming can jump in later where Brinkman describes methods for representing musical data, for generating a representation of the musical score inside the computer, and for analyzing a score based on this representation. The book is targeted primarily at "professionals and students in music theory, musicology, composition, and education" (xv)—probably in that order, since Brinkman's own background is primarily analytical, and pedagogical applications that can be generated from the material in the book are rather limited. In fact, readers unfamiliar with set theory and serial techniques will likely be at somewhat of a loss when these topics are addressed. Brinkman suggests the book "can be used as a textbook for the classroom or self-instruction, as a reference manual, or as a 'cookbook' of ideas and techniques" (xv). The book is unfortunately

less than ideal as a vehicle for self-instruction; it will be more successful, in this reviewer's opinion, as a textbook for a course or as a "cookbook" for those with previous programming experience.

Pascal Programming is divided into three parts: "Getting Started," "Structured Types," and "Applications." The first part provides a beginner's introduction to programming in general and programming in the Pascal language in particular.[5] Chapter 1 ("Introduction") provides an overview of computer hardware and the low-level software that make computers work. Its topics include binary numbers, the interaction of the central processing unit, main and secondary memory, input/output devices, machine and assembly languages, the program compilation process, and operating systems. The introduction to the inner workings of the computer is useful, although Brinkman jumps quickly into fairly heavy algorithmic thinking which may prove somewhat daunting to the novice. The effectiveness of the presentation is also sometimes hindered by the use of terms that are left unexplained until later, such as "instruction set" (used on pp. 3 and 5, but not explained until 7); or are inadequately explained, such as "one's complement" and "two's complement," two methods of representing negative numbers in the computer; or what the term "random" means in "random-access memory" and even a computer "program" (3). On occasion, awkward or imprecise wording gets in the way, like saying the binary values 1 and 0 can be used to represent "the presence or absence of *pixels* [picture elements—the small individual dots that make up the display] in a bit map of the computer

---

[5]The "standard" version of Pascal as adopted by the International Standards Organization (ISO) and the American National Standards Institute (ANSI), and adopted by Brinkman, is presented in "DP 7185.1. Specification for the Computer Programming Language Pascal: Second Draft," *Pascal News* 20 (1980): 1-83. It should be noted that this is the international Pascal standard. An American version, known as ANSI/IEEE 770 X3.97-1983, adopts level 0 of the international version, but omits level 1, which adds conformant arrays to the language definition. (IEEE stands for the Institute of Electrical and Electronics Engineers.) The American standard and its differences from the international standard are presented in Henry Ledgard, *The American Pascal Standard* (New York: Springer-Verlag, 1984).

screen'' (3). The pixels in a computer screen are, in fact, always present. Their binary aspect comes in whether they are on or off. Or, ''The term 'memory' has been used in different meanings in recent usage'' (5). Or ''A program is a sequence of instructions that performs a specific task. In general, the instructions are executed sequentially'' (19). These types of problems occur throughout the book and interfere with its effectiveness to differing degrees. Brinkman's discussion of operating systems (17-18) is terse and full of jargon, as well.

Chapter 2 (''A Tutorial Introduction to Pascal'') surveys the Pascal programming language in an informal, non-rigorous fashion. It presents the fundamental syntactic elements of the Pascal language. The purpose of the chapter is to set the stage for the more in-depth presentation given in succeeding chapters. It should be read with that in mind, since many aspects of the language are only minimally explained as to their purpose or proper use.

As in the first chapter, there are problems of ordering, of terms being used before they are explained, and of terms being used without any explanation. For example, Brinkman discusses variable and type declarations at some length on p. 54, but doesn't explain why they are needed until 56, and never explains what it means to ''declare'' a type or variable. (It is to define, for the compiler, user-defined data types, and to specify the names and types of any variables to be used within a program.) Also, an explanation of the symbol '':='' which is read ''set equal'' (as in *a := 10*, which assigns the value 10 to a variable named *a*) is relegated to a footnote where it is explained simply as Pascal's ''assignment operator,'' a term perhaps unknown to a novice programmer (27, note 6). Nevertheless, the chapter is useful in that it helps establish the flavor of the language and provides a good foundation for the later chapters.

