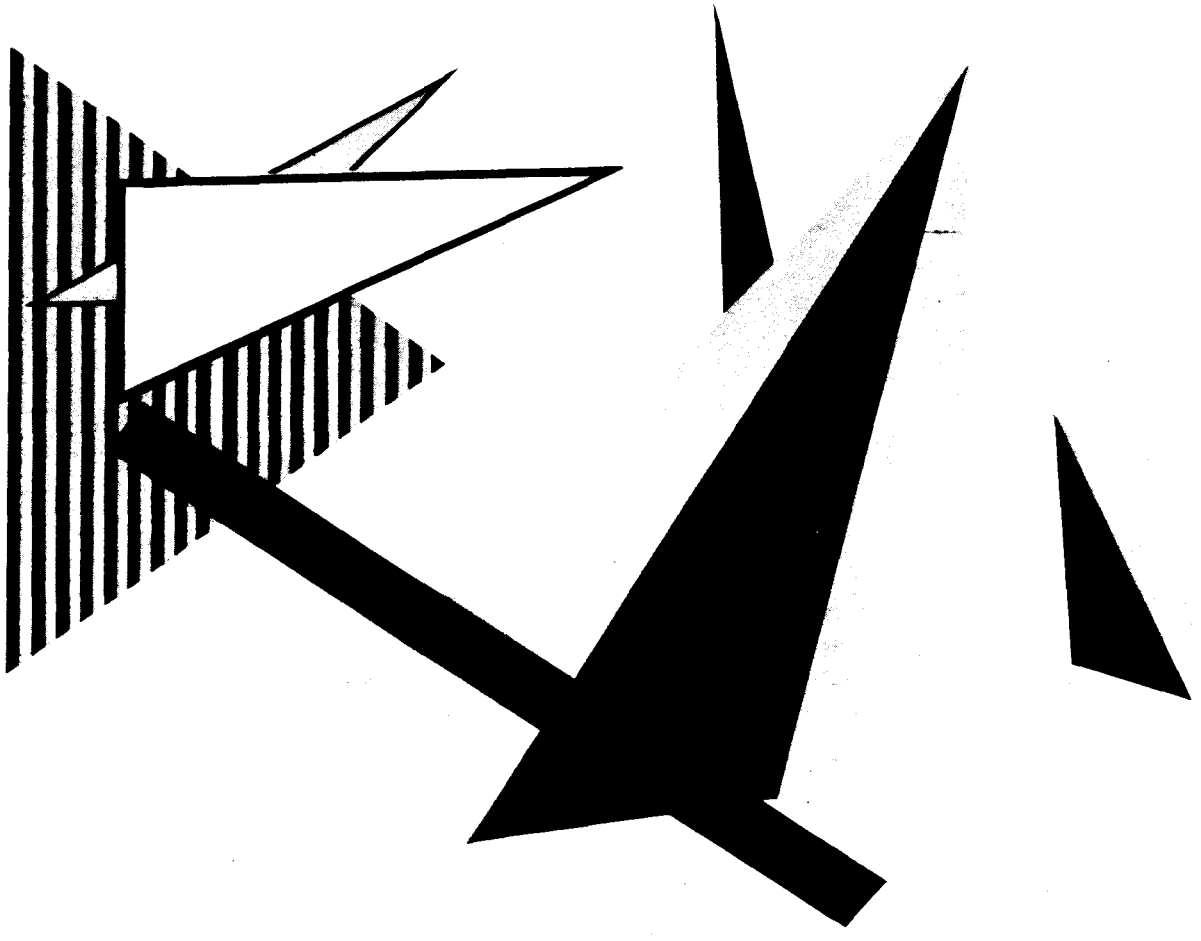
$10.00

Pascal & MSDOS



Computer Graphic Design by
Catherine Del Tito

Number
27
November 83

ISSN 0739-1900

SUBVERSIVE SOFTWARE

So cheap and useful it's...dangerous!

A radical idea in software development. Useful Pascal programs, with:

- Complete annotated *source listings;*
- Disk(s) with *source code and compiled code;*
- User manual;
- Complete programmer documentation describing data structures and algorithms; and giving suggestions for modification.

Modify them, include them in your own systems, or simply use them.

A growing library of software tools you'll find hard to resist.

SUBVERSIVE SOFTWARE

TPL

The Text Processing Language. A text-file runoff program consisting of a set of text-processing primitive commands from which more complex commands (macros) can be built (as in Logo). Features include:

- Complete customization of text processing through macro definition and expansion, looping structures, and conditional statements;
- Adapts to any printer;
- Pagination;
- Text justification and centering;
- Indexing and tables of contents;
- Superscripts and subscripts;
- Bolding and underlining;
- Multiple headers and footers;
- End notes and footnotes;
- Widow and orphan suppression;
- Floating tables and 'keeps.'

$50

SUBVERSIVE SOFTWARE

DBX

Blocked Keyed Data Access Module. Maintains disk files of keyed data. Can be used for bibliographies, glossaries, multi-key data base construction, and many other applications.

- Variable-length keys;
- Variable-length data;
- Sequential access and rapid keyed access;
- Single disk access per operation (store, find, delete) in most cases;
- Multiple files;
- Dynamic memory allocation for RAM-resident index and current "page" of entries;
- Includes demonstration program and testbed program.

$50

SUBVERSIVE SOFTWARE

PDMS

The Pascal Data Management System. A user-oriented data management system in which numeric and alphanumeric data are stored in tables with named columns and numbered rows. Currently being used for dozens of different kinds of business and scientific applications, from inventory management to laboratory data analysis. Includes over 20 Pascal programs; more than 10,000 lines of code. Main features include:

- Maximum of 32,767 rows per file;
- Maximum of 400 characters per row, and 40 columns per table;
- Full-screen editing of rows and columns, with scrolling, windowing, global search replace, and other editing features;
- Sorting, copying, merging, and reducing routines;
- Mailing label program;
- Reporting program generates reports with control breaks, totals and subtotals, and selects rows by field value; many other reporting features;
- Cross-tabulation, correlations, and multiple regression;
- Video-display-handling module;
- Disk-file-handling module.

