DECUS
PROGRAM LIBRARY

DECUS NO.	8-934 (EC)
TITLE	PASCAL OS/8
AUTHOR	JOHN T. EASTON
COMPANY	UNIVERSITY OF MINNESOTA MINNEAPOLIS, MN
DATE	FEBRUARY 1984
SOURCE LANGUAGE	PASCAL, PDP-8 MACREL-LINK

new
8-934

PASCAL OS/8

Version: V1-Ø-F, February 1984

Author: John T. Easton, University of Minnesota, Minneapolis, MN

Operating System: OS/8 V3

Source Language: PASCAL, PDP-8 MACREL-LINK

Memory Required: 24KW

PASCAL-OS/8 is a software system that implements the programming language PASCAL for the PDP-8 family of minicomputers running the OS/8 operating system. PASCAL-OS/8 consists of a compiler (written in PASCAL), a run-time-system with interpreter, and numerous utility programs and example programs written in PASCAL. Documentation is in printed form, about 135 pages long. PASCAL-OS/8 adheres to the ISO standard for PASCAL. It installs very simply. The implementation represents several years of work. It has been run on both PDP-8E and PDP-12. It is implemented with no PDP-8E dependencies.

Major limitations are: identifiers are distinguished on only the first eight characters; 24k memory is needed to compile; there is no library mechanism or assembly language linkage mechanism.

Features include: up to 32k memory is utilized; Post-Mortem-Display (PMD), execution error traceback is automatic; large programs may be segmented to fit available memory; several internal device handlers allow flexible interactive Input/Output; an FPP is used if present but is not required; it is compatible with two-page system handlers; it can run well under OS/8 BATCH.

Extensions include: flexible facilities for accessing OS/8 files, including direct (random or indexed) access files; three-way packed OS/8 character files may be acessed as type TEXT, file of ASCII, or file of EightBit; Date routine; Execute(string) calls CCL; Halt (message) aborts execution; otherwise in case statement; others.

Performance is greatly enhanced if the OS/8 system has a fast disk such as an RK05, and/or if a full 32k memory is available. Performance is roughly similar to OS/8 Basic. Installation consists of copying some files to SYS:. Test sites report that PASCAL-OS/8 is complete, easy to install, and easy to use.

Table of Contents

1. Preface

1.1. Overview

Welcome! This document describes Pascal-OS/8, a software system that implements the programming language Pascal for PDP8 minicomputers running the OS/8 operating system. The minimal machine required to support Pascal-OS/8 is described in section 3.4.

Pascal is a small, general-purpose language possessing algorithmic structures and data structures to aid systematic programming. Pascal is now implemented on most commercial computers and is widely used in teaching, commerce, industry and the military. This reference manual does not intend to teach Pascal.

At this writing (September 1983) the International Organization for Standardization (ISO) has adopted an International Standard for Pascal (see "Specification..." in appendix R). Numerous national organizations (e.g., ANSI, AFNOR, DIN, BSI) have participated in this effort. Pascal-OS/8 conforms closely to this standard.

1.2. How to use this document

Section 2 (User Guide) informally explains how to compile, run, and debug Pascal programs. The User Guide provides the essential manual for users who mainly work with standard conforming Pascal programs. Only minimal knowledge of Pascal and of OS/8 is assumed. Knowledge of an OS/8 text editor is assumed. Section 3 describes further features of the run-time system and compiler, extensions to the standard language, and what hardware and software requirements there are for using Pascal-OS/8.

Section 5 (Installation and Support) is a guide to installing the Pascal-OS/8 system, which is very easy, and includes the topic of configuring the system to fit your installation (section 5.4). It also describes what to do if you find errors in Pascal-OS/8 (section 5.6).

The remaining sections are appendices (identified by letters instead of numbers) providing additional details, including explanation of run-time-system error messages (appendix E), and a listing of compiler error messages that go with the error numbers on the compiler listing (appendix F). Appendix H, in particular, can be very useful as a brief guide to users of Pascal-OS/8. You may wish to copy the two pages of appendix H and make them available as a help file or hand-out. See Appendix R for some Pascal textbooks. The results of testing Pascal-OS/8 upon the Pascal Validation Suite is described in Appendix V.

1.3. Acknowledgements

We wish to express our thanks to the many people whose support and prior work has made possible our own.

Philip Voxland, as Acting Director of SSRFC, has generously provided resources, has offered suggestions and ideas, and has reviewed innumerable pieces of writing.

Nori, Ammann, Jensen, Nageli and Jacobi, at the Institut fur Informatik at Zurich, produced the excellent portable Pascal compiler which has been the basis for our implementation.

Jonathan R. Gross envisioned and began the initial implementation (the virtual-memory interpreter), and provided insight, ideas, and criticism that helped us in subsequent design work. Without Jon's early work in this area, we would probably not have attempted such a large task.

John Strait, a former maintainer of the CDC Cyber compiler, has participated in numerous discussions and has traded compiler modifications with us. We hope that as a result, both the PDP-8 compiler and the CDC-6000 compiler have been improved.

Jim Van Zee proved himself a PDP8 wizard (again) by diagnosing many bugs and deficiencies during field testing, and making many useful suggestions.

Wally Kalinowski (Aerospace Corp) and his co-workers have enthusiastically urged us to get this implementation out the door and made available to the PDP8 community.

Geoffrey Chase (Pascal-S support) encouraged us to resume work with his personal visit and interest in the project.

And, last but not least, our long-time associate and friend Andy Mickel has provided us with a flood of information and moral support.

2. Users Guide -- Standard Pascal-OS/8

This Users Guide informally explains how to use Pascal-OS/8 from the point of view of a programmer of Standard Pascal. It does not attempt to explain the language Pascal. Section 3 discusses extensions and adaptations available.

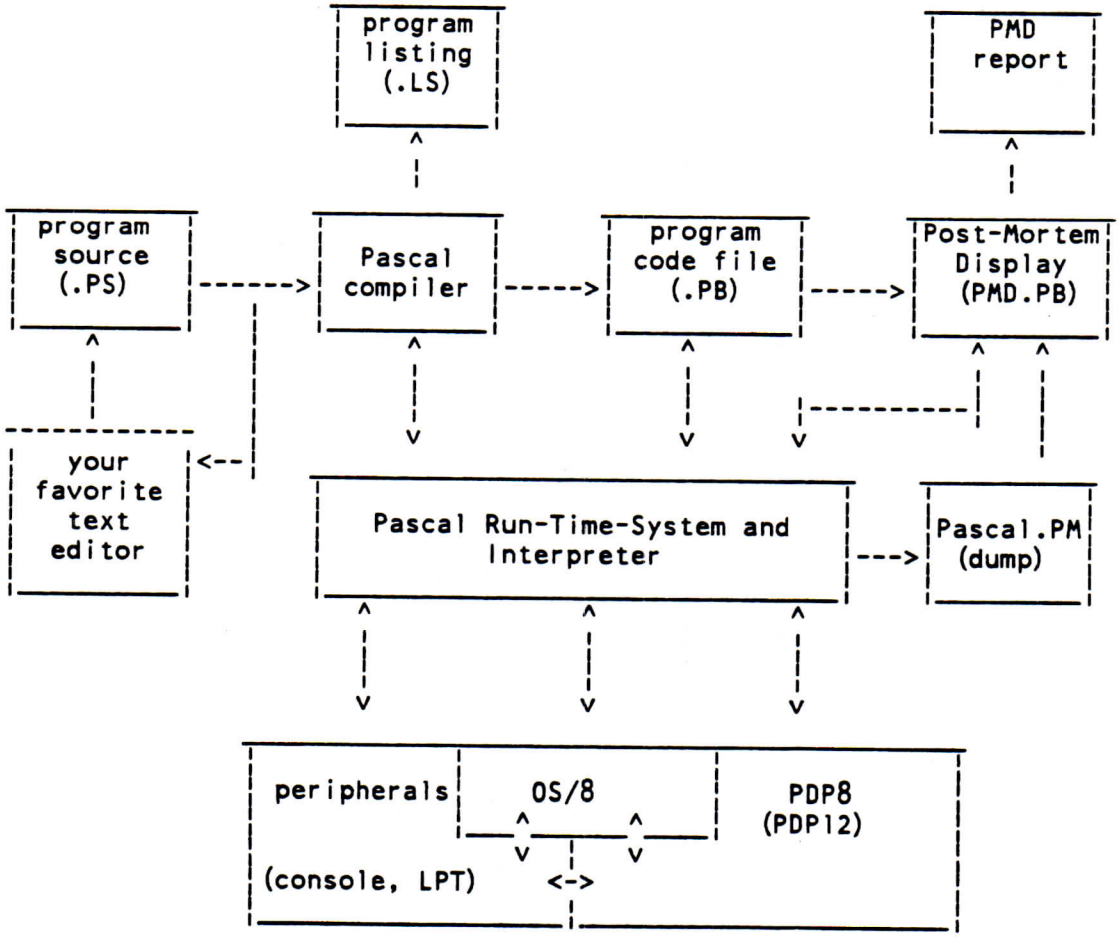
The Pascal-OS/8 system is intended to be simple and easy to use. Single command statements typed at the OS/8 console will compile a program or run a program. The system attempts to report in some detail when something goes wrong, such as a value out of range.

The Pascal-OS/8 system consists of a Pascal compiler, with its error message text file, and a Run-Time System, with its post-error program state report utility PMD (Post Mortem Display, see T.1.6).

In addition, the utility program IDMAP (see T.1.3) can make a program listing with line numbers and a cross reference map of identifiers.

The compiler, PMD, IDMAP and all the utilities are Pascal programs, and use the same run-time system facilities available to all programs.

Pascal OS/8 System Model



2.1. Running Programs

To run a Pascal-OS/8 program, the compiled form of the program (a .PB file) must be available (see section 2.2 for compiling instructions). Also the run-time system is necessary to run all programs, including the compiler.

