The BYTE Book of Pascal

Edited by Blaise Liffick

The BYTE Book of Pascal

Edited by
Blaise W. Liffick

Second Edition

BYTE/McGraw Hill
70 Main St
Peterborough, N.H. 03458

Library of Congress Cataloging in Publication Data

Main entry under title:

The BYTE Book of Pascal.

(Language series)
A collection of articles from BYTE Magazine.
1. Pascal (computer program language) I. Liffick, Blaise W. II. BYTE. III. Series: Language series (Peterborough, N. H.)
QA76.73.P2B18 001.6'424 79-22958
ISBN 0-07-037823-1

Printed in the United States of America

INTRODUCTION

This book is part of the "Language Series" of BYTE Books. It is a collection of the best articles from past issues of BYTE magazine, the leading technical journal in the microcomputer field. The language under discussion is a relatively new computer programming language, Pascal. Until recently, Pascal has only enjoyed a large following in the academic community, and only more recently has it been practical to use this language with microcomputers. But the curious thing about Pascal is its ability to win nearly instant converts; so, while Pascal may be one of the newest computer languages, especially in the field of microcomputers, it is also one of the fastest growing in use and acceptance.

The purpose of this book is twofold. First, for those uninitiated, the articles contained in this book can serve as a general introduction to Pascal, providing the background information necessary for a potential user. The **Comments** section itself is a general discussion of the properties, merits, and applicability of Pascal. It includes reprints from the "Languages Forum" of BYTE magazine, an ongoing dialogue among the magazine's knowledgeable readers. The Forum is intended as an interactive dialogue about the design and implementation of languages for personal computing. In addition, an editorial by Carl Helmers, one of the industry's leading proponents of Pascal, rounds this section out as a beginning point for those unfamiliar with the language.

Second, for those requiring a more in depth study of the merits of the language and its possible implementation, there are two sections, **About the Language** and **Applications**.

About the Language provides insights into the usefulness of Pascal by comparing it to BASIC and COBOL. Also, a detailed look at some possible implementations of the language helps define the scope of the impact on the industry. This includes listings of a Pascal to p-code compiler written in both Pascal and BASIC, and two listings in the appendices: one a p-code to 8080 assembly language conversion program in BASIC; the second a "tiny" Pascal compiler and p-code interpreter written in 8080 assembly language.

The final section is **Applications** and, as the name implies, includes several application and system programs written in Pascal. For general applications there is an automatic metric conversion program, nontrivial implementation of a chess program, and an implementation of a print utility program. In the area of system software there is the choice of two language implementations: one is a minimum implementation of a language, written in less than 256 words (it has surprising usefulness); the other is an APL interpreter.

So, this book provides not only a general introduction to the Pascal language, but is also a tremendous resource for software: two versions of a Pascal compiler, one written in BASIC and the other in 8080 assembly language; a p-code interpreter written in both Pascal and 8080 asembly language; a chess playing program; and an APL interpreter.

Finally, a note about how the articles in this book were updated. We have been very careful to make corrections to articles where an error has been made in the original article. However, because many of these articles are reprinted from back issues of BYTE magazine, some of the information contained in them is out of date. This information is flagged in the form of footnotes within the article, and includes such items as page references and the availablity of UCSD Pascal. All footnotes throughout this book can be taken as current as of 1 July 1979.

Blaise W. Liffick
Editor

Consistency — or a Lack Thereof . . .

Notes by C Helmers

Readers will note a lack of consistency in the typography of various articles on Pascal.

One area of questionable typography is a bit nebulous and less subject to editorial fiat when "camera ready" type is received from authors: the style of representation of Pascal program listings. The ideal style is of course that used by Niklaus Wirth in his book Algorithms + Data Structures = Programs, *published by Prentice-Hall in 1976. This style uses* **bold face type** *in lowercase for representation of the Pascal language keywords. It uses italics for the representation of specific variable names, procedure names and literal values which are part of the program. In articles by authors Ken Bowles (page 51), Charles Forsyth and Randall Howard (page 33), and Allan Schwartz (page 41) this notation was used. But in two of these cases, the authors supplied camera ready typeset copy along with the articles involved, in order to minimize potential errors due to keystroking. Since two of these were typeset at BYTE, and the other two were typeset with different type specifications on different machines, there is naturally a different aesthetic flavor to the listings in these articles. A close variant of this form is seen in the listings of David Mundie's article (page 7) where bold fact type and normal type are mixed in the listing.*

There is yet another variation on the graphics used to represent Pascal programs, provided by the listings accompanying Stephen Alpert's article (page 27). Here, the camera ready listing was supplied by the author as printed on an uppercase line printer, so keywords are indistinguishable from program details on the basis of typography alone.