Many other features. UCSD formats only.

$250

SUBVERSIVE SOFTWARE

ZED

Full-screen text editor; designed to be used either with TPL or by itself.

- Full cursor control;
- Insert mode with word wrap;
- 'Paint' mode;
- Single-keystroke or dual-keystroke commands;
- Command synonyms;
- Global search and replace;
- Block move, block copy, and block delete.

$50

SUBVERSIVE SOFTWARE

SCINTILLA

A log logit curve fitting program for radio-immunologic data; must be used with PDMS (described above).

- Multiple protocol files;
- Quality control files;
- Four-parameter non-linear curve fit.

UCSD formats only. **$250**

SUBVERSIVE SOFTWARE

CHROME

Chromatography data analysis program:

- Graphic display of analog data;
- Panning and zooming;
- Automatic peak-finding and baseline calculation;
- Full interactive peak editing;
- Computation of peak areas;
- Strip charts on C. Itoh and EPSON printers.

$100

SUBVERSIVE SOFTWARE

PLANE

Planimetry program:

- Bit-pad entry of cross sections;
- Real-time turtlegraphics display;
- Calculation of areas;
- Saves calculations to text file.

$100

SUBVERSIVE SOFTWARE

MINT

A terminal emulation program for communication between computers of any size.

- User-configurable uploading and downloading of files;
- X-ON/X-OFF and EOB/ACK protocols;
- Interrupt-driven serial input (for Prometheus Versacard in Apple II);
- Printer-logging.

$50

For more information, call 919-942-1411. To order, use form below or call our toll-free number: 1-800-X-PASCAL.
(In N.C. use 1-800-642-0949)

Check appropriate boxes:

FORMAT	PRODUCT	PRICE
8" UCSD SSSD	DBX	$ 50
5¼" Apple Pascal	PDMS	$250
5¼" UCSD IBM PC 320k	TPL	$ 50
8" CP/M SSSD	ZED	$ 50
5¼" IBM MS-DOS	MINT	$ 50
5¼" CP/M Osborne	SCINTILLA	$250
	CHROME	$100
	PLANE	$100

Name _____

Address _____

☐ MasterCard ☐ VISA ☐ Check ☐ C.O.D.

(Please include card # and expiration date)

SUBVERSIVE SOFTWARE
A division of Pascal & Associates.
135 East Rosemary St., Chapel Hill, NC 27514

Pascal & MODULA-2

Formerly *Pascal News*

Serving Pascal Users Group and the Modula-2 Users Group

November 1983 **Number 27**

About the cover:
The cover computer graphics were created by computer artist Catherine Del Tito. The program was written for an Apple IIe and a Vectrix computer system.

UCSD p-SYSTEM™ with UCSD PASCAL™ FOR THE VICTOR 9000™

Standard Features

- System Foundation:
  - Operating System - Single key commands
  - Filer - For file management
  - Screen Editor - Powerful text editor
  - Utility Library - Fast development tools
- **UCSD PASCAL Compiler** - Only requires 128K
- Utilities for the Victor:
  - Keyboard Editor - Define your own keyboard
  - Character Editor - Design character sets
  - Config - Define machine parameters
  - Diskutil - Disk copy/format
  - Remote - File porting
- Native Code Generator - For faster execution
- Turtlegraphics - 800 x 400 pixel access
- RAMDISK Capability - For RAM above 128K
- Assembler - 8086 with macros
- Documentation:
  - Owner's Manual
  - User Manual/Supplement
  - Architecture Guide
  - Installation Guide Excerpts
  - DEMODOC diskette - On line tutorial for p-System overview

Optional Features

- Hard Disk Support Software - For internal hard disk
- FORTRAN 77 Compiler - Only requires 128K
- BASIC Compiler
- 8087 Support Software
- Advanced Systems Editor
- Xenofile™ - Convert CP/M files
- Applications Software (call for information)

TDI SYSTEMS, INC.
620 Hungerford Drive
Suite 33
Rockville, Maryland 20850
(301) 340-8700

TDI LIMITED
29 Alma Vale Road
Bristol, U.K. BS8 2HL
0272 742 796

Editor's Notes

Announcing a $10,000 Contest for the Best Article or Application Voted by the Members

Dear Member:

By now those of you in the United States have received a questionnaire and announcement concerning *"Pascal News."* The reader survey is a very important element in our ability to contain the cost of this newsletter. When we solicit advertisers, they want to know about the members: Who we are, What we do, How much we spend. I realize these are very probing questions and you may feel they are too personal. To separate your name from your information, the subscription card is a self mailer and the return envelope for the survey will assure your anonymity. Thank you in advance for your help.

Let me explain the announcement of our name change from *"Pascal News"* to *"Pascal & Modula 2."* Pascal has encouraged and endorsed rational programming. The language aids in the segmentation of a job into small parts through procedures and functions. I have enjoyed learning and using this language, but Pascal does have limitations.

Many limitations are part of the design of a teaching language. Pascal is a very nice base, a core, to which extensions have been added by every implementor. We have accepted this and published reports of large extension packages, UCSD Pascal, Concurrant Pascal and Path Pascal.

These extensions were added to a language created for teaching programming principles. This design goal allows a general purpose language, but not an all purpose language. Nicklaus Wirth recognized these limitations and created a language that would contain Pascal's good features and satisfy the goal of an all purpose language.