Chapters 3 (''Pascal Basics and Simple Types''), 4 (''Input and Output''), 5 (''Control Statements''), 8 (''Eof, Eoln, and Input↑''), and 9 (''Functions and Procedures'') present more formally the fundamental elements of Pascal: basic program structure, simple data types (integers and real numbers; characters; and Boolean values, TRUE and FALSE), standard Pascal procedures for interacting with the user and with data files stored on external media (such as disk drives),

program control statements, and user-defined functions and procedures. The many programming examples found at the end of each chapter are on musical problems that deal with such ideas as the manipulation of simple representations for pitch or rhythm. The examples serve both to demonstrate effectively the uses of the various language elements and to lay the groundwork for later, more extended examples. As such, they are among the most useful features of the book.

The presentation in these chapters is generally clear and effective, though there are, again, occasional problems in ordering. In chapter 5, for example, Brinkman says the section on the **case** (98) and **for . . . do** (104) statements are optional on first reading. It is hard to imagine why this is so. In the latter case, immediately following presentation of the **for . . . do** statement, Brinkman gives a section on determining when to use which of Pascal's three types of looping construct (**while . . . do**, **repeat . . . until**, and **for . . . do**). Skipping the section on **for . . . do** would make this following section difficult to understand.

In chapter 8 the reader will encounter one of the most frustrating practical problems involved with the text. The final paragraphs of the chapter begin, "Unfortunately, the details of input and output, and particularly testing for end-of-file, are one area in which versions of Pascal sometimes differ" (226). This sentence would better be placed at the beginning of the chapter where it would warn the reader in advance that there might be inconsistencies and that the chapter should be read in combination with a manual for the particular version of Pascal being used. For example, the use of *input*^ to access the input file buffer will be a real problem to users of Borland's Turbo Pascal for MS-DOS computers. TP doesn't support *input*^ and many of Brinkman's early examples, examples which the reader is encouraged to try out, will simply not work in the form Brinkman presents them. While the problem appears as early as chapter 2, Brinkman doesn't mention the problem or a solution until far too late, chapter 8 (224, program 8.7, with the explanation that Turbo Pascal "does not provide access to buffer variables" in note 8), by which time the person who has been trying to run the example programs will be thoroughly confused and frustrated. The problem is also noted in the paragraph heading the "References and Selected Readings" section of chapter 8,

where the reader is (finally) directed to the reference manuals for the Pascal compiler being used to see how input and output are handled.

The source of this problem is that different computers have fundamental differences which require that those aspects of a computer language that must work with the computer hardware or operating system necessarily differ to some extent. It is differences in operating systems that actually account for the most substantial differences in Pascal implementations. The four operating systems with which potential readers of *Pascal Programming* are most likely to work are MS-DOS, used on IBM-PC and compatible computers; the proprietary operating system used on the Macintosh family of computers; Digital Equipment Corporation's VMS, which runs on DEC's VAX computers; and UNIX, which is used on many different types of computer. The first two run on relatively inexpensive microcomputers, while the latter two are more often found on multi-user computers or more costly personal workstations. Brinkman's own work has been primarily on UNIX-based computers (xviii) and this background is often turned into bias in his book. While Brinkman generally avoids platform-specific statements, when he does, they are almost invariably given from the UNIX perspective. For example, Brinkman says end-of-file mark (indicating the user is finished entering data) can be issued from the terminal by typing <control-D> (that is, by holding down the key labeled Control [or CTRL] and pressing D), adding parenthetically that "The end-of-file signal may vary from system to system" (204). On VAX computers running the VMS system and on MS-DOS computers, the key combination <control-Z> transmits the end-of-file mark. In general, statements beginning "On many systems . . ." refer to systems running UNIX. This reviewer suspects that readers of Brinkman's book are at least equally likely to be using microcomputers for their work, especially since they are more congenial to developing applications using music graphics and MIDI-interfaced devices, are more widely available, and often easier to work with without outside assistance. While it would be impractical, and likely impossible, to explain how to test for such simple conditions as the end of input on even the most popular computer systems, I think Brinkman errs in providing examples only for UNIX-based systems.