2.1.1. Specifying Load Files

In general, the command statement:

```
.R P,program
```

typed in response to a '.' from the OS/8 Keyboard Monitor will run a program. The line that the user types is referred to as a command statement, and may also appear in an OS/8 BATCH job. This line specifies multiple items:

-- an OS/8 Keyboard Monitor command ('.R')

-- the name of the Pascal-OS/8 run-time system, which is an OS/8 save file to be run ('SYS:P.SV').

-- the name of the Pascal-OS/8 executable file ('program' to be loaded, also referred to as a 'load file').

Two additional kinds of information may be given on the control statement:

-- the names of any files to be made accessible from OS/8 to the Pascal-OS/8 program via program heading parameters.

-- a comment portion of the command statement: everything after the first slash ('/') on the command statement; some programs, such as the compiler, interpret things after the slash as options, and so restrict what appears; other programs ignore what appears here.

A full example of the command statement is:

```
.R P,SYMREF,SYS:PASCAL.TX,LP: / Hi There!
 | | |      |             |   ^...comment field
 | | |      ^.............^..program file parameters
 | | ^...Pascal-OS/8 executable program
 | ^...Pascal-OS/8 run-time system
 ^...OS/8 Keyboard Monitor command
```

The program heading for the program SYMREF is:

```
Program Symref(input,output);
```

It is possible for the program to read the command statement that started it up by binding a text file to the internal read-only character-device handler 'CS:'. See section 3.2.2.

Pascal-OS/8 does <u>not</u> use Command Decoder. In particular, the characters

'<' and '_' have no meaning on a Pascal-OS/8 command statement. See also section 2.1.3.2.

2.1.2. Accessing the Run-Time System

There are three methods of access to the run-time system: (1) directly called from SYS:; (2) directly called from co-SYS: resident; (3) via a Save File header.

2.1.2.1. SYS: RTS

There are two modes for executing the Pascal-OS/8 run-time system: the single-statement mode, as in the examples above, and the job-step mode. The single-statement mode is expressed all on one command line, and after program completion, control returns to the OS/8 Keyboard Monitor. The job-step mode allows the run-time system to execute more than one program, one after another. The job-step mode is entered by running the RTS with no parameters, as in:

```
.R P
%first-program
%second-program
%...
```

The '%' is a prompt character serving the same purpose as the OS/8 Command Decoder's '*': a command statement is requested.

A typical response to the prompt would be:

```
%SYS:PASCAL,MYPROG
```

which would run the compiler (SYS:PASCAL.PB), which in turn would attempt to compile DSK:MYPROG.PS (see section 2.2 for details on compiling). When the executable program (in this example SYS:PASCAL.PB) completed, control would return to the RTS in job-step mode, another prompt character would be printed, and a new program could be executed, as in:

```
.R P
%SYS:PASCAL,MYPROG
%MYPROG
%$
```

To get out of job-step mode and return control to OS/8, you may:

-- type control-C at any time;

-- type an ESCAPE in response to the '%' prompt (as above);

-- terminate the last command statement with an ESCAPE.

2.1.2.2. Non-SYS: RTS

In the examples so far, it has been assumed that the run-time system resides on the OS/8 system device, SYS:. The RTS may also reside on a device that is co-resident with SYS:, such as the 'B' side of SYS: on an RK05 disk, which is frequently the 'DSK:' device. In such a case, the command form:

```
.RU RKB0:P,program,files
```

will access the RTS equally well. Similarly, the job step mode:

```
.RU RKB0:P
%program,files
```

is possible. The run-time system cannot be executed from a device that is not SYS: or not co-resident with SYS:.

2.1.2.3. Save File Header

One other way of accessing the run-time system exists: the compiled program may be converted into an OS/8 Save File and then run directly, without explicitly naming the run-time system. See section 3.2.3 for details.

2.1.3. OS/8 File Names

OS/8 file names are those names referring to an OS/8 file or device (such as 'PROG.PS' or 'MTA0:'), or to an internal device handler (such as 'TT:' or 'LP:').

OS/8 file names are used to refer to three different categories of OS/8 files: (1) Pascal-OS/8 executable load files, (2) OS/8 files to be bound to program file variables, and (3) other usages, such as the OS/8 file name of the Pascal-OS/8 run-time system.

2.1.3.1. File Name Construction

OS/8 file names used in Pascal-OS/8 are compatible in construction with file names for the OS/8 Command Decoder and are made up of three parts.

1. device name, ending in ':' or ';', with 4 significant characters.

2. file name, with 6 significant characters.

3. extension, starting with '.', with 2 significant characters.

Device names, file names and extensions are made up of one or more letters and digits, in any order. There are no 'wild card' characters.

Each of the parts may occur independently, in which case it overrides defaults already defined in the file binding process. See section 2.1.4 for a discussion on file binding.

2.1.3.2. Read Mode

Read mode versus write mode files are in no way determined by the command statement position of a file or any command statement syntax (as in the OS/8 command decoder). Read or write mode of a file is determined by the action of a program in executing either a Reset or a Rewrite of a program file variable which is bound to an OS/8 file.

It is possible to specify read-only permission for an OS/8 file that is to be bound to a program file variable, to ensure the program does not write on the file. To specify read-only mode, do:

```
file[r]
```

The 'r' in the square bracket sets read-only mode for the file variable; this attribute lasts until the file variable is re-bound, or the program finishes execution. This construction is not accepted for load files (program to be executed); these are only used in read-mode anyway.

2.1.4. Binding Pascal-OS/8 Files to Devices and OS/8 Files

Binding is the process by which an OS/8 file is coupled with a Pascal-OS/8 file variable, providing data to be read in the case of a Reset file variable, or a data sink in the case of a Rewrite file variable.

Binding occurs after the command statement is given. Each OS/8 file name is coupled with a program parameter in an order-dependent process. Given:

```
Program John ( Data, Result );
var Data, Result: text;
begin
  Reset(Data);
  Rewrite(Result);
  ...
end.
```

and a command statement:

```
.R P,JOHN              --or--          .R P,JOHN,DATA,RESULT
```

the file variable 'Data' will be bound to the OS/8 file 'DSK:DATA.'. 'Reset(Data)' will attempt to lookup the OS/8 file. If the OS/8 file does not exist (or is empty), 'eof(Data)' will be true; otherwise data from 'DSK:DATA.' will be available through the file variable 'Data'. Further:

```
.R P,JOHN,SYS:OTHER
```

will bind the file variable 'Data' to an OS/8 file name of 'SYS:OTHER.'.

Program parameters get a default binding pattern. For all files except 'input' and 'output', the default name to be bound is formed from the file name itself, using DSK: as a device, and the first 6 characters of the variable identifier as the file name, and '.' as the extension. Files 'input' and 'output' are made special cases by defaulting them to the device 'IO:', which provides support for interactive Input/Output (or a message stream under BATCH that tries to track the BATCH output device). The name of 'Input' or 'Output' is concatenated to the device 'IO:'. There are no file name extensions in the default binding patterns.

See section 3.2.1 for a discussion of extensions for the file binding process.

2.1.5. Run-Time Error Messages

The run-time system issues informative messages and error messages, which are discussed individually in appendix E. Most error messages are of the form:

```
'*** Error' message
```

where '***' marks a run-time-error, and 'message' is an explanation of the error that is being reported. If a file is involved in the error (such as an attempt to read past eof), the file name of the OS/8 file involved will also be reported:

```
'*** Error' message
'*** file:' name
```

If a program is executing with PMD enabled (default mode) then the run-time-system will pause for a short time while it writes a '.PM' dump file containing the state of the program, and executes 'SYS:PMD.PB' to perform an analysis.

2.1.6 Post-Mortem Display (PMD)

Debugging Pascal-OS/8 programs is made somewhat easier by a symbolic display of simple variables and of the current execution path. This symbolic display is called Post-Mortem Display (PMD). PMD can be invoked by:

-- control-D pressed upon the console; or
-- a program execution error

The execution error may be any of several things, such as a value out of range, an Input/Output error, or an internal run-time-system error. All will result in a PMD if the program was still running at the time of the error.

PMD will print out the location of the program at the time of the error, the procedure and function call path, and the values of simple variables in procedures and functions that were active, as well as in the main program. When PMD is done, it will abort a BATCH job if running under BATCH, or will return control to OS/8 (or P8RTS if in job-step mode).

The PMD is printed upon the system console. If you need to get a copy of the PMD output printed, you may re-run PMD with the statement:

```
.R P,SYS:PMD
```

This uses the last SYS:PASCAL.PM file and the .PB file named within it to reproduce the PMD report. Each time the run-time system processes an error that calls for PMD, the contents of the file SYS:PASCAL.PM will change. SYS:PASCAL.PM need not pre-exist in order for PMD to function correctly upon CTRL-D or an error.

PMD processing of a given program can be inhibited by the use of the compiler 'P-' option when the program is compiled (see section 3.1.1).

2.2. Compiling Programs.

Compiling is the process of converting a Pascal source file into a form suitable for execution on the Pascal-OS/8 system. This form is a Pascal-OS/8 code file (with a .PB file name extension).

The compiler is a descendant of a portable Pascal compiler known as P4. It is a one-pass, recursive-descent compiler which is itself entirely written in Pascal (adapted to Pascal-OS/8).

Since Pascal-OS/8 does not support separate compilation or libraries, the entire program is compiled in one operation, and the compilation product is immediately executable (there is no linkage step). The compiler can also produce an optional listing.

2.2.1. Running the Compiler

A Pascal program may be compiled with the simple command statement:

```
.R P,SYS:PASCAL,program
```

'SYS:PASCAL' is the compiler (it may reside on SYS:, DSK: or some other device on your system). 'Program' is the OS/8 file name of a Pascal source, the default filename extension being '.PS'. The result of this statement will be:

-- if there are no compilation errors, the listing will say only "NO ERRORS DETECTED", and an executable code file will be produced on the same device as the source program, with the same name as the source program, but with a '.PB' extension.