*What can we conclude about this inconsistency? Our goal at BYTE is to asymptotically approach the notation of Pascal programs in the bold face and italic form whenever we do the actual typesetting of a listing. The italic and the bold face typography provides an excellent contrast to normal type when elements of a program are mentioned within text. But when a manuscript comes with a usable camera ready listing of a Pascal program, such details of aesthetics must take second place to the goal of minimizing errors of transcription: it is far better to use a camera ready image derived from a machine produced listing than to key in a program manually in order to create a typeset form of the listing. . . .***CH**

TABLE OF CONTENTS

APPLICATIONS

APPENDICES

Comments

UCSD PASCAL:

A (Nearly) Machine Independent Software System (for Microcomputers and Minicomputers)

Kenneth L Bowles

Overview

This article describes a complete interactive software system which can operate virtually without change on many different microcomputers and minicomputers. Because the semiconductor industry is evolving new equipment very fast, it is becoming a practical necessity to have machine independent software to prevent rapid obsolescence of large application programs. The software described here has been developed at the University of California San Diego (UCSD), and is available to anyone for a $200 subscription fee. This article presents an appeal to readers of BYTE for help to bring about a true community-wide software system for business, educational and other professional users of small computer systems. Help is needed from the user community, since the manufacturers have so far avoided standardizing software except as regards some aspects of programming languages. For single user microcomputers, it appears to be far more practical to *standardize the entire software system* than the language processor alone!

The Software System

UCSD Pascal is a complete interactive software system for small computers, yet it offers many features normally found only on medium and large scale machines. It is designed to operate, with minimal adaptation, on most microcomputers or minicomputers based on 8 bit bytes or 16 bit words. Supported versions are now available for use on machines based on the Digital Equipment LSI-11 or other PDP-11 processors, and on the 8080 and Z-80 microprocessors. Having first been sent to users in August 1977, the system is in use on approximately 60 mainframes using these processors (as of mid February 1978), and the list of both users and processors has started to grow rapidly. Versions not yet supported by the Project are operating, or nearly operating, on four other processors (General Automation 440, Univac V75, Nanodata QM-1, National Semiconductor PACE). The UCSD Pascal Project is discussing arrangements with various manufacturers whereby supported versions can be released for most other popular microprocessors, and additional inquiries would be welcomed.

The system is written almost entirely in the Pascal programming language, extended for system programming and for disk based interactive applications. Far more than a simple *compiler* for Pascal, it should be viewed as a complete and fully integrated system which is self-maintaining, and generally independent of software from any other source. The system operates in a small pseudomachine (interpreter) which can be written in the native machine language of conventional processors, or can be microprogrammed on machines which provide that capability. The object code processed by the Pascal pseudomachine is compressed relative to conventional object code, and consumes roughly one third to one half as much space as the native object code of most present day processors. A feature to be implemented soon will allow mixing Pascal pseudocode routines, for efficient use of memory, with native code routines, for fast processing.

The system is the product of a growing project team, and is evolving rapidly in an upward compatible way. As of early 1978, the system represents the equivalent of about 15 full-time years of programming and design effort. Major components of the system currently being distributed include the following:

- *Single user operating system.*
- *Pascal Compiler.* Standard Pascal plus extensions for strings, disk files, graphics, system programming (business oriented extensions are planned).
- *Editors.* High performance screen oriented editor for program development and word processing, line oriented editor for hard copy devices.
- *File Manager.* General purpose utility for maintaining a library of disk files (usually floppy disks).
- *Debugger.* Single statement and breakpoint processing, access to program variables.
- *Utilities.* Programs for printing, communicating, accessing disks written under DEC's RT11 system, diagnosing disk faults, desk calculator, etc...
- *BASIC language compiler.* Implemented for those who insist on using BASIC, but may wish to write powerful subroutines in Pascal. (The compiler works, but subroutine binding is not yet ready.)