Modula-2, created in 1977, is that language. In this issue, you will read Modula-2 product announcements. In these announcements, and in other articles, claims are made that Pascal is finished, that Modula-2 should replace Pascal in all cases. Maybe. I believe Pascal can and should remain in the position for which it was designed. The premier teaching language is Pascal. The easy transition from Pascal to Modula-2 makes Modula-2 an excellent second year language. Restrictions are necessary in the introduction to programming and Modula-2's flexibility does not focus a student to basic principles.

Of course I may be "all wet," but I believe Pascal should remain.

Pascal's teaching tool strength, Modula-2's all purpose ability, and the relatively painless transition between them make Pascal and Modula-2 proper subjects for the Pascal Users Group. Pascal News should reflect this wider interest in content and name. The new name, *"Pascal & Modula 2,"* keeps Pascal as the first name and can be found in the same place in periodical indices as *"Pascal News."* This should make the libraries happy.

The new logo places Pascal within Modula2, a reference to Modla-2's ability for operating system programming, above Pascal, and its ability for machine specific programming, below Pascal. I hope you welcome the change and contribute to the discussion and promotion.

This brings me to the contest for $10,000. It really is a promotion. There are many ways to promote this newsletter and I have tried a few. One way is to keep the members we have now. I have promised four issues per year and this is the fourth of 1983. A renewal notice was sent out in January and I thank you for your subscription. In October the reader survey and subscription card were mailed and I hope you renew promptly. I placed a small ad in *"PC"* magazine. It is in their Blue Book section and will run from September 1983 through March 1984 and should attract new members. Announcement of our name change and subscription information was sent to fifteen magazines. Unfortunately there are many Pascalers who do not know of the Pascal Users Group.

I am now very familiar with the costs of this newsletter and can say that income closely matches costs. I also know that if membership exceeds 5,000, we will have a surplus. What would we do with this money? Well, I propose that it be used for a promotion, and I cannot think of a better promotion than to vote for the best article or application published in *"Pascal & Modula 2."*

I do not know whether the prize will be winner take all or first, second and third place division of the money. Your letters will help me form the rules.

The first rule is we must have 5000 members. Fewer members and we cannot afford this contest.

Now, you may recognize a little circularity here. The promotion attracts members and increased membership allows the promotion. Because of my other responsibilities (wife, home, job and country) I need your help to announce this contest in all quarters. If all 4000 members will send one letter (i.e. to friends, to users groups, to fellow students, to magazines) to announce this contest and one new member joins per letter, well, I think you get the idea.

Rule number two—all contestants must be members.

One more idea. If this contest generates enough material and money the newsletter will be published more often. Please send your ideas.

Editor's Notes

Pascal at Work

A typesetting program called TeX, created by Donald Knuth, is available for the cost of distribution. 15,000 lines of Pascal code puts TeX in the nontrivial category.

Basic information is available from two sources. The book *"TeX and METAFONT"* is available for $12. Digital Press, 12A Esquire Road, North Billerica, MA 01862. A new book, *"TeX Book,"* should be in print by the end of 1983 from Addison-Wessley.

Continuing information is provided by the TeX Users Group (TUG). Membership is $30. TeX Users Group, c/o American Mathematical Society, PO Box 1571 Annex Station, Providence, Rhode Island 02901, USA.

"Small Talk," a language from the Xerox Palo Alto Research Center was revealed in the August 1981 issue of *"Byte"* magazine. The Small Talk virtual machine, a software interface to the real machine, was given to four companies. They agreed to debug the virtual machine and share information.

I found it interesting that Tektronics implemented the virtual machine in Pascal. I understand that Tektronics internal programming uses Pascal or Modula-2. The papers regarding the Small Talk research are assembled in the book *"Small Talk-80 Bits of History, Words of Advice."* This book and *"Small Talk-80 The Language and its Implementation"* are available from Addison-Wessley.

Andy Michel informs me of a version of "C," the Bell Labs language implemented in Pascal. Very interesting.

This Issue

A large part of this issue is devoted to a compiler creator. With a language specification this program will write the compiler. I have been told this program is very valuable. Using it, a contract for a compiler was satisfied and a handsome profit gained. (Sounds good to me.)

Robert Gustafson in a letter complains about the typesetting of this newsletter. Expense, time and typographical errors are his concern. When printing more than 2500 copies, typesetting reduces overall expenses. Typesetting consumes 3 weeks' time, not an unreasonable amount for a quarterly.

Typographical errors are a problem and programs are photographed from originals to avoid them. A better way to assure correct information and dark, clean type is to capture the author's original keystrikes. Floppy disks and mag tape will do the job, but I would have to convert the many magnetic formats to one the typesetter could use.

Computer networks may be a better solution. Networks force all submissions to a common format. From this, the typesetter will make only one conversion to his format. If there is a consensus, I will establish a bulletin board and file system on the most popular network.

Please send your ideas. This is a one-man operation and I appreciate and need your help.

I hope you enjoy this issue.

Charlie

Pascal & MODULA-2

2903 Huntington Road
Cleveland, Ohio 44120

Publisher and Editor:
Charles Gaffney

General Consultants:
Studio Graphics Advertising

Production Manager:
Spence Coghlan

National Sales Representative:
John Bachmann

The *Pascal & Modula-2* is published for the Pascal Users Group and Modula-2 Users Group, 2903 Huntington Rd., Cleveland, Ohio 44120. *Pascal & Modula 2* is a direct benefit of membership.

Membership dues are $25.00 U.S. regular, other forms of membership please inquire. Inquiries regarding membership should be sent to the above address. Magazine correspondence and advertising should be sent to the editor at the aforementioned address. Advertising rates are also available from the above address.

Open Forum

August 22, 1983

Dear Charlie:

I have been a subscriber to *Pascal News* for a long time. According to the date on my mailing label, it looks like I will continue to be a subscriber well into the distant (85) future.

I agree with many of the notes published in *PN #26* concerning your contributions to the well being of *Pascal News*, particularly the general management and timely publication.