-- if there are compilation errors, or if run-time errors such as memory or disk overflow occur, no code file will be produced (the code file is always pre-deleted). The lines of the program that are in error will be listed, with error numbers pointing at the language element in error, and there will be a summary of error-number descriptions at the end of the listings.

When it starts execution, the compiler will pre-delete the code file. If there are compilation errors (or run-time errors), the new code file either will not exist or will be incomplete, and the run-time system will reject it. If there are no compilation errors, the code file produced will be marked complete, and will be executable. This design eliminates confusion as to which code file version is left around; if the file exists and if it is accepted by the run-time system, the last compilation had no errors detected.
1

2.2.2. Binding Compiler Files

Binding is the process of associating OS/8 files with the compiler files. The compiler has three program parameter files:

```
program PascalCompiler ( source, listing, code);
```

and does some processing on the file names using the extension GetFN and file name extensions for Reset and Rewrite in order to provide the automatic naming that make it easy to use. (These extensions are discussed in section 3.1.2.)

2.2.2.1. Source File

A source file name must be specified as the first file parameter. There is no default source file name. The default extension is '.PS' (for 'Pascal Source'). The default device is 'DSK:'.

The device and filename of the source are used in the code file name construction (see 2.2.2.3).

2.2.2.2. Listing File

By default, the compiler lists only those lines of the source file on which it detects errors. See 2.2.3 for a description of the contents of the listing file.

To obtain a compiler-produced listing of your full program, you may:

-- use the compiler 'L' option on the command statement, as in:

```
.P,SYS:PASCAL,program/L+
```

The listing will be on IO:

-- use the compiler 'L' option within your program text. See section 3.1.1.

-- specify an output device or file as a second file parameter to the compiler on its command statement, as in:

```
.R P,SYS:PASCAL,program,LP:
```

The listing produced on LP: will include any error messages.

Also, to get only error messages displayed on a different file or device, use:

```
.R P,SYS:PASCAL,program,listing/L-
```

2.2.2.3. Code File

The code file name is produced automatically, using the source device and file name, and using a '.PB' extension (for PascalBinary). The code file name may also be given explicitly, as in:

```
.R P,SYS:PASCAL,program,,SYS:TARGET
```

The code file must always be compiled on a file-structured device. If the code file name is allowed to default, the source file must be on a file-structured device.

2.2.3. Compiler Listing

See section 2.2.2.2 for how to produce a compiler listing.
A compiler-produced program listing has four features:

(1) a line number (decimal) column;
(2) a location counter (octal) column;
(3) an asterisk ("*") immediately after (2) if the last source line did not close a comment;
(4) the image of the program line.

The location counter displays one of two things: (a) the relative stack location of a variable, displayed during parsing of declarations, or (b) the code length of a program, procedure or function, displayed during compilation of a procedure or function block. Actually, the code length is an offset relative to the start of the current code segment. The allocations for a line of source appears in the second (octal) column of the next line of the listing.

Figure 2.2.3-1 Example Listing

```
P8COM V1-0-F F

    1 000002 program jte(Input,result);
    2 000044 { example program to demonstrate
    3 000044*  Pascal-OS/8 listing }
    4 000044
    5 000044 const
    6 000044   a = 1;
    7 000044
    8 000044 type
    9 000044   natural = 0..maxint;
   10 000044   positive = 1..maxint;
   11 000044
   12 000044 var
   13 000044   result  : text;
   14 000104   counter : integer;
   15 000106   done    : boolean;
   16 000107   factor  : real;
   17 000112
   18 000112   procedure P;
   19 000002     var
   20 000002       ch : char;
   21 000003       i  : integer;
   22 000005       f  : boolean;
   23 000006   begin { P }
   24 000007     i := 0;
   25 000014     ch := '!';
   26 000020     f := true
   27 000021   end { P };
   28 000054
   29 000054 begin { jte }
   30 000147   P
   31 000150 end { jte }.
NO ERRORS DETECTED.
```

Figure 2.2.3-1 gives an example listing.
The listing starts with the version message from the compiler, "P8COM V1-0-F F". The "1-0-F" must match the version message from the run-time system.

The declaration sections are lines 1 thru 17 for the main program, and 18 thru 22 for the procedure P.

The executable section is lines 23 thru 31 for procedure P and the main program combined. For an unsegmented program, the code length at the end of the main program (line 31, 000150 in the example), together with the additional code generated for the PMD information, is the total code length of the program. Segmentation is discussed in section 3.1.3.

Line 1: 2 words are allocated for program setup.

Line 2: 2 more words have been allocated for program setup; 40 (base 8) words allocated for Input File Information Table (FIT). Input and Output, if named in the program heading, have their FITS allocated immediately. Other files have their FITS allocated when they are declared in the global VAR section.

Line 3: the "*" immediately after the second column of numbers signals the reader that there was an un-closed comment on the prior source line. This is allowed in Pascal, and this signal helps the user to locate un-intentional un-closed comments, which are not always syntactically discoverable by the compiler.

Lines 2 thru 11: no stack space (or code space) is allocated during parsing of const and type declarations.

Lines 13-14: 40 base 8 words are allocated for the FIT for file result.

Lines 14-15: an integer is allocated 2 words.

Lines 15-16: a boolean is allocated 1 word.

Lines 16-17: a real is allocated 3 words.

Lines 19: 2 words are allocated for a new stack frame for procedure P setup.

Lines 20-21: a character is allocated 1 word.

Lines 21-22: an integer is allocated 2 words.

Lines 22-23: a boolean is allocated 1 word.

Lines 24-27: executable code is being generated.

Line 28: PMD information for procedure P is allocated.

Lines 29-30: main program setup executable code creates FITs for input and result, and implements the automatic reset(input).

Lines 30-31: main program executable code.

The compiler options T (run-time tests) and P (PMD) affect allocation of code space; T- takes less space, and P- takes less space (independently). If P+, then V- or N- will take less space. See section 3.1.1 for a discussion of compiler options.

At the end of the compiler listing, the message:

"NO ERRORS DETECTED."

informs the user that compilation is complete, and no errors were diagnosed. The compiler spends a short time sorting and producing the code file, and then finishes execution.

If the listing is turned off, the compiler will still produce the first and last message:

```
P8COM V1-0-F F
 NO ERRORS DETECTED.
```

If the listing is turned on and the compiler detects errors, it will produce a pointer to the language element where the error was realized. Figure 2.2.3-2 shows the effect.

Figure 2.2.3-2 Example "L+" Listing with Errors

```
P8COM V1-0-F F

    1 000002 program Errors(data);
    2 000004 var
    3 000004   a : integer;
    4 000006
    5 000006 procedure
 UNDECLARED EXTERNAL FILE: DATA
                          p;
 ***********.........^172
    6 000002 begin
    7 000000   a := 2 * b
 ***********...........^104
    8 000000 end;
    9 000000
   10 000000 begin
   11 000000   p(a)
 ***********.....^126
   12 000000 end.

 ERRORS DETECTED.
 104: Identifier not declared.
 126: Number of parameters does not agree with declaration.
 172: Undeclared external file.
```

With the listing turned off, if any errors are detected, the compiler will list the source line with the error, and produce a pointer to

the language element where the error was realized.

Figure 2.2.3-3 Example "L-" Listing with Errors

```
P8COM V1-0-F F
 UNDECLARED EXTERNAL FILE: DATA
    5 000006 procedure p;
 ***********.........^172
    7 000000   a := 2 * b
 ***********...........^104
   11 000000   p(a)
 ***********.....^126

 ERRORS DETECTED.
 104: Identifier not declared.
 126: Number of parameters does not agree with declaration.
 172: Undeclared external file.
```

2.3. Other Topics for Using Pascal-OS/8

Collected here are various other topics for the user of Pascal-OS/8 who is not departing from standard Pascal, but wants to run programs smoothly on OS/8.

2.3.1. Interactive Input-Output

Interacting with a human from a Pascal-OS/8 program is easy, and requires no special-case coding. A program may write or (writeln) a prompting message, and then read items from the user. The files input and output are by default bound to the OS/8 console terminal, via the internal TT: handler. Input will read from the keyboard, and output will write to the console terminal display (printer or screen). A model of a simple interactive program follows.

```
program interactive(input,output);
  var ch:char;
begin
  write(output,'type first line:');
  while not eof(input) do
    begin
      while not eoln(input) do
        begin
          read(input,ch);
          write(output,ch)
        end;
      writeln(output);
      readln(input);
      write(output,'type another line:')
    end
end.
```

If this program is compiled and run with:

```
.R P,SYS:PASCAL,INTERACT
.R P,INTERACT
```

it will prompt you with "type first line:" and take a line of input, then echo the whole line. This is because the default handler for input (TT:) is line-buffered, with the normal OS/8 conventions for editing the input line. A more immediate interaction is possible with the TTO: handler, but there is no editing performed by the run-time system. Try:

```
.R P,INTERACT,TTO:
```

This assigns the TTO: handler for the file input, and by-passes the line-buffering routine in the run-time system. Non-printing characters will not be "visible" to this program, however, because the files input and output are of type text, which is defined to be printing ASCII characters only, and the run-time system filters out all non-printing characters for text files. See section 3.1.2 for a discussion of the extended character file types ASCII and EightBit. See the source for SvUtil for a complete example of a close-interacting program.

Any text file may be bound to the TT: handler (or TTO:) for interactive I/O. A reset text file will be bound to the keyboard, and a rewrite text file will be bound to the display. It is usually easier to use input and output, simply because that is conventional, and input is automatically reset, output automatically rewritten.