Major components now operating, but not quite ready for general distribution, include the following:

- *CAI Package.* Adaptation of the major Computer Assisted Instruction package developed at University of California Irvine; includes automated materials for an introductory Pascal programming course.
- *Assemblers.* For the PDP-11, 8080 and Z-80, these are written in Pascal for machine independence, but generate native code for those processors.
- *TREEMETA.* A metacompiler developed at UC Irvine.

The UCSD Pascal Project

The Project is one of the principal activities of the Institute for Information Systems, an embryonic "organized research unit" concerned with interdisciplinary studies, and with related instructional and public service activities. The main objectives of the Project include the following:

- *Machine Independence.* To foster the widespread use of machine independent software systems, particularly for small computers, as a means to avoid software obsolescence. A major premise of the project is that applications software can best be made truly portable by making *the entire operating system and support software* portable to a new processor at the cost of only a small effort (eventually: one to three programmer months; currently: about six months).
- *Pascal.* To promote the widespread use of standard Pascal, and standardized extensions, as (the basis of) a general purpose programming language, both for writing system programs such as operating systems and compilers, and for applications software in education, research and business data processing.
- *Software Exchange.* To foster the development of a national or international marketplace within which authors of computer based course materials, and other applications software, may receive reasonable *royalties* to compensate them for their work. As an initial step, the Project will operate a Software/Courseware Exchange, using Tele-Mail techniques, for users of the UCSD Pascal Software System.
- *Mass Education.* To demonstrate that it is practical to improve the quality of mass education at the college level (and adult training in technical topics), while simultaneously reducing costs, through the use of microcomputer based course materials.
- *Research and Development.* To provide facilities, a team working environment above critical size, and salary support for students and faculty members who wish to conduct research or development projects in software engineering and many related fields of study.

Hardware Configuration

The UCSD Pascal system has been designed to run as a single user interactive system with superior response characteristics when one or more floppy disks are used for secondary storage. Wherever possible, single character commands are used, and prompting messages remind the user of the significance of the various commands that are available in different contexts. While the system has proven that machine independence of a complex software system is practical, there are of course practical limits to the range of characteristics that can be accommodated on the host machine. The major characteristics of a typical system needed to run UCSD

Pascal include the following:

- *Main memory.* 56 K bytes (48 K will do, but only for compiling small programs).
- *Word Size.* 8 bit bytes, 16 bit words (hardware or simulated).
- *Secondary Storage.* Standard 8 inch floppy disk (the major system program files occupy roughly 70 K bytes).
- *Console Display.* 9600 bps ASCII terminal with x-y cursor addressing works best (slower CRTs or hard copy terminals can be handled, but less effectively).
- *Keyboard.* Uses ASCII keys for CR, ESC, ETX, BS, DEL and four positioning arrows (up, down, left, right).

In addition, the system is being used to drive a variety of printers such as the Diablo HYTYPE and Printronix 300, and for communicating via standard asynchronous lines.

Compatibility with Other Software Systems

In Project discussions with manufacturers of computers, on which the UCSD Pascal System might potentially be run, the most frequently asked question is: "How much effort will it take to adapt Pascal to run under my software system?" This question is understandable in view of the approach generally taken by the computer industry when a new language is to be installed on a machine produced in quantity. Unfortunately, this question misses the main point the Project is trying to make regarding transportable software. The effort needed to convert the Pascal compiler to run under the operating system of manufacturer "X" will generally be far greater than the effort to make the entire UCSD Pascal system run on that manufacturer's hardware. In the interest of promoting software transportability, the Project will generally not agree to adapt just the compiler to run under another operating system.

Pascal Language Extensions

Like many others who use Pascal as the basis for writing large system programs, the Project has found it necessary to extend the language. The most notable extensions have to do with strings of characters, for natural reading and writing from and to interactive files, and for tools needed in writing the software. A concerted effort has been made to implement all of the "standard" Pascal language as defined in *Pascal User Manual and Report*, by Kathleen Jensen and Niklaus Wirth (Springer Verlag, New York and Heidelberg, 1975). (However, UCSD Pascal still lacks the ability to allow procedure and function names to be passed as parameters.) The Project is making an effort to serve as coordinator among several large industrial firms which are preparing to use extended versions of Pascal for major programming projects. It is hoped that a consensus will emerge from this effort on extensions to the language for system programming. UCSD Pascal implements integers in two's complement form in 16 bit words, and real numbers in a 32 bit field. Since neither form is suitable for large integers or for business applications, it is planned to add the facility to handle fixed decimal numbers whose precision may be declared by the programmer.