Chapter 6 (''Encoding Music'') is an interlude of sorts that covers methods of numerically representing pitch and rhythm.[6] A variety of pitch representations are given which preserve in varying combinations octave, pitch-class, and note-name information. Though he begins a discussion of pitch class with a quote from Babbitt (119) which is more confusing than enlightening, the remainder of the chapter is well organized and largely easy to follow. The fullest representation for representing pitch, called *continuous binary representation* (*cbr*), is an especially efficient and effective means of storing and manipulating pitch information in the computer. The term ''binary'' reflects the fact that both pitch-class and note-name information are preserved, while ''continuous'' means that octave information is preserved. *Cbr* uses a four-place integer value in the format *opcn*, where *o* represents the octave (middle C = 4), *pc* is the pitch-class number (C = 0, B = 11), and *n* represents the note name (C = 0, B = 6). F♯4 (octave 4, pitch class 6, name code 3) would be represented as 4063. The representation preserves the precise spelling of the note. Brinkman should be a little clearer about the fact that the octave number refers to the note name rather than the pitch class. So 3110 is the *cbr* representation of C♭3 and is enharmonic with 2116 (B♯2).

During the discussion in chapter 6, Brinkman appears to introduce mathematical rigor for rigor's sake. A section examining extensively the mathematical ''Properties of the Binomial System'' (131-32) seems only to fulfill some desire for mathematical exactitude that is out of place here. One example is the statement that the first seven of eight properties listed ''are sufficient to show that the binomial system is a *commutative ring with unity*'' (132). And the purpose of the eighth property, scalar multiplication, is not at all clear: ''We define scalar multiplication such that, for any positive integer *n* and any binomial pc $<a,b>$:

$$n \times <a,b> = <(n \times a) \bmod 12,(n \times b) \bmod 7>$$

[6]Much of the material in the first part of the chapter was published earlier in Alexander R. Brinkman, ''A Binomial Representation of Pitch for Computer Processing of Musical Data,'' *Music Theory Spectrum* 8 (1986): 44-57.

While its inclusion may be appropriate for a journal presentation, it is a distraction in what is intended to be a practical manual.

Chapter 6 also summarizes the DARMS and SCORE music encoding languages. DARMS, developed in the early 1960s by Stefan Bauer-Mengelberg, was originally intended to facilitate music printing. Leland Smith's SCORE language has been used as an aid in both music synthesis and music printing. Some comments on Brinkman's description of DARMS code: In a section in chapter 6 introducing DARMS code, the sentence, "Another shortcut is that either the space code or the rhythm code may be omitted if it is identical to the previous note" (138) comes before any mention of rhythm code. Brinkman restates this on p. 142, a more logical place. Later, he introduces the terms "delta suppression" and "sigma suppression" to refer to this omission of *d*uration (delta) or *s*pace (sigma) code upon their repetition (144). Since these terms are not used elsewhere, they are, in this context, simply unnecessary jargon. From Table 6.8 (144) it is not clear whether the DARMS code for the staccato mark is the comma or the apostrophe. It is the latter. And Brinkman says "The exclamation point functions as an escape character in DARMS" (371) without explaining what an "escape character" is.

In several of the musical examples given with DARMS code, there are errors either in the musical notation or in the DARMS code: (1) Figure 6.20 encodes part of the Mozart Quartet No. 19, K. 465, III, but there is no comment that the dynamics and label "Trio" are omitted. (2) Figure 6.21 (147), the first 6 measures of the *Requiem per flauto solo* of Kazuo Fukushima, contains two DARMS errors. The last note of the penultimate measure, C7, is encoded with the space code 38 when it should be 40; and the first note of the last measure, given as A♮ in notation, is encoded in DARMS as A♭ (4-). (3) The second example in the middle of page 149 is missing a tenuto mark. (4) Figure 6.25 (151), Bach, *Nun komm' der Heiden Heiland*, BWV 599, mm. 1-2, contains two encoding errors: groups of sixteenth notes in the right hand, m. 1, beat 3 and m. 2, beat 1, are encoded as eighth notes. (5) In chapter 13, the comments for the grouplet definitions in the DARMS code input to program 13.2 (449) contain typographical errors.