Other sections relevant are 3.2.1 File Bindings, 3.2.2 Internal Device Handlers, 3.1.2 Extensions to Standard Pascal. Interactive I/O is implemented with the "Lazy-I/O" technique of Saxe and Hisgen (see reference section R.3). The actual read of the text file f does not occur for get(f) (therefore does not occur when reset(f) is performed), but does occur when f^ is referenced, or when eof(f) or eoln(f) is executed.

2.3.2. Running Under OS/8 Batch

Pascal-OS/8 is designed to run under OS/8 Batch V7A. The Batch monitor is not kept in memory, so the ability to read the Batch stream (BAT: handler) is not available.

Pascal-OS/8 control statements are the same under Batch as under the Keyboard Monitor.

If the user presses control-C on the console or any active peripheral terminal, the Batch run will be completely shut down.

If any execution error, user halt, control-A, or control-D occurs, the execution error-traceback (PMD) will be produced (if applicable) and then the executing Batch job will be aborted with the regular Batch abort. The Batch monitor is loaded at this time in order to abort the current job. Batch will initiate a scan for the next $JOB card, and will resume normal Batch processing if it finds one, or else it will return control to the Keyboard Monitor.

3. User Guide -- Pascal-OS/8 Extended Topics & Fine Tuning

This part of the user manual tackles compiler and run-time system topics that go beyond standard Pascal usage, and considers topics for fine-tuning Pascal programs for residence on Pascal-OS/8.

3.1. Compiler.

The compiler (PASCAL.PB) supports fine-tuning efforts via compiler options, which do not affect the syntactic meaning of the program (section 3.1.1.) and it supports various extensions to the language via syntactic constructs (section 3.1.2.). Segmentation of the code for large programs is described at length in section 3.1.3.

3.1.1 Compiler Options

The user may control the operation of the compiler through the use of various options specified on the Pascal compiler control statement or within special comments. Compiler options that are specified within comments are designated by a dollar sign as the first character of the comment.

```
        (*$option-sequence any-comment*)
or
        {$option-sequence any-comment}
```

Example: (*$X+ allow Pascal-OS/8 extensions*)

Control statement options are specified after the '/', as in:

```
        .R P,SYS:PASCAL,program/option-sequence
```

Example: .R P,SYS:PASCAL,program/X+

The option-sequence is a series of individual options separated by commas. Each option consists of one letter (upper or lower case) designating the options, followed by the new value of the option setting. The value may be a "+" or "-" which turns some options on and off like switches. Alternately the value may be a decimal or octal (indicated by a post radix "B") integer for numeric options. The numeric options (B and W) can only appear in program comments.

The option-sequence is terminated by anything that is not an appropriate option letter or option value. Setting an option to an inappropriate value (e.g., setting a switch option to a numeric value) will cause option scanning to end. An error is issued when an invalid option syntax is found.

Because compiler options may be written anywhere in the program, it is possible to activate the options selectively over specific parts of the program (except I).

The following options are available.

Bn Buffer factor for files. This sets the file buffer size ranging from 1 to 512. For character files (text, file of char, file of EightBit, file of ASCII) this factor is in blocks (of 256 words); for non-character files, this factor is in file elements. For non-character files, the actual buffer size will be the Bn setting multiplied by the number of blocks needed to store each file element.

The buffer allocation factor determines the size of buffers allocated for file variables of a specific type. The buffer factor is an attribute of a type, and must be correctly set before the file type is declared. Note that the type text is "declared" when the program heading is compiled. Thus, to change the buffer size of files of type text, the B-option must be specified before the program heading. Also note that the buffer factor has no effect on indexed files; their buffers are always one file element long, taken up to a block boundary.

Example:

```
1  {$B4}program example(OUTPUT,A,F);
2  {$B10}
3  type othertype = file of integer;
4  var A:text;
5      F:othertype;
6  begin end.
```

In this program example, files OUTPUT and A are of type text and will have a buffer length of 4 blocks. File F is of an other type declared after a re-setting of the B option, and will have a buffer size of 10 elements. A setting of B4 can speed up text processing programs nicely. A setting of B1 will take the minimum space for the file input/output buffers. Default is B1.

I+,I- Interactive Input/Output. This option may be turned off to speed up text file processing a very small amount, but then the special "lazy-I/O" implementation of text files that allows interactive programming with no special program coding considerations is not used. Should always be left on for any program that needs to interact with people. This option affects all the text files in the program, and must be used before the program heading to be effective.
Default is I+.

L+,L- Listing of program by compiler. If L+, the compiler will list program lines. If L-, the compiler does not make a listing, except that lines with an error detected are listed, with a pointer to the error. The listing will be turned on for the entire line in which the L+ option appears.
Default is L-.

N+,N- Line number reporting within PMD. PMD can report by line numbers (if N+) or by program address (if N-). Setting N- eliminates the code necessary to keep the line number current

and therefore requires less memory to execute a program, and slightly speeds up execution. If N- and P+, any PMD report interpretation will require a compiler listing which supplies the octal code length offsets into segments.
Default is N+.

O+,O- Option processing. This option controls whether the compiler will process further comment options. This option takes effect at the end of the current option-sequence. This option is to help in importing or exporting programs that use or even depend on comment options to control compilation or the meaning of a program. Once set to O-, no further options are honored, including O option.
Default is O+.

P+,P- Allow (+) or inhibit (-) Post-Mortem-Display (PMD), or execution error trace-back, in the event of an error. The PMD is activated by most run-time errors and provides a symbolic display of simple variables and procedure and function calls. Setting P- prior to the first "begin" of a procedure or function will inhibit any display for that procedure or function, and the code file length will be somewhat smaller.
Default is P+.

S+,S- Program segmentation. This option is off (S-) by default, and the entire program is one segment. If this option is on (S+) when the block of a procedure or function is encountered, the code for that block will be a separate code segment, and can share space with other code segments. Nested procedure and functions that are not separate segments become part of the segment of their enclosing block. See section 3.1.3 for a discussion of how to tailor the segmentation of a program.
Default is S-.

T+,T- Compiler-generated run-time tests. If T+, the compiler will generate run-time tests to check the range of array references, pointer references, assignment to subrange and set variables, and case statement jumps. Any value that is found by these run-time tests to exceed its allowed (declared) range will cause a run-time error. If T-, these tests are not added to the program, and any error that might have been caught by the tests will be allowed and can cause undefined results (even corrupt and crash the RTS). The program may run somewhat faster with the tests turned off. It is our experience that the time saved with T- is rarely worth the confusion if an error occurs and is not detected.
Default is T+.

V+,V- Variable reporting within PMD. If V+ (default), PMD will report the (first 8 characters of the) names of simple variables within each procedure or function that was active at the time of the error, along with their values. Setting V- prior to

the last "end" of a block will inhibit display of variables for the block. Setting V- will usually reduce the size of the program code, but will not affect execution speed.
Default is V+.

Wn Workspace. Set workspace to n, where n is in words. No effect in this version (V1-0-F).
Default is all available memory.

X+,X- Extensions. With the default of X-, no Pascal-OS/8 (or other) extensions in a source program will be accepted. Extensions that Pascal-OS/8 recognizes will be flagged with the error number 100 ('Use of extension makes program non-standard'). Other extensions will usually be flagged with other error numbers. This option aids in keeping a program portable to other Pascal implementations. Use X+ to allow use of Pascal-OS/8 extensions.
Default is X-.

3.1.2. List of Extensions to Standard Pascal

Pascal-OS/8 offers some extensions to standard Pascal. To use any extension, the compiler option "X+" must be used. All extensions are subject to change without notice in future versions of Pascal-OS/8. Pascal-OS/8 does not offer the extension known as "conformant arrays" (Level 1 of the ISO standard). Many of our extensions are related to accessing OS/8 files and devices.

File-Related Extensions:

3.1.2.1 ASCII, EightBit

ASCII and EightBit are non-standard pre-defined types much like the standard pre-defined type char which, in Pascal-OS/8, contains the printable ASCII graphic characters, ' ' thru '~'. There are 95 printable characters in ASCII.

Figure 3.1.2.1-1 Char-Like Type Ranges

	Char	ASCII	EightBit
low:	chr (32)	chr (0)	chr (0)
high:	chr (127)	chr (127)	chr (255)
members:	95	128	256

The range of type Char is designed to allow the useful declaration "set of Char". Sets in Pascal-OS/8 can contain a maximum of 96 elements. Types ASCII and EightBit extend upon this range, and provide convenient super-ranges of type character. The declarations "set of ASCII" or "set of EightBit" are not allowed, because the base type has too many members to represent in Pascal-OS/8 sets.

File of char, file of ASCII and file of EightBit are all three-way packed on OS/8 file-structured device.

Text files are special files of char. Characters outside of the range of char are ignored (dropped) when reading from a text file. The readln and writeln procedures are only allowed on text files. Only when reading from a text file, tabs are expanded to the customary OS/8-style tabulation positions.

When reading from a "file of ASCII", nulls are dropped, and the parity bit is masked off. When reading from a "file of EightBit", all possible values in the EightBit range are allowed through.

The following program example shows how to perform useful

Input and Output from non-text files.

Figure 3.1.2.1-2 Non-Text Strategies

```
{$X+} program p(i:'tt:',o:'tt:');
type NonText = file of {Char or ASCII or} EightBit;
var i,o:NonText;
    a:integer; b:real; c:char;

  procedure GetEoln(var f: NonText); {Readln for non-text}
  begin
    while f^ <> chr(13) do get(f); get(f)
  end;

  procedure PutEoln(var f: NonText); {Writeln for non-text}
  begin write(f,chr(13),chr(10)) end;

begin
  reset(i);
  rewrite(o);
  write(o,'enter char:');
  read(i,c); GetEoln(i);
  write(o,' ''',c,''''); PutEoln(o);
  write(o,'enter integer:');
  read(i,a); GetEoln(i);
  write(o,' ',a); PutEoln(o);
  write(o,'enter real:');
  read(i,b); GetEoln(i);
  write(o,' ',b); PutEoln(o)
end.
```

3.1.2.2 Close(f), Close(f,i)

Close is a non-standard pre-defined procedure that can close a file. Files are automatically closed in Pascal-OS/8 when exiting the block in which they are declared. The Close procedure is used in three situations:

1) To make permanent a local Pascal file variable. Local files are those Pascal files that are not named in the program statement, called external files. Only external files are retained upon program completion. With the Close(f) statement, where f is the Pascal variable name of the local file, the local file will be made permanent, with its temporary name from the sequence PASCAM.TM, PASCAN.TM, PASCAO.TM, ..., ZZZZZZ.TM. Note that this form uses only a file parameter.