Speed of Execution

Although the system is entirely interpretive, as currently implemented, execution speed is fast enough to permit highly interactive programs to be run on microcomputers. For example, compilation speed ranges from 600 to 700 lines per minute on the DEC LSI-11, or on an 8085 with a 3 MHz clock.

Availability

Copies of the system may be obtained by writing to UCSD Pascal Project, Maildrop C-021, La Jolla CA 92093. The system is available at a subscription fee of $200, made payable to "Regents of the University of California," which pays for materials, handling, and a limited amount of direct assistance to users. Those who wish to order the system should send details describing the system on which they wish it to run, or should request an order blank from the project. The system is copyrighted, but rights are granted to educational institutions and to bonafide computer clubs to make additional copies for their own noncommercial uses. A copy of the latest package of printed user manuals (about 250 pages) is available at a charge of $15, again made payable to the Regents of the University of California.

Though plans are in motion to convert the system to run on many different processors and configurations, the only systems currently supported use LSI-11, 8080 or Z-80 microprocessors with at least 48 K bytes of main memory, and IBM 3740 compatible standard floppy disk drive(s). For 8080 and Z-80 users, the method of adapting the system to run on new hardware is similar to that used by Digital Research Inc in distributing the CP/M operating system; and the Project will distribute a conversion package similar to theirs. Versions of the sys-

As of this writing (1 July 1979), SofTech Microsystems Inc (94 Black Mountain Rd, Building 3, San Diego CA 92126) is the sole licensee of the UCSD Pascal system. Questions about prices and availability of the system can be directed to the above address.

Also, note that UCSD Pascal *is a trademark of the Regents of the University of California.*

tem for other microprocessors are not likely to be ready for release until October 1978 at the earliest. Release on floppy disks other than those compatible with the 3740 format will depend upon availability of hardware to the Project.

In addition to the main software system, educational materials are available separately for an introductory course on problem solving and programming using Pascal. A textbook *(Microcomputer) Problem Solving Using Pascal* is available from Springer Verlag Publishers, 175 Fifth Av, New York NY 10010 ($9.80). The Project can supply a set of automated quizzes designed for use with the textbook in a self-paced course of study.

Help from the User Community

Readers can help by letting their favorite hardware vendors know that they want UCSD Pascal to be available in machine independent form. The Project has noted an increasing number of manufacturers who report that customers are requesting Pascal, and this has a real influence on their business decisions. Readers can also help by joining the international Pascal Users' Group (send $4 c/o Andy Mickel, 227 EX, 208 SE Union St, University of Minnesota, Minneapolis MN 55455) and pressing PUG to establish a technical board to oversee UCSD Pascal as a community project.■

Note on the Pascal User's Group

As of July 1, 1979 the Pascal User's Group (PUG) has over 3300 members in 47 countries. Those interested in joining can contact Andy Mickel at the University of Minnesota Computer Center, 227 Ex Engr, University of Minnesota, Minneapolis MN 55455, (612) 376-7290. The Pascal Newsletter *is published four times a year on a July to June schedule, with a subscription fee of $6 per year. All issues for the current year are sent with a new subscription, and back issues are available.*

In Praise of Pascal

David A Mundie

As has been pointed out in these pages before, personal computing will never achieve its full potential as long as our state of the art machines are hobbled down with a language as far from state of the art as BASIC is. Some have argued for designing a special high level language for microprocessors, but I personally fail to see why we don't just implement Pascal and be done with it. I would like to look briefly at the language itself and try to explain why it seems the logical choice to me.

I am an applications programmer with no theoretical interest in computing whatsoever. What I like about Pascal is not the theory of its design, though that seems sound enough, but rather the fact that it lets me formulate my problems in my own terms. In Pascal more than in any other language I know, I can remain on the abstract, algorithmic level where, as a human being, I function best. Because of this pragmatic bias, much of what follows will be an informal discussion appealing to the reader's intuitions rather than a technical demonstration. I shall use BASIC for comparative purposes, since it is the tyrant in the field.