In chapter 7 (''Program Design''), Brinkman suggests a strategy for developing an algorithm and realizing it in a working Pascal program. This is among the most important of the early chapters because, in it, Brinkman guides the reader through the application-development process by producing a pair of program groups, giving step-by-step instructions on how one might move from the analysis of a problem to the full implementation of a solution.

Again, occasional lack of clarity reduces the effectiveness of presentation. In a sample of program verification, by the sentence ''This variable is initialized to 0 outside the loop so the expression (*measures* + *1*) can be evaluated'' (172), Brinkman really wants to convey (1) that it is *in general* necessary to initialize the control variable before entering a loop which tests its value, and (2) that in *this* case, 0 is the desired initial value.

In part II, Brinkman presents more complex data structures: arrays (chapters 10 and 11), sets (chapter 12), records (chapter 13), and external files (chapter 14). In addition, he presents specialized extensions of Pascal's user-defined functions and procedures (chapter 15, ''Recursive Algorithms'') and records (chapter 16, ''Linked Data Structures''), the latter making use of dynamically created data structures. The more complex data structures allow data to be combined in meaningful ways and are the foundation of the larger applications presented in part III of the book. Although many of the same types of problem found in the first part of the book occur in the second as well, no examples will be given here.

In the final part of his book, ''Applications,'' Brinkman develops several substantial applications, some of which are the culmination of programs of smaller scale developed in the earlier chapters of the book. In the first (chapter 17, ''Prime-Form Algorithms''), Brinkman presents several methods for calculating the prime form of a pitch-class set. The algorithms, by Starr, Forte, Alphonce, and Rahn, differ in their approaches to the problem and exemplify well the trade-offs between speed and memory requirements which sometimes influence decisions in algorithm implementation. (One typographical error requires correction: the description of the final step in calculating Rahn's normal order of a set should read, ''The pitch class in the first column of the

first row is added to each interval in the first *row* [sted column], resulting in the normal order of the original set," 666).

The second application chapter ("A Matrix-Searching Program," chapter 18) presents a program to locate occurrences of pcsets in a serial matrix. While the search algorithm itself is general, Brinkman introduces a hardware-specific (for DEC's VT100/VT200 series of terminals) method of displaying the results on the screen. Those using Turbo Pascal for MS-DOS computers should know that the control sequences described at the beginning of the chapter (in table 18.1) for controlling the movement of the cursor on DEC VT100/VT200 terminals and compatibles all have equivalent procedures in Turbo Pascal.

Chapter 19 ("Spelling Pitch Structures") presents a simple instructional application which drills the spelling of intervals, scales, and chords. The program, although unoriginal pedagogically, is a useful model for those wishing to develop instructional software. Because the use of music graphics and sound generation are highly dependent on both the hardware and software being used for development, these issues are ignored entirely, leaving the readers to work them out on their own.

The final chapter (chapter 20) presents the book's most ambitious application. Entitled "Score Processing," Brinkman presents in it a means of converting a score encoded in DARMS into a Pascal data structure in such a way that the score can be relatively easily analyzed in a variety of ways.[7] The score structure was designed to meet three important criteria (751): *function*—the representation makes it possible to search the score either vertically or horizontally, with segmentation according to various musical parameters possible; *detail*—the representation preserves practically all the detail in the notated score, from pitch and rhythm to articulation and dynamics; and *extensibility*—the representation can be expanded to accommodate future needs. All three criteria are more than adequately met. Although the DARMS

[7]Some of the material in chapter 20 was first published in Brinkman's "Representing Musical Scores for Computer Analysis," *Journal of Music Theory* 20 (1986): 225-75.

interpreter is not fully implemented in the book,[8] it is complete enough to understand the approach being taken and to be quite functional.

In addition to the DARMS interpreter, chapter 20 also presents several programs which use the resulting score structure to display score information to verify its proper encoding; to find recurring melodic contours; to determine the prime form of vertical time slices; to segment the score according to rests, slurs, and other means; to locate instances of a given pitch-class set in the score; and to locate "cadences in Bach chorales and to draw conclusions regarding their tonal implications" (806). A final example produces a graphical display of the voice leading of a piece, with time along the horizontal axis and pitch along the vertical axis. Interesting extensions of this idea were presented by Brinkman and Martha R. Mesiti at a poster session of the 1991 meeting of the Society for Music Theory in Cincinnati.