2) To close an indexed file with a definite size. OS/8 imposes the restriction of only one open-write file on a device at a time. It is possible to create a file of a defined size, without having to write it sequentially and then close it. The following program fragment illustrates this.

```
const
  WorkingSize = 100;
var
  f: file [1..WorkingSize] of integer;
begin
  rewrite(f);
  close(f,WorkingSize);
  reset(f);
  index := { 1 ... WorkingSize };
  get(f,index);  f^ := {IntegerValue}; put(f,index)
end;
```

Note that this form uses two parameters, first an indexed file, second a positive, non-zero integer.

3) To close a file before exiting a block. This can be done, but the only reason I can think of is discussed in 1), above.

3.1.2.3 Get(f,i), Put(f,i)

The standard procedures Get and Put have been extended for indexed files to express the desired file index. This form is only valid when doing Input (get) or Output (put) on a non-standard indexed file. The index must be within the declared range of the indexed file. If extending the file with put, the index must be one greater than the last highest index for the file. Read and write cannot be used with indexed files.

3.1.2.4 GetFN(f,string14)

This non-standard pre-defined procedure can return the file name string of the file parameter, f. The second parameter to GetFN must be a string with length 14. The format returned is "dddd:nnnnnn.xx" where "d" and "x" could be spaces. The following example uses GetFN, and writes out the file name.

```
program p(f,output);
const len=14;
var f:text; string14:packed array[1..len] of char;
begin
 {$X+}
 GetFN(f,string14);
 writeln('''',string14,'''')
end.
```

One can alter the value of the file name string and re-present it as a new file name using the extended form of reset or rewrite. For instance, if string14[1] is a space, then no device has been specified for the file yet, and one could assign characters to set a device. Note that the default device is always 'DSK:'.

3.1.2.5 {Indexed} file [1..n] of {type}

Pascal-OS/8 offers a non-standard indexed file (random-access file) mechanism. This allows non-sequential access to a mass-storage-based data file. This cannot be a text file, but can be a file of any element type. Packing is not supported on indexed files (or non-character sequential files), however, and we have found the practical declaration of an indexed file to be as demonstrated in the following example.

```
 1  program ExampleIndexed;
 2    type
 3      BlockIndex = 0..255;
 4      WordRange = 0..4095;
 5      IndexedRange= 1..4095;
 6      block = array[BlockIndex] of WordRange;
 7    var
 8  {$X+}
 9      f: file [IndexedRange] of block;
10      WordNumber, index: IndexedRange;
11  begin
12    rewrite(f);
13    index := 1;
14    for WordNumber := 0 to 255 do {zero block}
15        f^[ WordNumber ] := 0;
16    put(f,index);       {write out first block}
17  end.
```

The property indexed is syntactically denoted by the subrange notation ("[1..4095]" in the example above). The program SVUTIL, included in the Pascal-OS/8 distribution, uses an indexed file that is declared much like this example, and offers a complete example of a method of using this feature. See also INDEXD.PS in the distribution for a brief example. Indexed files are an ad-hoc design, an experiment, totally non-standard, and are subject to revision in future releases of Pascal-OS/8.

To create an indexed file, use rewrite, and use put(f,i) to write each element of the file, starting with the first element, and adding elements one at a time in sequential order. It is an error if a put is attempted to a new indexed file (rewritten) with an index more than one higher than the prior highest index to the file.

To use an existing file as an indexed file, use reset, and then use get(f,i) to access element i of the file, and use put(f,i) to re-write element i. The next section discusses Length and MaxLength, which are needed for accessing indexed files. An indexed file opened with Reset cannot be extended.

To create a fixed size indexed file in order to reserve disk space, use rewrite(f); close(f,length) where length is the number of file elements desired as the size of the file. Such a file is undefined in all its' elements. See also section 3.1.2.2 Close(F,I), above.

3.1.2.6 Length(f), MaxLength(f)

The non-standard pre-defined function Length(f) returns the current length of file f, as a positive integer number of elements. Length is not implemented for text files.

The non-standard pre-defined function MaxLength(f) returns the maximum length of file f as a positive integer number of elements. MaxLength is not implemented for text files.

3.1.2.7 Program Heading Specified-Default File

The program heading files can have an extended syntax which specifies a default file name pattern for the file. This looks like:

```
{$X+}  program p( f :'device:file.extension')
```

Each program heading file can have its own specified-default. When the program is executed, the file parameters given on the command statement are substituted in an order dependant fashion for the program heading files. See section 3.2.1.2 for a full discussion of this extension and how it fits into the general file binding scheme.

3.1.2.8 Reset(f,string); Rewrite(f,string);

Either reset or rewrite may have a non-standard second parameter which is a string literal or a string variable specifying the OS/8 file name to be associated with the Pascal file variable 'f' before the reset (lookup) or rewrite (enter) is performed. Spaces are ignored in the literal or string. The literal can be up to 148 characters long, while a string variable passed to Reset or Rewrite can be up to 4095 characters long. This is another opportunity to alter the default binding of the file. See section 3.2.1.4 for a discussion.

3.1.2.9 ValidFN(string)

The non-standard pre-defined function ValidFN takes a file-name-string literal or variable as its parameter, and returns the Boolean value true if the string parameter meets the syntax rules for a file name, otherwise ValidFN returns the Boolean value false if the string parameter does not meet the syntax rules for a file name.
Spaces are ignored in the literal or string. The literal can be up to 148 characters long, while a string variable passed to ValidFN can be up to 4095 characters long.
ValidFN does not check to see if the device or file exists and is accessible (this is done upon Reset or Rewrite).

Other Extensions:

3.1.2.10 Date(string10)

Date is a non-standard pre-defined procedure which returns the current OS/8 date in its string parameter. The string must be defined as:

```
var
  string10: packed array[1..10] of char;

begin
  {$X+}  Date(string10);
  writeln(string10);
  ...
```

The string returned is in the I.S.O date format:

```
'yyyy-mm-dd'
```

or is ten blanks if the OS/8 date is missing or invalid.

3.1.2.11 DevGet(i), DevPut(i,i), DevReady(i)

These non-standard pre-declared procedure and functions are part of a device-access experiment. See section 3.2.4 for a description.

3.1.2.12 Execute(string)

Execute is a non-standard pre-declared procedure which takes a string argument and executes it either under the control of OS/8 CCL or of the Pascal-OS/8 run-time system. A call to Execute never returns.

If the string starts with a dot ('.'), the dot is removed and the string is used as an argument to CCL, and the run-time system chains to CCL to execute the string as a CCL command. This string cannot be longer than 36 characters long. Example:

```
{$X+}   Execute('.SUBMIT JOB')
```

If the string starts with the run-time system command prefix character ('%'), the '%' is removed, and the string is used as an argument to the run-time system, which processes the string as a new command statement. This string can be 200 characters long. For example:

```
{$X+}   Execute('%SYS:PMD,DSK:USER/R+')
```

The characters in either execute call may be upper or lower case, and are all converted to upper case characters.

All files open will be closed when Execute is called.

3.1.2.13 Halt, Halt(string)

The non-standard pre-declared procedure Halt causes immediate program termination with a PMD (Post-Mortem Display), if PMD was enabled in the source of the program. Halt accepts a string parameter, which is printed at the time of the halt execution as a message. For example:

```
{$X+}   halt('you went too far this time, bud!')
```

3.1.2.14 Kludge n

Kludge is a machine-access experimental feature, used in place of a procedure block, as in:

```
procedure x; kludge n;
```

where n is a non-zero integer. See section 3.2.5 for a description.

3.1.2.15 Mark(p), Release(p)

Mark and Release are non-standard pre-declared procedures that serve the needs of the compiler. Both take a pointer parameter. Mark(p) initializes p to the current location of the stack. Later Release(p) releases the stack back to the point marked by p. These procedures should be used only with a thorough understanding of the standard procedures new and dispose, and the stack mechanism.

3.1.2.16 OCT

The write procedure has been extended to write integer values in octal (base 8) with the following syntax:

```
write(f, v: fw oct);
```

where f is a text file, v is an integer expression, fw is the field width for the write, and the 'oct' directive tells the compiler to produce a write-octal instead of write-decimal. The field width can be at most 8 digits, preceded if necessary by spaces. Leading zeroes are printed.

3.1.2.17 Ord(pointer-type)

The standard procedure Ord has been extended to allow arguments of pointer types. Ord(pointer-type) converts a pointer to an integer, normally for diagnostic writes.

3.1.2.18 Otherwise

The case statement has been extended by an otherwise clause which may specify one or more statements to be executed if the selector expression value does not match any of the

explicit case labels. Otherwise is a reserved word.

Example:

```
type days = (sun, mon, tue, wed, thu, fri, sat);
var  today: days;
  .
  .
  .
case today of
  mon: shop;
  tue: clean;
  wed: practice;
  otherwise
    play; garden; visit; etc
end
```

In this example, if today has one of the values mon, tue, or wed, the corresponding statement (shop, clean, or practice) will be executed. For all other values of today, the statement sequence (play; garden; visit; etc) will be executed. If the otherwise clause had not been used and the value of today was not mon, tue, or wed, the program would have been terminated with a fatal error.