I find Pascal easy to use because it allows me to define new data types which express my data meaningfully. It provides control structures with which I can express what I want done to my data clearly and naturally. Pascal allows and encourages me to formulate my thinking in a structured way. Let us examine these three aspects of Pascal in reverse order.

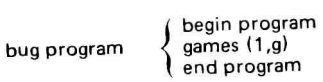
Program Structure

Pascal is a resolutely structured language. A Pascal program is structured into blocks. Each block bears a heading which gives it a name and specifies its parameters. Roughly speaking, a block consists of a definition part, in which constants, types, variables, and subroutines are defined, and an action part, which contains the algorithm of the block. This rigorous separation of data definition and algorithm expression is partly responsible, it seems to me, for the greater legibility of Pascal compared to ALGOL.

Subroutines are themselves block structured and may thus be nested within one another. This allows the declaration of "local" variables and subprograms, meaning that storage may be allocated efficiently; yet it is easy to guard against unwanted side effects.

What does all this mean for the practicing programmer? The answer may perhaps best be seen in the light of a claim recently repeated by David Higgins in the October 1977 BYTE ("Structured Program Design," page 146). Higgins presents the now well established arguments in favor of structured programming, but goes on to contend that once a program is designed in a structured way, using for example Warnier-Orr diagrams, "it does not matter what programming language you code it in." This assertion seems pretty improbable on the face of it, and if true it would be a powerful argument against Pascal. I think that a rapid examination of two test cases will show it to be quite unjustified.

Let us take our test cases from the "bug" program which Higgins uses as his own example. Higgins would have us break the program down into three parts, as expressed in the following Warnier-Orr diagram:



Nothing in the BASIC listing which accompanies the article even remotely suggests

this overall algorithm. Look at what we might have in an equivalent Pascal program:

```
program bug;
    begin
        beginprogram;
        games;
        endprogram
    end.
```

Need I point out that to all intents and purposes the Pascal program *is* the Warnier-Orr diagram, with only a few notational differences such as the replacement of the brace by the symbols **begin** and **end**? Are we really asked to believe that this one to one correspondence between the problem and the program does nothing to simplify the programming task? On the contrary, it simplifies matters enormously.