Especially helpful in association with the last chapter are complete listings—in the form of three appendices—of the program code of the DARMS interpreter, a library of procedures to create and examine the score structure, and the score manipulation programs described in the second part of chapter 20. As an added bonus, each appendix contains an alphabetical index to the functions and procedures contained therein. Even more helpful would have been if the author had indicated a willingness to make the code available electronically or otherwise, since the three appendices take up nearly 90 pages of two-columned program code.

Pascal Programming fulfills many of its aims admirably. The order in which topics are presented is logical. The reader who has followed the material of the first two parts of the book will be well equipped to understand the application chapters in the last part of the book and to generalize from them in the design and implementation of original applications.

---

[8]Indeed, sentences such as, "As of this writing, the parenthesized [DARMS] attributes have not been implemented in the software described later in this text" (142-43) give the impression that there was a deadline to meet and that we are not getting a complete product.

From the start, Brinkman stresses structured programming, a programming philosophy in which the most general outlines of the solution to a problem are developed first as a sequence of broad steps. Then, through a process called "stepwise refinement," each of these steps is broken down into sub-tasks, each of which is defined separately. This process continues until each subroutine (user-defined function or procedure) carries out but a single task. This approach, which is also known as "top-down design," has many advantages. It is easier, for one, to ensure that each routine works correctly if it is asked only to perform a single, relatively limited, task. An emphasis on top-down design is a minimum condition for any book purporting to teach a programming language, but it is a condition which Brinkman's text meets quite satisfactorily.

Throughout the book, Brinkman provides ample programming examples which are used to illustrate not only some aspect of the Pascal language, but to show how the concept under consideration might be applied to a musical context. This is one of the very positive features of the book. Programming languages are often learned in courses that stress applications in business or science. The music researcher must then try to translate techniques learned in that context to music, where many problems are of a very different nature and generalizations from non-musical circumstances do not always come easily.

As the book progresses, many of the examples build on prior examples, or at least use much of the same program code. To the credit of both Brinkman and the University of Chicago Press, Brinkman repeats a code each time it is used in a new context, even though it has previously appeared. While this leads to a certain amount of redundancy, the space is not wasted. The duplication puts the code where it is needed, right within the text where it is being used. (Some of the repeated procedure or function definitions are modified upon repetition to reflect the differing requirements of different contexts.) That much of this code is yet again duplicated in the formal program listings also makes convenient the testing of the program by the reader, since these program listings put all the code together in one place. By Part III, "Applications," the size of the programs being developed makes such duplication impractical, and the codes of many of the

programs are simply listed in order, with annotations before each section of the code, a solution which works well.

Throughout the various programming examples, Brinkman strives for, and for the most part realizes, a consistent programming style. To achieve complete consistency would undoubtedly have been an immense task, since the book brings together a collection of programs developed separately over a period of several years. As a result, some programs found in the book differ in the way they are laid out (with respect to indentation and other visual formatting) or in the names of variables and constants. To give but one example, in a series of programs given in chapter 7, a constant called *doublebar* is used to define the code used to signal the final barline in the program input. But program 7.6 uses a constant of a different name, *fine*, for this exact purpose. Such inconsistencies, where they occur, are minor, but they are distracting and could prove somewhat confusing to the novice who may not understand why, as in the above case, a different identifier is being used in just one of several otherwise consistent instances. The extraction of programming examples in earlier chapters from the later, larger applications also causes a few problems. On p. 370, for example, Brinkman includes some code in the text clearly taken from another, larger application. The programming example contains some calls to procedures (*error*, for example) whose purpose is not explicitly stated. Also, as he starts using excerpts in chapter 11 from the DARMS program presented in its entirety in chapter 20, functions and procedures are invoked without explanations as to their purpose, global variables are referenced but not explained locally, and procedure and function declarations begin to appear with an *external* directive, which is also not explained until chapter 13 (447, n. 5). (Nor is the *external* directive part of Pascal, so the user trying to test the programs as given needs to go to the library of procedures given in appendix D, though, again, this is not noted at the point in the text where the information is needed.)