Since the otherwise statement can be empty, the following is valid:

```
case today of
  mon: shop;
  tue: clean;
  wed: practice;
  otherwise
end
```

This is semantically equivalent to:

```
begin
  if today = mon then shop
  else
    if today = tue then clean
    else
      if today = wed then practice
end
```

3.1.2.19 Pwr0f10(r)

Pwr0f10 is a non-standard pre-declared function that takes an integer number as its argument, and returns the corresponding power of ten as its result.

3.1.2.20 'B', 'C'

Integer constants may be modified by the 'B' suffix, the 'C' suffix, or both. The 'B' suffix denotes a base-8 number,

rather than the default base-10 number. The 'C' suffix denotes a character constant. The suffixes can be upper or lower case.

```
Eight =          10b;
ControlC =        3BC;
ControlY =       32bc;
ControlZ =       26c;
```

3.1.2.21 '_' in Identifiers

The underline character is allowed as a non-standard character in identifiers. An underline must not be the first character in the identifier. Each underline is considered significant, just as letters and digits are. Note that only the first eight characters are considered significant, however.

3.1.2.22 {$compiler-options }

The compiler-options provision does not extend the language, but does control the compiler in important ways, useful to users of Pascal-OS/8 who hope to tailor programs to the OS/8 environment. See section 3.1.1.

3.1.3. Segmentation of Large Programs

If the error message:

"Ran out of memory during program or segment load"

occurs while running your program, this section may be for you. If a program is too large to fit in the available memory, it is possible to direct the compiler to break the program code up into segments, which do not need to be in memory if they are not executing. Then the program can execute in less memory, or more memory is available for data space, or both.

The program is still compiled in one whole piece. The only purpose of segmentation is to fit programs with a lot of code, or programs that need more memory for data (stack and heap) into the restricted memory of a PDP8. There is no scheme for separate compilation or linking of external routines.

There are three strategies of segmentation:

1) No segmentation (all procedures and functions in one segment, the default mode).

2) Structured segmentation.

3) Global segmentation (each procedure and function is a separate segment).

3.1.3.1. No Segmentation

This is the default mode for segmentation, and corresponds to the default setting of the compiler "S" option, "S-". The entire program is one code segment, and there must be enough memory to accommodate the program, plus enough memory for the data space (stack and heap). Small programs normally need not be segmented, and medium programs need not be segmented if the machine they are to run on have enough memory. The three advantages of this mode are simplicity, the code file is a minimum size, and there are no delays for loading code during execution.

3.1.3.2. Global Segmentation

Try this segmentation mode first before designing a structured segmentation layout. It is easy to try, and will tell you if the program in question will execute in the amount of memory desired. If the compiler option S+ is set at the beginning of a program, and left at this setting for the whole compilation, the program is said to be globally segmented: each procedure and function will become a segment, and the amount of memory needed for the program code is the amount of memory needed by the largest procedure or function. So, very simply, large programs can often be allowed to run. The main disadvantages are (1) the code file may explode in size (space is wasted because each procedure's code, even short ones, takes up at least one disk block), and (2) the program may "thrash" as segments

contend for residency. If the program must be segmented, and either of these two problems is unacceptable, try structured segmentation.

3.1.3.3. Structured Segmentation

Try global segmentation first before designing a structured segmentation layout. If no segmentation and global segmentation are insufficient to meet the needs of a program, you can structure the segmentation to tailor the program to a target amount of memory. This mode takes the most thought and work. The advantages of pursuing this mode are achieving a better trade-off between required memory size, program data needs, program performance, and code file size. The Pascal-OS/8 compiler is carefully segmented in order to execute in a minimum of 24k. Without segmenting, the compiler would need over 32k of memory. In fact, segmenting was implemented primarily in order to be able to support the compiler on 32k machines.

3.1.3.4. Segmentation Essentials

A segment is a piece of code containing one or more procedures and functions (the main program is functionally identical to a procedure). Each procedure or function that is a separate segment contains within it all nested procedures and functions that are not separate segments.

A segment is read into memory all at once before any part of it can execute. A segment is limited in length to 512 disk blocks long, which is very large, and any practical program would have been broken down into multiple segments before hitting this limit. A 32k 12-bit machine, less 8k for the run-time system and interpreter, can host at most a 95 disk block long segment, also very large, leaving less than 250 words for data! (A disk block is 256 (400 base 8) words long.)

The length of a segment is rounded up to the nearest disk block boundary, to support reading the segment into memory independently of other segments. So a very short program, or a very small segment, which may only be a few code words long, "wastes" the remainder of the disk block of space. This is why a globally segmented program may "explode" in code file size: each procedure or function, when a separate segment, takes at least one disk block of storage.

When first starting a program, or when the program calls (or returns to) a procedure or function not in the currently executing code segment, the segment that does contain the procedure or function must be loaded (read in) if it is not already in memory. If there is not enough room to load the desired segment, then all currently resident segments will be dropped from memory (forgotten). If there is still not enough room to load the desired segment, the error "Ran out of memory during program or segment load" will be reported. If a globally segmented program persists in getting this error, and you are up to your available memory limit, then you must consider algorithm changes to accommodate the program on your machine. This will happen sometimes; this implementation does have its limits. There is an implementation of the Adventure game in Pascal, that we could not run on our 32k machine with global segmentation (and T-,P-). Such is life.

3.1.3.5 Design of Structured Segmentation

The major constraint is the amount of memory available for code and data. The minor constraints are segment loading activity and program code file size (on disk). The layout of many programs for optimal segmentation is often the same as good structured program layout. Procedures and functions should be nested within their calling Pascal block. (See a good Pascal text book for an explanation of the idea of a Pascal block.) Related procedures and functions should be grouped together in the source, nested according to usage.

So, first, group related procedures and functions in the same segment, so they can call each other without segment load activity whenever possible. Next, make all segments small enough to fit in memory, by splitting off nested large procedures and functions as separate segments. Usually the code file size will be the least constraint, and will be larger than a non-segmented program, but much smaller than a fully globally segmented program. The design process may involve iterations of shuffling the procedures and functions, setting the "S" option, compiling, and examining the results with the Pascode utility (described below).

The setting of the "S" option at the time the Pascal block of a procedure or function is encountered (that is, right after the procedure or function heading) will determine whether the Pascal block is a separate segment: "S-" to continue the current segment, "S+" to starting a new segment. Nested procedures and functions that are not separate segments become part of the segment of their enclosing Pascal block. Section 3.1.1 discusses in full how to use compiler options.

Section 2.2.3 gives an extensive example of a compilation listing, with descriptions of how the code lengths appear. In the compiler listing, the second column of numbers (reading from the left) gives the stack allocation space, if in a declaration part, or the code length, if in executable statements. We are interested in the code length report. This column is expressed in octal, and corresponds to the number of 12-bit words needed to contain the code. In an unsegmented program, the code length will be cumulative from one procedure to the next. In a globally segmented program, the code length will start over at zero for each procedure and function. In a structured segmented program, the code length will accumulate while within a segment, and start over at zero for each new segment. The program Pascode can report the segment lengths (in disk blocks), using the compiled .PB file (see below).

3.1.3.6 Restriction

Consider the following program example:

```
program P;

   procedure A;
   begin end;

   procedure B;
   begin end;
```

```
        begin end.
```

The Pascal block of program P contains both procedures A and B. It is not possible to group procedures A and B into one segment that is distinct from the program segment (the enclosing Pascal block), because A and B are at the same level, and share a common enclosing Pascal block. If there were a large number of procedures or functions at the same level, one might wish to make groups of them into segments, but this is not possible. Segment grouping must follow the nesting structure of Pascal.

3.1.3.7 Segmentation Example

The following is an example of a structured segmentation design, and the use of the "S" compiler option to achieve the segmentation desired.

Figure 3.1.2-1 Segmentation Example

```
P8COM V1-0-F F

    1 000002 Program SegmentationExample;
    2 000004
    3 000004   procedure A;
    4 000002   begin end;
    5 000015
    6 000015   procedure B;            {$S+}
    7 000015
    8 000015     procedure B1;         {$S-}
    9 000002     begin end;
   10 000015
   11 000015     procedure B2;         {$S+}
   12 000015     begin end;
   13 000015
   14 000015     procedure B3;         {$S-}
   15 000015     begin end;
   16 000027
   17 000027   begin {B} end;
   18 000015
   19 000015   procedure C;            {$S+}
   20 000015
   21 000015     procedure C1;
   22 000002     begin end;
   23 000000
   24 000000     procedure C2;
   25 000000     begin end;
   26 000000
   27 000000   begin {C} end;
   28 000015
   29 000015   procedure D;            {$S-}
   30 000015   begin end;
   31 000027
   32 000027 begin {main program}
   33 000042 end   {main program}.
NO ERRORS DETECTED.
```

Notes:

Line 1: The default setting of "S-" is in effect. Segment one always contains the main program.

Line 3,4: proc A is S- and becomes part of the segment containing the main program, segment one.

Line 6: proc B is S+, becomes segment two.

Line 8: proc B1, nested within B, is S-, and becomes part of segment two with B.

Line 11: proc B2, with S+, becomes a new segment, segment three.

Line 14: proc B3, with S-, is not a new segment; it reverts to be part of the segment of its containing Pascal block, that is, part of segment two with proc B and B1.

Line 17: this is the statement part of procedure B, segment two. The Pascal block of B is lines 7 ... 17.

Line 19: proc C is S+, therefore a new segment, segment four.

Line 21: the S option is not set explicitly here, so the last setting given in the source, S+ on line 19, applies to the Pascal block of proc C1. C1 becomes a new segment, segment five. We have found it helpful to state the S option explicitly after each procedure and function heading in order to avoid accidental settings such as this might be.