However, I believe that *PN* is doomed to die because of one change you have made. Previously, the magazine was printed using the original letters submitted by *PN* correspondents. This was an inexpensive method of getting the information out to the readers. Your current system of rekeyboarding all of the correspondence can only be costing you dollars without increasing the utility of the information. Also, as people decide they can do without *PN* for $25/year, your circulation will decrease. One of the previous attractions of *PN* was that since it was so cheap, it reached everyone. A reduced circulation will cause contributors to look for other distribution media. I realize you are saving money on postage, but since *PN* is sent out bulk rate, the savings are only a small fraction of the cost of typesetting. Since these typesetting costs are the same whether you send out 1 copy or 3000 copies, I predict that you will not be able to compete with other publications in the field and *PN* will die.

If you return to direct printing of correspondent copy you will be in good company. The DECUS special interest group newsletters (RSX SIG for example) are printed from yechy LA36 printer copy. I read them cover to cover as soon as they arrive. There are also a lot of expensive stock market newsletters which are printed from typewriter copy. These sell for big bucks because the buyers are interested in TIMELY information, not typography and art work suitable only for decorating a coffee table.

By concentrating on quick publication, I believe you will find advertisers willing to pay for space in *PN*. Also, when it comes to selling advertising space, 3000 subscribers are much better than 300! Another important criterion for an advertiser is knowing that what is sent to you will appear unchanged in the publication. I have noticed that there are a number of typographical errors in the typeset issues of *PN*. These are inevitable in a keyed-once, proof-read-once publication (I know the statistics, one of our programming systems typesets approximately 40% of the municipal bond issues in the U.S. Errors here are a no-no. As a consequence, we have done quite a bit with automatic proofreading and word processor telecommunications). If you use advertiser copy directly, there can be no problems later with omissions of important parts of the copy or inadvertent changs to prices, etc.

You might consider including a questionnaire in the next issue of *PN*. I'm sure that the majority of your current subscribers would be much more enthusiastic about paying $9/year for the latest information from your correspondents and advertisers printed "as is" than $25/year for the present "remassaged" format. If you choose an even lower price, advertisers will pay more for the resulting increased circulation. Simple economics!

Sincerely,

Robert D. Gustafson
President
Simulation Specialists Inc.
609 West Stratford Place
Chicago, IL 60657

June 29, 1983

Dear Editor:

I enclose a short article, *Two Pascal Devices*, for publication in *Pascal News*. I shall appreciate your acknowledgement and, if you use the article, a copy of the issue in which it is printed.

Sincerely yours,

Harley Flanders
Professor of Mathematics
Florida Atlantic University
Boca Raton, FL 33431

P.S. May I humbly suggest that you do not print the output of line printers or dot matrix printers. It is just too hard to read.

July 5, 1983

Dear Mr. News:

This is to request address correction from that found on the label:

Jeff Davis [81]
135 Turtle Creek
#1 Roper Mt. Rd.
Greenville, SC 29603

to the following:

Jeff Davis
6549 Quiet Hours Apt. #201
Columbia, MD 21045

I am entering this using an editor found in "Software Tools in Pascal," one of the best books on toolbuilding I've ever read, and printing out using a copy of "Prose" from an early copy of *Pascal News* recompiled on an Apple ///! As you see, my interest is in learning by building tools in Pascal.

By the way, there is a local bulletin Board system (my next tool may be a spelling checker!) called Magus which is actually an operating system written in Pascal by Craig Vaughn that is worthy of note. I'll suggest that he submit an article describing it and see what his reaction is.

As a last comment, I had been out of the country for a few years and only recently resubscribed. How's Modula-2 doing in the states? I've ordered their documentation but version for my computer (Apple ///) somewhat tardy.

Thanks and it's great to be back in touch with Pascal reality again!

Sincerely,
Jeff Davis

May 31, 1983

Dear Mr. Gaffney:

We have purchased a Motorola EXORciser development system for developing 6809 based products. We contracted to an outside vendor for the initial software development on a new product. All software was written in HP64000 Pascal. We will be doing all software maintenance and enhancement in house and we are reluctant to do this in assembly language. Therefore, I am attempting to find a Pascal compiler to run on the 6809 EXORciser which is as compatible as possible with the HP Pascal. I would appreciate any help you can give me on this.

We also have a Texas Instruments DS 990 minicomputer with a Pascal compiler which we would like to use for electrical and electronic engineering support and development. Any information on available Pascal electronics packages would be helpful.

Very truly yours,

MILLER Electric Mfg. Co.
Bruce A. Casner
Project Engineer
P.O. Box 1079
Appleton, WI 54912

Open Forum

June 2, 1983

Could you send me any information you have on Apple and JRT Pascal? Also, I would be very appreciative if you could recommend any books for someone who knows BASIC and 6502 Asembler.
Thank you.

Larry Houston
169 West 8th Street
Peru, IN 46970

December 10, 1982

Mr. Gaffney:

I read about *Pascal News* from a UNIX NetNews (a network of UNIX installations sharing news) article. Could you send me the back editions containing the Lisp compiler-interpreter (written in Pascal)? Enclosed is $15 for the year of back issues containing the Lisp compiler. Send me a bill for shipping if $15 does not cover it all. I am glad to see you continuing this magazine.

Respectfully,

Fred R. Finster
8549 Evanston Ave. N.
Seattle, WA 98103

December 20, 1982

Dear Charles Gaffney (Charlie),

I have been infomed that PUG (Australia) has distributed its last issue (#24) of *Pascal News*, and that the subscription list and balance of funds has been transferred to the U.S.

According to my records, I was paid up through 1984 (renewed for 3 years mid-1981).

However, I understand that there will probably need to be an adjustment, so please could you apply my outstanding subscription toward whatever extension is appropriate.

Please, if possible, inform me what the final position is; I am very keen to maintain my continuity of membership, as I think PUG is very worthwhile.