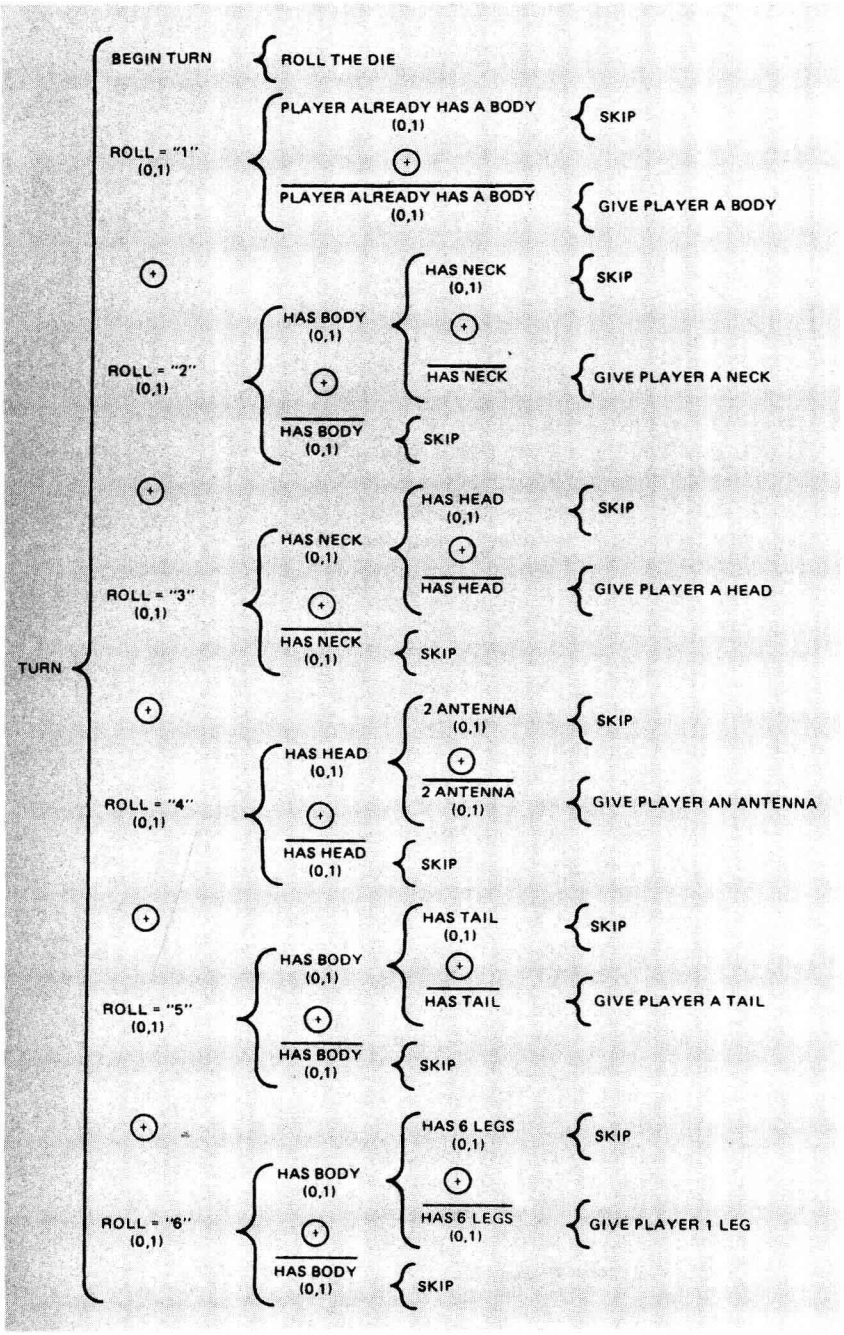


Figure 1: Warnier-Orr diagram for subprogram "turn" of the bug program. This is clear, but note how much bulkier it is than the Pascal program in listing 2. The Warnier-Orr diagram won't even run on a computer.

Considerations of space prevent me from giving the rival BASIC and Pascal versions in full. Another striking example is presented in figure 1 and listings 1 and 2, which show the Warnier-Orr diagram for the "turn" subprogram, Higgins' coding of the subprogram in BASIC, and the Pascal equivalent. Higgins calls his BASIC coding "simple and straightforward." Tastes differ but that is a phrase I would have reserved for the Pascal version. Higgins has had to fake truly structured programming in a language which fights his efforts every step of the way, and the results are tortured and confusing. In contrast, the Pascal coding is, once again, a nearly perfect reflection of the Warnier-Orr diagram, so much so, in fact, that most Pascal users will probably feel, as I do, that the diagrams are a useless intermediary step, less clear and bulkier than the program itself. The intent of the Pascal program segment is so transparent that in my opinion it could almost be understood by a complete programming novice.

Before leaving the topic of program structure, we should perhaps remark that Pascal subprograms (procedures and functions) bear names, not numbers, virtually eliminating the need for the comments which pepper any well documented BASIC listing. Furthermore, because Pascal subprograms can have parameters, the programmer is encouraged to use a single subprogram for a single task. Higgins has written separate subprograms for each body part, whereas for a Pascal user it is virtually impossible to resist the temptation of passing the arrays body, neck, head, etc, to a single procedure "give" as parameters.

Algorithm Expression

Program structure alone does not explain the relative clarity of the Pascal listing in listing 2. We may also use that listing to illustrate the tools which Pascal provides for expressing algorithms.

Logical operators: Pascal provides the logical operators (**and, or,** and **not**) which are so painfully lacking in BASIC and without which expressing an algorithm is so clumsy. The use of the operator **and** in the turn subprogram is a good example; or the reader may want to express "if (x=1) or ((y>2)and(z=3)) then. . ." in BASIC.

Conditional statements: Pascal's **if** structure groups statements with the condi-

tions for their execution. The **if** statement is of the form:

```
if<expression>
   then<statement_1>
   else<statement_2>
```

The expression is evaluated as being either true or false. If it is true statement_1 is performed; otherwise statement_2 is performed. Suppose the expression is: X=1. In English the **if** statement translates to:

> if X equals 1 then perform statement_1; else perform statement_2.