Although Brinkman says most of the program code was taken directly from working versions of the program, thereby minimizing errors that are prone to appear in books of this type, errors do indeed appear. Most of these errors appear in code, given in the text, which

is not actually part of a programming example, and result from simple typographical errors. Some selected examples:

- In a program given at the top of p. 210, the variable *done* is not declared. The program will not work without the declaration.
- In the text on p. 225, there is an extraneous reference to a variable *c* which does not occur in the referenced sample statement; the statement Brinkman refers to is *write('x')*, but should have been given as *write(c)*.
- Reference to a variable *setnum* in the text when in the program it is *bitvec* (681).
- In a sample program, middle of p. 289, the arguments in a call to a procedure called *incr* should be given in parentheses instead of square brackets (*incr[p]* should be *incr(p)*).
- In a procedure called *delete*, which deletes an element from a linked list (574), the second **if** statement reads **if** *q* := *q*^.*link*, but should read **if** *q* = *q*^.*link* (it is a test of equality, not an assignment statement).

Other errors involve mistakes in programming logic. The most serious of these occurs in Brinkman's *quicksort* procedure (540-41), which sorts a list of numbers stored in an array. As presented, Brinkman's procedure will not work with certain lists of numbers. The flaw in the algorithm will result in the reference of array elements in the list beyond the bounds of the array. If range-checking is not done by the compiler, this is unlikely to result in an error, but the success of the algorithm depends on the loop finding a random integer value beyond the list of numbers greater or less than the current number. Brinkman's version is given in Figure 1 and a corrected version in Figure 2 (the second version conforms to my own programming style, so indentation and capitalization differ somewhat).

The identifiers (type and variable names) Brinkman uses in his examples could sometimes be better. For example, procedure *durcode* (part of program 13.2, declared on 460) includes a parameter *x* which is used to store the duration of a note. A variable name *dur* would

certainly be more suitable. In *markpath* (part of program 15.5, 537) he uses *i* and *j* to refer to the maze's row and column, where variables called *row* and *column* would be ideal. And in chapter 17 ("Prime-Form Algorithms") Brinkman starts using non-mnemonic data types (*ar100, ar300*, for example), giving as his reason: "Most of the array types are not named mnemonically since some are used for more than one purpose" (635). In each of these cases, Brinkman undermines the previous careful control of identifier names and suggests to the reader that the convention of using mnemonic names has been pedagogically and not practically motivated. If otherwise identical data types will be needed for two different purposes, go ahead and declare two different data types, one for each purpose. There is no penalty in program efficiency and the benefits in program readability are worth the slightly extra programming.

In addition to providing detailed programming examples, Brinkman gives sample output for most programs as well. This is extremely valuable for the reader, since it provides the opportunity both

```
procedure quicksort(var A : list; m, n : integer);
  var
    i, j : integer;
    k    : integer;
  begin  { quicksort }
    if m < n
      then
        begin
          i := m;
          j := n + 1;
          k := A[m];
          repeat
            repeat
              incr(i)
            until A[i] >= k;
            repeat
              decr(j)
            until A[j] <= k;
            if i < j
              then swap(A[i], A[j]);
          until i >= j;
          swap(A[m], A[j]);
          quicksort(A, m, j - 1);
          quicksort(A, j + 1, n)
        end
      end;
```

Figure 1. Brinkman's quicksort procedure (540-41).

```
Procedure Quicksort(var a: list; Lo,Hi: integer);
  Procedure Sort(l,r: integer);
    Var
      i,j,x,y: Integer;
    Begin
    i:=l; j:=r; x:=a[(l+r) DIV 2];
    repeat
      while a[i]<x do i:=i+1;
      while x<a[j] do j:=j-1;
      if i<=j then
        begin
        y:=a[i]; a[i]:=a[j]; a[j]:=y;
        i:=i+1; j:=j-1;
        end;
    until i>j;
    if l<j then sort(l,j);
    if i<r then sort(i,r);
    end;

  Begin {Quicksort};
  Sort(Lo,Hi);
  End;
```

Figure 2. Corrected quicksort procedure

to follow the computer program and to understand its operation by seeing it actually work, without having to enter the program on the computer immediately. In a few instances, sample program output has been edited without an indication to that effect (in the output of Program 2.1 (29), for example). In the case of program 8.6, which produces one value per line, that the output has been edited is noted in a footnote, but *how* the output has been edited is not explained. The output should be read in columns (top to bottom), then left to right (rather than in rows, left to right, then top to bottom).