Thanks a lot.

Yours sincerely,

G. A. Foster
5/138 Clarence Road
Indooroopilly
Queensland, Australia 4068

February 15, 1983

Dear Mr. Shaw:

I know that your job keeping going the PUG is a great one. As we create the Mexican Wang Users Group with 200 members now, after 4 years about 4 people do the whole work.

Our company with 32 people has an obligation to use an accumulative half hour daily to do some investigation; that is why we are interested in implementing the Pascal in our machine, a Want 2200-VS-80.

I believe that our specifications were wrong when we asked for the Portable Pascal P4, because we have not been able to get started.

Dr. Niklaus Wirth wrote me that the PUG has an 360 IBM compiler. I would like to know how can we be able to get it, because our computer with very little modification can run IBM assembler programs.

If I can do anything to help the PUG please let me know.

Thanking you in advance for all the trouble I may cause, looking forward to your answer.

Sincerely,

Miguel M. Soriano Lopez
Technical Director
Data, S.A.
Av. Homero No. 1425-1201
Mexico 5, D.F.

June 5, 1981

Dear Mr. Soriano,

It was a pleasure to hear from you after so many years. I fondly remember that stay in Mexico in 1963.

I guess the best way to "get in touch" with me is by writing, as you did. However, I do not see a chance for me to visit Mexico in the near future, as I am quite committed and busy at my position here at ETH.

Good luck for your Pascal compiler project. Are you aware of the compiler for the 360, also available from PUG? Perhaps it would be easier to use that compiler instead of the P, because your machine is—as you write—similar to the 360.

Sincerely yours,

Prof. Niklaus Wirth
Eidgenossische Technische Hochschule
Instit fur Informatik
ETH-Zentrum
CH-8092 Zurich

August 31, 1983

Dear Charlie:

I use Pascal at the National University of Mexico in a Burroughs 7500 or on a PDP-11, and at my work I am trying to install the Pascal-P you send me last year.

Several doubts I had, I asked Dr. Wirth, who answer me and recommended that will be easy to implement the IBM-370 version, which is more similar to my Wang 2200-VS-80 machine.

On the 25th *Pascal News* is a report about an IBM-370 Pascal. I will like to know if Joseph A. Minor of Cornell Computer Services would like to work together with me, to give to the PUG a Pascal compiler for the Wang VS. Only in Mexico there are more than 200 installations; I believe that at the USA are several hundreds who may be Pascalers, if the PUG will have it.

I hope to hear from you soon, thanking you for the trouble I may cause you.

Sincerely,

Miguel M. Soriano Lopez

May 26, 1983

Dear Editor:

I would appreciate it if you would publish the enclosed **announcement** of the availability of the Edison System Report entitled "Programming a Personal Computer" in the *Pascal News*.

Yours sincerely,

Per Brinch Hansen
Henry Salvatori Professor of Computer Science
University of Southern California
University Park
Los Angeles, CA 90089

Pascal & Modula 2

Press Release

FOR IMMEDIATE RELEASE: October 11, 1983

Cleveland, Ohio: The ten year old publication, Pascal News is changing its format and name to Pascal and Modula 2, according to publisher, Charles Gaffney.

For those unfamiliar with computer terminology, Pascal is a small and general purpose computer programming language, originally designed 10 years ago by Professor Niklaus Wirth of Switzerland as a teaching language. Because Pascal is easy to learn and read, and can be efficiently translated by computers, it was adapted for use in business. However, it does have certain restrictions. To meet the increased demand for an all purpose programming language, Wirth designed Modula 2. The structure of Pascal is included in Modula 2, affording simple and quick transition for programmers.

Pascal News was originally established to be a forum of correspondence for the Pascal Users Group (PUG). Because of the close linkage between these languages, and the rapid growth in the day to day use of computers, Gaffney expanded the publication to include articles and correspondence about Modula 2.

"Modula 2 is a newer language on the cutting edge of computer science," said Gaffney. "Its design allows Pascal to settle into original standards, and removes the pressure to be all things to all people."

Modula 2, sold only under license, will be protected from incompatible revision. The new availability of Modula 2 from at least 4 vendors demands a forum for new users as well as Pascal users. Pascal and Modula 2, sponsoring the Pascal Users Group and organizing the Modula 2 Users Group, will provide that forum.

Pascal News has over 4,000 subscribers in 41 countries, according to Gaffney who is confident the new publication will continue to serve these user groups. Pascal and Modula 2 will provide application software, software tools, articles on programming philosophy, the use of Pascal as a teaching tool, the promotion and application of each language, as well as an important open forum where users contribute informal correspondence of general interest to the group.

Two Pascal Devices

By Harley Flanders

Florida Atlantic University

1. THE FORWARD DECLARATION

It was brought to my attention by H. S. Wilf that the reserved word **forward** is not necessarily included in all versions of Pascal. If we examine Jensen and Wirth [1], we find **forward** discussed on page 82 of the ***User Manual***, but not in Appendix C, *Syntax*, pages 110-115, nor in the railroad diagrams, pages 116-118; however, in Appendix E, *Error Number Summary*, page 120, while nowhere in the ***Report***, pages 133-167.

Suppose we have a program in which several procedures call each other recursively. The usual way to handle this is via a series of **forward** declarations. It is possible to accomplish the same thing without using **forward** at all. Suppose, for example, that we have the declarations

```
procedure  B;  forward;

procedure  C;  forward;

Procedure  A;
   <declarations>
   begin
   <statements  A1>;  B;  <statements  A2>
   end;

procedure  B;
   <declarations>
   begin
   <statements  B1>;  C;  <statements  B2>
   end;

procedure  C;
   <declarations>
   begin
   <statements C1>;  A;  <statements C2>
   end;
```

The following single procedure declaration does the same.

```
var  CONTROL: Char;

procedure  ABC;
   <declarations>
   begin
   case  CONTROL  of
      'A':  begin  <statements  A1>;
            CONTROL := B;  ABC;
            <statements  A2>  end;
      'B':  begin  <statements  B1>;
            CONTROL := C;  ABC;
            <statements  B2>  end;
      'C':  begin  <statements  C1>;
            CONTROL := A;  ABC;
            <statements  C2>  end
      end   { case }
   end;     { ABC }
```

The statement calling ABC must initialize CONTROL to the first entry value. Clearly, many variations on this theme are possible.