Line 24: proc C2 becomes a new segment, segment six.

Line 27: this is the statement part of procedure C, segment four.

Line 29: S- is set for the Pascal block of proc D, and it becomes a part of the Pascal block containing it, which is the main program, segment one.

3.1.3.8 Segmentation Utility: PasCode

The program PasCode is included in the Pascal-OS/8 distribution, and can aid in determining the effect of the use of the "S" compiler option in structuring segments. PasCode, by default, will decode the P-code of a program, unless the "/S" option switch is given to PasCode on the command statement, as in:

```
.R P,PASCOD,SegmentationExample/S
```

PasCode is described more fully in appendix T.4.

3.2 Run-Time System

The Pascal-OS/8 Run-Time System supports various ways to bind Pascal files to OS/8 devices and files. The following sections discuss how to do it. Good luck!

3.2.1. File Binding

File binding in Pascal-OS/8 can be separated into four levels: (1) compiler-defaulted; (2) specified default; (3) control statement; and (4) reset/rewrite binding.

The default binding pattern, command statement processor, GetFN extension, and Reset/Rewrite string parameter extension provide a very flexible file binding mechanism.

3.2.1.1. Compiler-Defaulted Binding

When a Pascal-OS/8 program specifies file variables in the program heading, the compiler constructs the default name binding for that file variable from the file variable name itself by truncating the name to six characters and setting the result into an internal table (in the code file). For example, in the program heading:

```
Program P (Data, Results);
```

the default binding for file Data is 'Data.', and the default binding for file Results is 'Result.'.

The files Input and Output present an exception in that their device is set it 'IO:', for interactive I/O and BATCH support. (See section 5.3.5.)

3.2.1.2. Specified-Default Binding

The default binding pattern may be specified by the user as a language extension which specifies a string as the default binding pattern for a program header file. This looks like:

```
{$X+ accept Pascal-OS/8 extensions }
Program P ( F :  'device:file.extension' );
```

If this extension is used, the compiler defaulted binding pattern for the file affected is completely superseded. Any logical default binding pattern may be provided by the user.

```
F : ' '                 {empty, one or more blanks}
F : 'D:'                {device name only}
F : 'X'                 {file name only}
F : '.EX'               {extension only}
F : 'D:F'               {device and filename}
F : 'F.EX'              {filename and extension}
F : 'D:.EX'             {device and extension}
F : 'D:F.EX'            {device, filename and extension}
```

This specified-default gets stored in the program executable file (.PB) header, and is further processed by the run-time system in combination with the command statement given at the time the program is started.

3.2.1.3. Control Statement Binding

When the command statement (C.S.) is processed, the file names given by the user are integrated into an internal table with some additional information.

The actual binding is done in the MPP (Match Program Parameters) routine:

-- if a C.S. device is non-blank, it is used in lieu of any program device default.

-- if a C.S. filename is non-blank, it is used in lieu of any program filename default. Also, a C.S. specified filename will clear a program device default.

-- if a C.S. '.' was seen, the C.S. extension is used ('.' alone clears extension, like OS/8 CCL) in lieu of any program extension default.

3.2.1.4. Reset/Rewrite Binding

Except for the files Input and Output, which are automatically reset and rewritten, respectively, if they exist in the program header, all files must be reset or rewritten explicitly by the program. Herein lies one more opportunity for modifying the file binding using GetFN and reset/rewrite. GetFN can be used (see section 3.1.2.4) to obtain the file name string pattern as developed so far (after command

statement processing), the program can modify the string, and the (modified) string is re-presented to the RTS binding mechanism via the second parameter (string) to reset or rewrite.

```
var
  f : file of {anything};
  fn: packed array [1..14] of char;
...
  {$X+ accept Pascal-OS/8 extensions}
  GetFN(f,fn);
  if (fn[13] = ' ') and (fn[14] = ' ') then
    begin (* convert blank extension to 'ex' *)
      fn[13] := 'e'; fn[14] := 'x'
    end;
  reset(f,fn); { bind file and start reading }
...
```

Note that use of a string parameter to reset or rewrite completely replaces any prior file name pattern associated with the file.

When reset or rewrite is called, if the device name for the file is empty, device 'dsk:' is substituted "at the last moment". Therefore the device name is never blank when the RTS does an OS/8 USR Inquire for the device.

The filename part of the full file specification may be empty (blank) at the time of the OS/8 USR Lookup or Enter, and, if the device being accessed for the file is file-structured, the missing file name will result in a run-time error message: 'Invalid filename'.

3.2.2 Internal Device Handlers.

The Pascal-OS/8 run-time system implements a small number of additional character-oriented device handlers for use by the Pascal program. These handlers exist in addition to the normal OS/8 device handlers, and in fact "hide" the names of OS/8 device handlers. All of these handlers are available for any character type file (text, file of char, file of ASCII, or file of EightBit).

The handlers fall into three classes: (1) line-buffered terminal handlers; (2) non-buffered terminal handlers; and (3) miscellaneous handlers.

The following control-characters have effect on any terminal handler (buffered or immediate) that is assigned to a file and in use:

Name	Display	Effect
Control-C	"^C"	abort program, stop BATCH, return to OS/8.
Control-S		X-OFF, hold output until X-ON. (DC3)
Control-Q		X-ON, resume output after X-OFF. (DC1)
Control-O		slough off (mute) output, any other key will resume.

The following control-characters have effect only from the TT: (or TTO:) handler:

Name	Display	Effect
Control-D	"^D"	abort program, close files, do PMD, abort BATCH.
Control-A	"^A"	abort program, close files, abort BATCH.
Control-B		debug enable: interpreter state frames.
Control-E		debug disable: interp. and segment state frames.
Control-F		debug enable: segment loader state frames.

(1) Line-Buffered Terminal Handlers:

These handlers all share the same characteristics; all are line-buffered with editing the same as the OS/8 KeyBoard Monitor during input; all are immediate, with no buffering, on output.

TT: console terminal (keyboard and display)
LP: line printer terminal (keyboard and display)
RIO: remote Input/Output (modem or terminal)
TEK: auxiliary terminal (keyboard and display)
GIN: same as TEK: but no echo at all
IO: Input/Output stream terminal (keyboard and display)

The TT: handler is the default handler for interacting with a person using the files Input and Output. Any other handler may be substituted by name to achieve interaction on another terminal. For example, the program:

```
program example(input,output);
...
```

can be used on the auxiliary terminal using the command statement:

```
.R P,EXAMPLE,TEK:,TEK:
```

The following are the effective editing characters during input from the line-buffered terminal handlers.

Name	Display	Effect
BackSpace		delete one character, erase from screen (control-H).
Rubout or Delete		delete one character; if OS/8 scope bit not set, echoes as "\x\", the TTY convention; if OS/8 scope bit is set, same as BackSpace: erases from screen.
Control-U	"^U"	delete current input line.
Line Feed		re-print current input line.
Tab		perform OS/8 tabulation; always expanded to blanks.
Carriage Return		on command statement: process command and return to the run-time system if in job step mode with the '%' prompt else return to OS/8; as program input: terminate current line and pass it to the program.
Escape	"^["	on command statement: process command and return to OS/8; as program input: no special meaning; screened from text files.
Control-Z	"^Z"	on the command statement: no special meaning; as program input: cause eof(f) to become true (after the end of the current input line, if any).

(2) Non-Buffered Terminal Handlers:

These handlers share characteristics: they are immediate upon read and write: there is no buffering or editing. They do not echo characters on input.

```
TTO:  console terminal (keyboard and display)
LPO:  line printer terminal (keyboard and display)
RIOO: remote Input/Output (modem or terminal)
TEKO: auxiliary terminal (keyboard and display)
```

(3) Miscellaneous Handlers:

These handlers are not interactive terminals, but have distinctive purposes. There are no active control-characters for these handlers.

CS: The CS: handler accesses a copy of the command statement that initiated execution of the current program. It is a read-only character handler. Following is an example of CS: usage.

```
procedure EchoCS;
  var  temp: text;   ch: char;
begin
  {$X+}
  reset(temp, 'CS:');
  while not eoln(temp) do
    begin
```

```
            read(temp,ch);
            write(output,ch)
          end;
        writeln(output)
      end;
```

Normally, the control-statement image available through CS: would be analyzed for control information.

PT: PT: is a high-speed paper tape reader and punch handler. Reset(f,'PT:') will bind the character file f to the paper tape reader; rewrite(f,'PT:' will bind character file f to the paper tape punch.

NULL: The null handler acts as an immediate eof if read from, or as an infinite data sink if written to. Null: is not available for non-character type files.

3.2.3. Creation of SV files from PB files

It is possible to make Pascal-OS/8 programs more directly executable by converting them to a Save File which, when run, will chain to the run-time system.

Three important restrictions apply:

1. The Pascal-OS/8 run-time-system must exist on SYS: with a stable name (internal to P8HEAD.SV). See Below.

2. The user program converted to a .SV file must also reside on SYS:.

3. Your system must be OS/8 V3D or later (support of SOFSET is mandatory, PIP V3D is mandatory).

A special header file, P8HEAD.SV, is provided with the distribution. The operation requires OS/8 PIP as distributed with OS/8 V3D: PIP must support the /I image concatenation option as used for making MACREL/LINK .RB libraries. To convert a .PB file to a .SV file, simply do:

```
.R PIP
*SYS:program.sv < dev1:P8HEAD.SV,dev2:program.PB/I$
```

Now the command statement:

```
.R program,files
```

will start your program executing and initiate binding of files to your program. In this mode of Pascal-OS/8 usage, access to the run-time-system is fully automatic, as long as the restrictions stated

above are observed.

No method is provided to reverse the PIP concatenation, so you are advised to keep a copy of the .PB file, especially if the .PB file cannot be re-compiled from a Pascal source file.