There are also some minor errors in Brinkman's description of

the Pascal language itself. He erroneously refers to pointers as structured types (Table 2.3, and elsewhere). A pointer contains the address of the memory location of some Pascal object, usually a variable. Structured data types often include pointers and pointers are often used to reference structured types, but they are not themselves structured types. Brinkman also includes linked structures in a list of "structured types" (the others of which are arrays, sets, records, and files) (43). While Pascal provides the means to create linked structures (through the *pointer* type), it does not provide linked structures themselves. On p. 603 Brinkman uses a type *long* in a sample program which is not standard and cannot be user-defined. Though common, it is not standard Pascal and, given Brinkman's previous strictness in using non-standard features, this is out of character. Also, the fact that he does not explain that it is non-standard is problematic. (He does this for an algorithm requiring a maximum value of 823,542, a value larger than the standard **integer** type is usually able to store.)

Determining the level of background of a book's readership is a problem with any text. Brinkman assumes his reader is familiar with relatively standard math symbols and notation. While little of the math is that difficult, it would have been a relatively simple matter to explain a few symbols here and there for the benefit of readers for whom math may have become a little rusty. To give a few examples, in a section defining numeric literals, Brinkman gives an example which includes Pascal's e-notation (e.g., -6.2123e+17), used to represent exponentiation, without explaining what the notation represents (the example means -6.2123 times 10 to the 17th power). In his Figure 11.3 the symbols "::=" (meaning "is" defined "as") and $\in$ (meaning "is a member of") are not interpreted. And in a program to calculate the prime form of a set (chapter 17), if the user enters a character that cannot be interpreted as a pitch class, Brinkman's suggested error message is: "Error: pc = 0|1|2...." That all users (or readers) would know that the symbol "|" means "or" should not be assumed.

More serious are expectations that the audience knows more about computers than should be assumed. For example, the opening comments for program 15.2 (513) use Backus-Naur notation to describe

legal program input, but the notation is not explained. And occasional references to memory allocation within the computer, such as "Recursion is possible because Pascal allocates space for local variables and subroutine parameters dynamically whenever a function or procedure is called" (500), or, in describing records with variant parts, saying that "each variant is actually the same size" (468) by which he means that each variant is allocated the same amount of memory in the computer, are likely to be confusing to someone not already familiar with some of the inner workings of the computer.

In several places throughout the book Brinkman suggests running programs with sample input read from an external file, but he never tells how this might be done. To do so, of course, would be difficult because the method differs from one computer to another, but it may leave the reader working alone with many questions. Helpful, if perhaps impractical, would have been for Brinkman to be more explicit about the software *he* was working with and provide possible workarounds for major alternative implementations.

The reading lists found at the end of each chapter provide valuable references to a wide range of additional sources. Helpful annotations will in most instances direct the reader to books or articles that provide general background on computing and music, expand on topics presented in the chapter, or fill in gaps in the reader's knowledge. There are some curiosities, however, particularly in the "Other Suggested Readings" sections. Brinkman seems to have taken elements of a general bibliography on computers and music and sprinkled them throughout the various chapters, for it is not always clear what the relation of a particular entry is to the topic of the chapter. In his preface, Brinkman says these sections include "representative articles from the field of computer-assisted music research and related fields [and] are included to familiarize the reader with the primary literature" (xviii). Nevertheless, many of the readings given are only marginally computer-oriented. And to suggest the reading of entire books without annotation as to the purpose of the book, an indication of sections relevant for study, or without an explanation of how the book connects with the contents of the chapter just read, provides little benefit to the reader. To cite but two of