2. THE EXIT PROCEDURE

UCSD Pascal restricts the **goto** statement so it only allows jumps within a block. Hence **goto** cannot be used to exit a nested sequence of procedures. The Exit procedure was introduced into UCSD to make up for this shortcoming; however, it fails to do the job if the nested sequence happens to include recursive calls of a procedure. This can be quite inconvenient at times. Suppose, for example, one is searching an array, and the search proceeds by testing an element then, if unsuccessful, it partitions the array and tests the pieces recursively. Of course, once the sought element is found, the search should be stopped. But the search procedure may be deeply nested at that time.

A clumsy way out is to introduce a Boolean variable FOUND and rewrite the body of the search procedure as follows:

```
procedure  SEARCH;
   <declarations>
   begin
   if  not FOUND  then  <statements of  SEARCH>
   end;
```

The (external) call of SEARCH must be replaced by

FOUND := False; SEARCH

This is costly because it adds an extra test to each call of SEARCH. The following is an alternative using Exit. Assume that we are searching for X and that A denotes a test value.

```
procedure  DUMMY;
   procedure  SEARCH;
      begin
      <statements of  SEARCH>
      if  A = X  then  Exit(DUMMY)  else ... SEARCH
      end;
   begin  SEARCH  end;
```

Remember the UCSD rule is that Exit (PROC) is a jump to immediately after the **most recent** call of PROC, passing through all more recently called procedures along the way, and doing some incidental housekeeping, like closing files those procedures opened.

Reference

1. K. Jensen and N. Wirth, ***Pascal User Manual and Report***, 2/e, Springer-Verlag, 1978.

"Zuse"

A Compiler Writer

ZUSE USER'S MANUAL
Unix Implementation
Version 1.0

by
Arthur Pyster

Department of Computer Science
University of California at Santa Barbara
Santa Barbara, CA 93106

This work was in part supported by a grant from the Instructional Development Program of UC Santa Barbara.

1.0 Introduction

1.1 Background

Zuse is a translator writing system written in Pascal which produces translators which are themselves written in Pascal. It is quite simple to use and alleviates much of the tedium inherent in writing a translator from scratch. It is named after Konrad Zuse whose visionary work on the programming language Plankalkul in the mid 1940s should be an inspiration to everyone.

This user's manual assumes that you are already familiar with the principles of translator writing and syntax-directed translation. Such terms as "BNF grammar" and "LL(1) parser" are used freely without explanation. If you lack this background, you should refer to one of the standard texts on translator writing (Aho and Ullman, "Principles of Compiler Design", Addison-Wesley 1977; Lewis, Rosenkrantz, and Stearns, "Compiler Design Theory", Addison-Wesley 1976; Pyster, "Compiler Design and Construction", Van Nostrand Reinhold 1980).

Zuse has been designed to be highly portable across different Pascal implementations. Only a handful of lines of code have been written using implementation-dependent features; e.g., Zuse presumes that type char is the ascii character set. This manual describes Zuse as it is implemented on Unix. A separate document describing any deviations from this manual should be obtained from whoever is responsible for Zuse's installation at your computer center.

1.2 How to use Zuse

Zuse actually consists of two programs: generate.o - an executable program which accepts a translation grammar as input and generates several files which will be needed for the translator eventually produced; and skeleton.p - a partial translator written in Pascal which must be augmented by files produced both by you and by generate.o in order to become a complete Pascal program which can be compiled. Figure 1 shows the creation of a translator, skeleton.o. Figure 2 shows the execution of that translator to translate source string x. When creating a translator, all of the files except for the user-defined translation grammar, and LLsup.i are part of or generated by Zuse. When executing the translator, only the source string which is to be translated needs to be provided by the user. LLgram is produced by Zuse. The particular manner in which Zuse creates the necessary files is described in later sections of this manual.

Zuse's two programs are used in the sequence listed below to create a translator:

1) Write a context-free grammar which specifies the syntax of the source language. The grammar should be prepared as a file using any text editor. It can be stored under any file name. Because of the parsing method supported by Zuse, the grammar must be an extended LL(1) grammar. The permissible extensions are specified in later sections. This grammar will eventually be used to produce a top-down left-to-right parser.

2) Embed action code into the grammar which specifies the steps to be taken by the translator during parsing. A language definition with both a syntactic specification and action code is referred to as a "translation grammar". The translation grammar specifies the actions to be taken during the parsing of a string. These actions produce a translation of the source string.

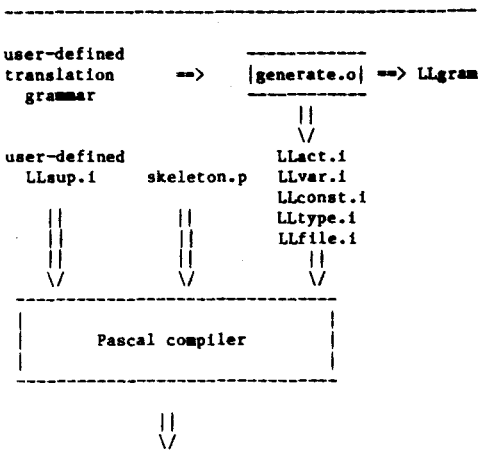


Fig. 1 Creating a Translator

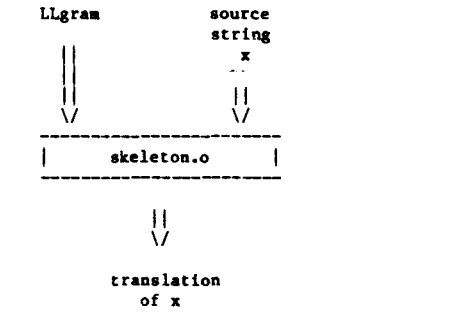


Fig. 2 Executing a Translator

3) Prepare Pascal code which defines all supporting routines which will be called by the action code, including the lexical analyzer -- LLNextToken. These routines, which will be needed when skeleton.p is compiled, should be stored in the file LLsup.i.