Pascal offers a very flexible **case** statement which is remotely related to the computed GOTO statement to be found in some BASICs. It is much more powerful because, among other things, selector values need not be contiguous, and actions are grouped with the conditions for their execution. A good example of the **case** statement's clarity is to be found in the procedure "turn," where the action taken depends on the value of roll.

Repetitive statements: BASIC provides only one repetitive control structure: the FOR statement. But there are innumerable situations where we do not know ahead of time how many times a given action is to be repeated. In such cases BASIC users have two choices. One is to set up a dummy FOR statement with a jump out of it when a certain condition is met: whence the ubiquitous "FOR I=1 TO 9999" statements in BASIC programming. This is bad because it seriously disguises the intention of the algorithm. One's natural expectation is for such a loop to be executed 9999 times, but that is not the case. The other solution is for the programmer to fake an appropriate control structure with GOTOs or conditional jumps. That is what Higgins has done in his program to express the fact that the computer and the human take turns until the game is over:

```
210 REM TURNS (1,T)
220 LET EGAM = 0
230 GOSUB 390
240 IF EGAM = 0 THEN 230
250 REM END GAME
260 GOSUB 1150
```

This is no doubt the best one can do in BASIC, but just consider how much more elegant the Pascal version is:

```
repeat turns until endofgame
```

This is typical of the way in which Pascal's control structures make algorithm expression a source of joy rather than a contortionist exercise. In addition to the **repeat** statement, Pascal offers a **while** statement for the case when an action is to be repeated as long as a condition is true.

Data Definition

Now that we have seen how much easier it is to express what one wants done to data in Pascal than in BASIC, let us turn to the wonderful data types which Pascal makes available for manipulation. Data types are the programmer's buffer between his abstract formulation of an algorithm and the messy realm of bit level details where that algorithm will eventually be executed. Pascal makes defining new types a trivial task. Once a new data type is defined, it is in effect indistinguishable from a predefined type and may be used in any way a predefined type may be. We leave BASIC behind at this point, since that language has no facilities for creating new types.

The bug program was too simple to provide examples of data structuring, so we shall have to turn elsewhere. Being a birdwatcher, I shall replace the traditional "Christmas card list" example by a bird data bank. I can do no more than skim the surface, so I ask the reader's indulgence if some of the listings are not fully explained. I am not trying to teach Pascal, but merely to spark intuitions.

Pascal distinguishes between simple

```
490 REM TURN SUBROUTINE
500 REM PLAY=1;PLAYERS TURN-PLAY=2;COMPUTERS TURN
510 REM ROLL DIE
520 LET ROLL = FIX@(((RND(0))*6.0))+1
530 PRINT:"ROLL IS A",ROLL
540 IF ROLL = 1 THEN IF BODY(PLAY)#1 THEN GOSUB 690 ELSE;ELSE;
550 IF ROLL = 1 THEN 650
560 IF ROLL = 2 THEN IF BODY(PLAY) = 1 THEN IF NECK(PLAY)#1 THEN GOSUB 760
570 IF ROLL=2 THEN 650
580 IF ROLL=3 THEN IF BODY(PLAY)=1 THEN IF NECK(PLAY)=1
    THEN IF HEAD(PLAY)#1 THEN GOSUB 820
590 IF ROLL=3 THEN 650
600 IF ROLL = 4 THEN IF HEAD(PLAY)=1 THEN IF ANTE(PLAY)#2
    THEN GOSUB 880
610 IF ROLL=4 THEN 650
620 IF ROLL = 5 THEN IF BODY(PLAY)=1 THEN IF TAIL(PLAY)#1 THEN GOSUB 940
630 IF ROLL=5 THEN 650
640 IF ROLL = 6 THEN IF BODY(PLAY)=1 THEN IF LEGS(PLAY)#6 THEN GOSUB 1000
650 LET A=3
660 RETURN
```