several such examples, among the entries in the list at the end of chapter 5 which describes Pascal's control statements—those statements in the language that permit the repetition of a statement or series of statements—are David Lewin's "Intervallic Relations Between Two Collections of Notes," and "The Structure of All-Interval Series," by Robert Morris and Daniel Starr, as well as Fred T. Hofstetter's *Computer Literacy for Musicians*.[9] Why these references are given here as opposed to at the end of another chapter is not explained. The reference list at the end of chapter 8 on file input and output includes David Lewin's, "A Theory of Segmental Association in Twelve-Tone Music," and Robert Morris's, "A Similarity Index for Pitch-Class Sets," and the reference list at the end of chapter 10, "The Array and List Processing," includes Lewin's *Generalized Musical Intervals and Transformations*, and the whole book at that![10] In some instances, more explicit references are needed. For example, Allen Forte's *The Structure of Atonal Music* is cited as a reference for Forte's set names (362), but Brinkman does not give the page numbers in which the names are explained, or the appendix in the book that lists the set types.

There are also times when Brinkman seems to be trying to give the book a scholarly bent. Of the bibliographies and summaries of the state of music theory, contained in *Music Theory Spectrum* 11/1, which collectively cover an exceptionally wide and diverse set of topics,[11] Brinkman inflates the importance of computers in music research in saying that *all* the bibliographies "are relevant to the subject of this

---

[9]David Lewin, "Intervallic Relations Between Two Collections of Notes," *Journal of Music Theory* 3 (1959): 298-301. Robert Morris and Daniel Starr, "The Structure of All-Interval Series," *Journal of Music Theory* 18 (1974): 364-89. Fred T. Hofstetter, *Computer Literacy for Musicians* (Englewood Cliffs, NJ: Prentice-Hall, 1988).

[10]David Lewin, *Generalized Musical Intervals and Transformations* (New Haven: Yale University Press, 1987).

[11]Including the areas of Schenkerian theory, analytic approaches to nineteenth-century and twentieth-century musics (including twelve-tone and atonal repertoires), the history of theory from the sixteenth through the nineteenth century, music theory pedagogy, interdisciplinary approaches, and others.

book'' (20), before singling out those by Bo H. Alphonce, ''Computer Applications: Analysis and Modeling'' and Gary E. Wittlich, ''Computer Applications: Pedagogy,'' which are clearly and directly relevant. If computer-aided research is truly applicable in each of the areas summarized by the bibliographies, such an all-encompassing statement becomes virtually meaningless.

At the end of each chapter is a series of (mainly) programming exercises based on the techniques presented in the chapter. Most of the exercises focus on problems related to music processing and are thus of immediate interest to the reader. The exercises are consistently relevant, well-conceived, and thorough. Many, in fact, provide extensions of material presented in the chapters and thus amplify the subject matter, providing additional insights into possible applications and approaches to solving various musical problems. These sections are excellent resources for the instructor using the book as a textbook for a course in computers and music. The reader who is working through the book independently will be disadvantaged, however, because Brinkman provides no solutions to the exercises. Some exercises are quite extensive and their possible solutions numerous, so providing sample solutions would be impractical. Yet the answers to many exercises, particularly those in the early chapters, are either right or wrong answers, and the reader would greatly benefit from the feedback that would be provided by a set of answers in the back of the book, in the manner of traditional math textbooks, for example. As it is, the reader cannot easily determine whether or not the material has been properly understood.

Overall, *Pascal Programming for Music Research* is to be highly recommended. Its many minor faults by no means undermine the fundamental strengths of the book. The book should remain an important practical contribution to contemporary music scholarship. Hopefully, its publication will inspire researchers to consider new, thoughtful approaches to computer-aided music research and analysis. A host of recent publications related to computing in music, including Todd and Loy's recent compilation of chapters, some of which reprint articles originally appearing in *Computer Music Journal* and David Cope's book describing a computer program which can compose music

in imitation of other composers,[12] testify that movement in this direction has already begun.

[12]Peter M. Todd and D. Gareth Loy, ed., *Music and Connectionism* (Cambridge, MA: MIT Press, 1991). David Cope, *Computers and Musical Style*, The Computer Music and Digital Audio Series 6 (Madison: A-R Editions, 1991).