4) Execute generate.o on the translation grammar just prepared. Generate.o expects the grammar to come from the standard input file so you must redirect your file:

```
generate.o < MyFile
```

Any errors it uncovers will be reported on the standard output file. Generate.o creates several Pascal text files which must be embedded into skeleton.p along with LLsup.i. Skeleton.p has been peppered with "include" directives so that these files will automatically be included during its compilation.

One additional file will be created - LLgram. It is a modified version of the grammar specified in step 2, which has been compressed to facilitate use by skeleton.o, the compiled form of skeleton.p. Skeleton.o reads LLgram before translation begins.

5) Compile skeleton.p. If you are using the Pascal compiler "pi", then just type:

```
pi skeleton.p; mv obj skeleton.o
```

If you have another Pascal compiler, then invoke it in the standard way. Assuming there were no errors in the specifica-

tion of the grammar and the definition of the routines in LLsup.i, the resulting object code, skeleton.o, will be a working translator. If there are errors in the grammar, then all the steps just listed will have to be repeated. If the translation grammar is correct, but the support routines are incorrect, then they must be corrected and skeleton.p must be recompiled. You must repeat this process until you are satisfied that the translator is correct.

6) Add error processing capabilities. Skeleton.o will detect any syntactic error in a source string. The detection and processing of semantic errors is left entirely in your hands as the translator writer. Skeleton.o's default error recovery is to terminate the translation, causing an appropriate message to be printed. Zuse also supports optional sophisticated error processing facilities which allow you to specify a number of possible recoveries when a parsing error is detected. These are explained in section 7.

1.3 Manual organization

This user manual itself is organized into sections based on the steps just listed in section 1.2. A running example is used throughout to illustrate concepts as they are introduced.

For further information about Zuse, the reader is urged to contact the author at the Department of Computer Science, University of California, Santa Barbara, California 93106, phone: (805) 961-3236 or x-4321.

2.0 Write A Context-Free Grammar

The first step in generating a translator is to prepare a context-free grammar which specifies the syntax of the language to be translated. A context-free grammar G classically has four components:

- nonterminals - the grammatical categories
- terminals - the alphabet of the language
- axiom - a nonterminal which begins all derivations
- productions - the rewriting rules

In fact, Zuse uses a somewhat different structure dictated by the need to distinguish between different types of terminals. Two classes of terminals will be defined: "groups" and "literals". Details about them are presented in section 2.3.

The grammar specification is divided into declarations and productions. Each vocabulary symbol used in the grammar must be declared, indicating what type of symbol it is. Just as in Pascal, a symbol must be declared before it can be referenced. But before describing how to write the grammar, the language which will be the basis for a running example throughout the manual will be described.

2.1 The language EXPRESSIONS

At this point the language which will serve as the running example throughout the manual will be introduced. The language EXPRESSIONS contains possibly empty sequences of arithmetic expressions terminated by semicolons. Its grammar is called G_EXPRESSIONS. The operators are four basic arithmetic operations on integer values:

```
+ - * /
```

Operands are unsigned integer literals. Some sample strings of EXPRESSIONS are:

```
(3+4)*(5-6); 4-6; 4 / 6;

12 / 3;

3;
```

The arithmetic operators follow common precedence rules; i.e., * and / are performed before + and -, with operations being performed left to right within precedence. Parenthesization can override any default precedence.

The translation of a member of EXPRESSIONS will be its numeric value. Hence, the translator in this case will be an expression interpreter. For the three examples above, the translations are:

```
-7  -2  0

 4

 3
```

You will type in an expression from the terminal, and your interpreter will print its value immediately below your input. If you type an invalid expression, then an appropriate error message will be printed just below the incorrect input. Hence, in this case there really is no "object code" for the translation, since the display of the expression's value on the terminal is the only desired output.

2.2 Lexical structure of Zuse grammars

2.2.1 Tokens

Zuse grammars have much the same lexical structure as Pascal programs. Within a grammar a "token" is a sequence of printable characters which has no embedded blanks, tabs, or newlines. In certain cases it may be necessary to surround a token with single quotes:

```
' .. token .. '
```

because the unquoted token is part of the metalanguage used by Zuse to specify the grammar. For example, each production ends with a semicolon. Hence, it is impossible to have a literal semicolon as a terminal symbol on the right-hand side of a production unless it is surrounded by quotes:

```
X = ... ';' ...    ;
```

In other contexts where there is no ambiguity, the semicolon can be used without quotes.

Tokens may start in any column. Blanks, tabs, and newlines are token separators. Blank lines may be freely inserted anywhere in a grammar. For convenience in later discussion, the word "spacer" will be uniformly used to mean any positive number of blanks, tabs, and newlines.

Zuse is sensitive to upper and lower case letters. For example, the following three symbols could all be declared as distinct nonterminals in your grammar:

```
PROGRAM  program  PrOgRaM
```

so be very careful, especially if your Pascal compiler does not make such distinctions for identifiers declared in Pascal programs. Even if your compiler would treat the three symbols written above as the same identifier, Zuse will not.

The maximum permissible length of a symbol is 12. If you write a symbol longer than the permitted maximum, generate.o will print an error message and disgard all characters of that symbol beyond the maximum.