Listing 1: BASIC listing for Warnier-Orr diagram in figure 1. This is the best one can do in BASIC, but is still a far cry from the clarity of the Pascal listing.

```
procedure turn;
  begin roll:=trunc(random(1)*6)+1; writeln('roll is a',roll);
    case roll of
      1: if(body[player] ≠1)then give(body);
      2: if(body[player] =1)and(neck[player] ≠1) then give(neck);
      3: if(neck[player] =1)and(head[player] ≠1) then give(head);
      4: if(head[player] =1)and(ante[player] ≠2) then give(ante);
      5: if(body[player] =1)and(tail[player] ≠1) then give(tail);
      6: if(body[player] =1)and(legs[player] ≠6) then give(legs)
    end
  end;
```

Listing 2: The Pascal listing equivalent to listing 1. Note the clear affinity between the listing and the Warnier-Orr diagram. Notice that arrays are indexed using square brackets.

and structured types. Let us examine each in turn.

Simple types: These are the basic building blocks of which any structured type, no matter how complex, is ultimately composed. In addition to integer, real, and character types, Pascal offers two additional simple types which as far as I'm concerned come close to exhausting the simple types needed in a general purpose language. The first is the defined scalar type, and is defined by simply listing the values which a variable of the new type may take on. Suppose I need a data type for the various habitats in which a bird may appear. In Pascal I write:

```
type h = (ocean,rivers,fields,suburbs,forests,
          mountains)
```

A variable of type h may take on any of the values listed. This means that while programming I may continue to think in terms of habitats, and am not forced to descend from that abstract level and think in integers, as I would have to do in BASIC. This also makes for virtually self-explanatory programs. Compare "IF HABITAT=3 THEN. . ." with the much more transparent "**if** habitat=fields **then**. . . ."

The second simple data type is the Boolean, and is extremely useful in programming since one is constantly controlling program flow with Boolean expressions. Boolean variables take on the values true and false. Languages without such variables must make do with integers, which muddles things since one's natural expectation is for integers to count something. The Pascal user may simply write "**if** good **then**. . .", which is the way we think; the BASIC programmer must write "IF GOOD = 1 THEN. . .", which is alien to the way we think.

A large part of Pascal's elegance comes from the fact that in most contexts these simple or scalar types may be used indifferently. Thus for example the type h as defined above could be used as the index variable in a **for** statement:

```
for habitat := ocean to mountains do
```

or in a case statement, or as the index type of an array:

```
if foundin [fields] then
```

Furthermore, functions may return any scalar type: we have already seen the function "endofgame" which returns a Boolean value.

Structured types: In addition to the simple types, Pascal offers five different structuring methods: arrays, records, sets, files, and pointers. These different methods may be combined in virtually limitless ways. One may have files of arrays, pointers to records, arrays of sets, pointers to files of arrays of records, and so on. This extreme flexibility of data structuring methods is one of Pascal's most exciting features. The type array should be familiar, but let us look briefly at the other four structured types.

Sets: Each bird in my hypothetical data bank has associated with it a set of habitats in which the bird may be found. Having defined the type h as above, all I need to do to set up a variable habitats which will be a set of different habitats is to write:

```
var habitats: set of h
```

When constructing the entry for the robin, I will write:

```
habitats := [fields,suburbs]
```

thus assigning to the robin the set of habitats containing the two elements fields and suburbs. When going on a trip to the mountains, I can test whether mountains are in a given bird's set of habitats by the following simple test:

```
if mountains in habitats then
```

Imagine trying to do this in BASIC. Pascal provides a variety of set operators which allow set manipulation in all its generality.

Records: Let us imagine that each entry in my data bank will contain the bird's name, its length, and a set of habitats where it may be found. The entry cannot be an array, since components of arrays must all be of the same type. The appropriate data type is the record, defined in Pascal as follows:

```
type bird =  record
               name: string;
               length: real;
               habitats: set of h
             end
```

This is a simple and logical way of grouping data of different types into a meaningful whole. Given variables robin and redbreast of type bird, a simple assignment statement will set one equal to the other:

```
robin := redbreast
```

To test whether a robin is more than 20 cm long, we would have:

```
if robin.length>20 then
```

and so on. These are simple examples, but they suffice to illustrate the flexibility of the record type.

Files: Now let us suppose that I have 600 entries of type bird in my data bank, and want to make a list of all the birds whose length is greater than 20 cm. It is pointless and wasteful to keep all 600 records in memory for such a task; all I