2.2.2 Comments

A comment may be inserted anywhere a spacer is allowed. In Zuse comments have the form:

```
(*   .. comment .. *)
```

This is one of the two formats for comments used in Pascal. The other form of Pascal comment, { .. }, is not allowed in Zuse because the curly brackets are used for other purposes as described later.

2.3 Declarations

Every symbol used in the grammar must be declared, although the order of declaration is not significant. A symbol can only be declared once. Zuse will tell you if you declare a symbol more than once or reference an undeclared symbol in a production.

The same basic format is used for all declarations:

```
%SPECIFIER  LIST_OF_SYMBOLS
```

3.0 Embed Action Routines

3.1 Format

Once the grammar has been defined, you must add action code which specifies how the translation will proceed. Although this step can be done while the grammar is being written, it is often more convenient and more reliable to first develop the grammar, embedding action code only after the grammar is complete.

Action code may appear anywhere among the list of vocabulary symbols on the right-hand side of a production. The code is actually a sequence of Pascal statements enclosed by curly brackets:

```
'{a' Pascal statements '}'
```

The first character after '{' must be an 'a' to indicate that it is action code. Other characters, discussed in section 7, are used for other types of code.

Rewriting the grammar productions for G_EXPRESSIONS to include action code yields:

```
Ax = ;   (* empty production *)

   = {a init} E {a writeln(popopand);}
     ';' Ax ;  (* note the use of a quoted ; *)

E = T E-list ;

E-list = Asop {a pushoptor($1.operator)} T
           {a r := popopand;
              l := popopand;
              popoptor(op);
              if op = '+' then
                 pushopand(l+r)
              else
                 pushopand(l-r)}
         E-list ;

       = ;    (* another empty production *)

T = P T-list ;

T-list = Mdop {a pushoptor($1.operator)} P
           {a r := popopand;
              l := popopand;
              popoptor(op);
              if op = '*' then
                 pushopand(l*r)
              else
                 pushopand(l div r)}
         T-list ;

       = ;

P = ( E ) ;
  = INTEGER {a pushopand($1.operand)} ;

Asop = + {a $0.operator := '+'};
     = - {a $0.operator := '-'};

Mdop = * {a $0.operator := '*'};
     = / {a $0.operator := '/'};
```

Several aspects of this revised grammar warrant explanation. For the moment ignore those strange looking identifiers which begin with "$". They refer to special variables which will be explained in section 3.5.

Action code specifies how the translation is to take place. All aspects of that specification are left in your hands. You can either write all of your code directly in the grammar or you can call separately declared procedures and functions from within the action code, or mix the two styles. The grammar above is written in a mixed style. The action code in the first alternative for T-list has two assignment statements followed by a procedure call, followed by an if-then-else statement. The action code for the second alternative of P has just a single procedure call. You must decide which support routines (as those user-defined routines called from action code are called) to define and what they will do. G_EXPRESSIONS has five support routines referenced in the action code:

```
init pushopand popopand pushoptor popoptor
```

The declaration of these five routines must be included in LLsup.i, which contains the support routines, before skeleton.p can be compiled. However, at this point you only need to know what these routines do so that you can properly write the action code.

The action code will map the infix expressions into two stacks — "opandstk" (operand stack) and "optorstk" (operator stack) -- where the operands and operators of the expression will be held, respectively. The algorithm to evaluate infix expressions using two stacks is quite standard, and it is assumed that you are familiar with it. The code defining these five functions is specified in section 4. "Init" initializes the two arrays which represent the stacks and two integer variables, "topopand" and "topoptor", which point to the top of "opandstk" and "optorstk", respectively. "Popopand" and "pushopand" pop and push elements onto opandstk, while "popoptor" and "pushoptor" are the analogous routines for optorstk.

3.2 Variable, constant, type, and file declarations

3.2.1 Variable declarations

Two variables are referenced in the action code -- "l" and "r". These variables hold temporary values of the left and right operands of some operator. Because Pascal requires the declaration of each variable which is referenced, there must be some provision for declaring these variables. Note that Zuse itself cannot have already declared them because the particular set of variables needed for the action routines will vary from translator to translator. To permit user-declared variables, an additional declaration type is permitted in grammars in the declaration section:

```
%v  Pascal variable declarations
```

If a 'v' (or 'V') specifier is used in a declaration, then the variable declaration is copied verbatim into file "LLvar.i". Note that each declaration ends with a semicolon. If more than one variable is declared, they are copied in the order in which they appear in the grammar.

G_EXPRESSIONS needs several declarations:

```
%v op: char;  (* the operator popped from
                 optorstk *)

%v toptorstk: integer; (* top of optorstk *)
   topandstk: integer; (* top of opandstk *)

%v opandstk: array[1..stksize] of integer;
   optorstk: array[1..stksize] of char;

%v l,r: integer;  (* temps *)
```

These six variable declarations can appear anywhere in the declaration section. Since the order of variable declarations is not important in Pascal, they can be declared in any order. Note that several variables can be declared using a single "%v" as in the declaration of toptorstk and topandstk.

3.2.2 Constant declarations

The integer constant "stksize" is referenced in the declaration of opandstk and optorstk above. This constant and any others which you need for your action code can be specified through a constant declaration in the grammar. A 'c' or 'C' is used to specify a constant declaration. G_EXPRESSIONS needs:

```
%c stksize = 20;  (* maximum depth of stk *)
```

Constant declarations will be placed into the file "LLconst.i" for inclusion into skeleton.p. They will appear in LLconst.i in the same order as they appear in your grammar. It is your responsibility to guarantee that a constant is defined before it is referenced.

3.2.3 Type declarations

Although they are not needed for this example, you can declare new types using the 't' or 'T' specifier. For example, if we wanted our interpretor to only work on nonnegative numbers, we might declare:

```
%t NonNegative = 0..maxint;
```