P8HEAD chains to P8RTS and must know its name. If the name of P8RTS is changed from 'P.SV', P8HEAD.SV must be patched. See section 5.4.2.

Note: if the program.PB file (code file) is longer than 255 blocks, PIP will fail! If your program is actually that long, you will have the challenge of creating your own concatenation routine.

3.2.4 Device Access Experiment

The three non-standard predeclared Pascal-OS/8 routines:

```
function DevReady(unit:0..maxdev) : Boolean;

function DevGet(unit:0..maxdev) : integer;

procedure DevPut(unit:0..maxdev; value:integer);
```

comprise the device-access experiment. They all take as their first parameter a unit number from Table 3.2.4-1, below, specifying which device is to be accessed.

DevReady returns false if the unit is not ready (no key struck on a keyboard, or display not done with last character).

DevGet returns an integer value from the device specified in the unit parameter.

DevPut displays the value given as the second parameter upon the device specified in the unit parameter. Excess high-order bits are ignored by the device (character devices ignore the upper 16 bits).

The following program shows how to make simple use of the DevGet and DevPut procedures.

```
program dv(input,output);
{$X+ non-standard programming}
  var InUnit,OutUnit:integer;
begin
  write(output,'enter InUnit OutUnit: ');
  read(input,InUnit,OutUnit);
  while true do DevPut(OutUnit,DevGet(InUnit))
end.
```

The two capabilities that have proved useful to us have been the ability to check a terminal keyboard for key struck without having to commit a read (using DevReady(unit)) and the XY8E plotter access (DevPut(5,function)). The conventions for sending data to the plotter are described below.

For instance, to see if the console keyboard has been struck, do:

```
if DevReady(3) then {read console}
```

The following table gives the unit-number -- device correspondences.

Table 3.2.4-1 Device Unit Numbers

unit#	DevGet	DevPut	DevReady
0	SW front panel	MQ register	always true
1	TEK: kbd	error	true if TEK: key struck
2	error	TEK: display	true if TEK: display ready
3	TT: kbd	error	true if TT: key struck
4	error	TT: display	true if TT: display ready
5	error	XY8E plotter	always true

An out-of-range unit number will cause a run-time error. A call to DevGet with a unit number of 2, 4 or 5, or a call to DevPut with a unit number of 1 or 3 is considered out-of-range, as is negative or too large.

Plotter "language".

The XY8E plotter unit (5) implements the following conventions, which are very close to the hardware conventions. Each call can move only one step in the North or South direction, and/or one step in the East or West direction. (N-E, N-W, S-E, S-W are allowable combinations, but N-S, E-W might damage the plotter!). Penup and pendown also must be separate calls to DevPut(5, nnnn).

plotter unit number:

```
XY8E= 5
```

pen control functions:

```
penup=      2000b       (gets translated to PLPU iot)
pendown=    4000b       (gets translated to PLPD iot)
```

single step functions:

```
stepnorth=  0020b
stepsouth=  0040b
stepeast=   0010b
stepwest=   0004b
```

function call:

```
DevPut(XY8E, function);
```

The XY8E direction register layout corresponds to the step functions given above.

```
    ac0  ac1  ac2  ac3  ac4  ac5  ac6  ac7  ac8  ac9 ac10 ac11

                                   r    l    d    u   (600-700)

west=   0004     (u) = drum up   = -x
east=   0010     (d) = drum down = +x
north=  0020     (l) = pen left  = +y
south=  0040     (r) = pen right = -y
600-700 = fancy plotters. the bits to utilize these plotters
          are passed right though.
```

Notes:
-- drum up corresponds to top of drum moving away (viewed from front of plotter).
-- ac10 causes pen down on calcomp 563 plotter.
-- ac11 causes pen up on calcomp 563 plotter.

3.2.5 KLUDGE Procedure Experiment

KLUDGE procedures are non-standard quasi-predeclared procedures implemented in the run-time-system. The following procedures are available:

```
procedure DoIOT(IOT, ACbefore: integer; var ACafter: integer;
                                        var Skip: Boolean);
                        KLUDGE 1;

procedure Abort; KLUDGE 2;

function BatchRunning : Boolean; KLUDGE 3;
```

DoIOT can perform an IOT (or OPR but don't try any memory reference) instruction to access any device on the bus. To use it, be very careful about the IOT value and be willing to take risks -- this can be a very dangerous KLUDGE procedure. The ACbefore and ACafter include the LINK value.

Abort simply aborts the currently running Pascal-OS/8 program, and OS/8 BATCH, if running, without producing a PMD report.

BatchRunning simply tells if OS/8 BATCH is running or not.

These three KLUDGE procedures, and two test procedures (KLUDGE 0 and KLUDGE 4) are used in the first example program, KludgeTest.

```
program KludgeTest(output);  {$X+ very non-standard}

const  MQL = 7421b;

var ACafter: integer; skip: Boolean;
```

```
    procedure TestZero; KLUDGE 0; {null test}

    procedure DoIOT(IOT, ACbefore: integer; var ACafter: integer;
                                            var Skip: Boolean);
                        KLUDGE 1;

    procedure Abort; KLUDGE 2;

    function BatchRunning : Boolean; KLUDGE 3;

    procedure TestFour; KLUDGE 4; {message printed}

  begin
    TestZero;
    TestFour;
    write(output,'BATCH is ');
    if not BatchRunning then write(output,'not ');
    writeln(output,'running.');
    DoIOT( MQL, 2525b, ACafter, skip);
    write('ACafter=', ACafter:5 oct);
    if skip then write(output,' skip')
            else write(output,' no skip');
    Abort
  end.
```

The next program shows DoIOT by itself in a little more specific test.

```
  program KludgeOneTest(input,output);  {$X+ very non-standard}

  var IOT, AC, after: integer; skip: Boolean;

    procedure DoIOT(IOT, ACbefore: integer; var ACafter: integer;
                                            var Skip: Boolean);
                        KLUDGE 1;

  begin
    repeat
      write(output,'Enter IOT, AC: ');
      read(input, IOT, AC);
      write(output,'IOT=',IOT:8 oct,' AC=',AC: 8 oct,'   ');
      DoIOT( IOT, AC, after, skip);
      write(output,'ACafter=', after:5 oct);
      if skip then write(output,' skip')
              else write(output,' no skip');
      writeln(output)
    until false
  end.
```

You will be careful, won't you?

3.4 Pascal-OS/8 Requirements (Hardware, etc.)

Software Requirements

- OS/8 Version 3 or later.

- Your choice of any text editor (EDIT, TECO, SCROLL, VISTA, etc.) for preparing program text files and data text files. In text files compatible with Pascal-OS/8, tabs are expanded according to OS/8 conventions by the run-time system, carriage-return signals end-of-line (EOLN) and control-Z signals end-of-file (EOF); all other control-characters are ignored. See section 3.1.2.1 for ASCII file treatment.

Hardware Requirements

- Digital Equipment Corporation PDP8 or PDP12 or equivalent computer. (PDP8e, PDP12 tested)

- Memory (Adaptation to amount of extant memory is automatic; see section 5.3.1):

  a. 24k minimum memory for compiling Pascal programs; performance is better with more memory.

  b. 12k minimum memory for executing small programs.

  c. up to 32k memory is currently supported.

- A TTY-like or more advanced console device (LA30, LA120, VT52, VT100, etc.).

Optional Hardware

An FPP8a or FPP-12 will be used if present at the time the program is executed, and if any real numbers are used in the program. See section 5.3.2.

The internal device handlers allow for a line-printer terminal, and two additional terminal-like devices. See sections 3.2.2 and 5.4.2.

KT8A Note:

The internal design of Pascal-OS/8 allows for 17-bit addresses, which could be supported by the KT8A memory controller, but, because we did not have a KT8A and compatible memory, KT8A support is not implemented. Memory support is limited to 15-bit addresses. If you are interested in supporting development of KT8A support, please contact the Pascal Group at the address on the cover of this documentation.

3.5 Performance

The compiler processes about 90 to 100 lines per minute.

The execution performance of Pascal-OS/8 is compared here to OS/8 FORTRAN IV, OS/8 BASIC, and the Pascal-S implementation for the PDP8. All benchmark programs are listed, and also serve as some comparative programming examples. All tests are on a PDP8/e with 32k memory, running under OS/8 Version 3D. All FPP tests are using a FPP8a on the same PDP8/e. Pascal programs were compiled with default range tests on and with line-numbers for error traceback (compiler options T+,N+). Pascal-OS/8 programs generally run faster with the tests off (T-) and/or with the line numbers eliminated (N- or P-). Timing tables follow the listing of the bench mark programs.

BenchMark 1: Matrix Multiply

Pascal-OS/8 does not have fast array referencing and the timings for this algorithm show this weakness. See Table 3.5-1.

---------- BenchMark 1 in Pascal: Matrix Multiply

```
program MatrixMultiply;
(* adapted from Wirth, 1971 *)
  const  n = 50;
  var    i, j, k: 1..n;
         a, b: array[1..n, 1..n] of real;
         s: real;
begin
  for i := 1 to n do
    for j := 1 to n do a[i,j] := 1.0;
  for i := 1 to n do
    for j := 1 to n do
      begin  s := 0.0;
        for k := 1 to n do s := s + a[i,k] * a[k,j];
        b[i,j] := s
      end
end.
```

---------- BenchMark 1 in FORTRAN: Matrix Multiply

```
        REAL A(45,45), B(45,45)
        DO 100 I = 1, 45
        DO 100 J = 1, 45
100     A(I,J) = 1.0
        DO 300 I = 1, 45
        DO 300 J = 1, 45
        S = 0.0
        DO 200 K = 1, 45
200     S = S + A(I,K) * A(K,J)
300     B(I,J) = S
        STOP
        END
```

---------- BenchMark 1 in BASIC: Matrix Multiply

```
100 REM BM1.BA
110 DIM A(50,50),B(50,50)
120 FOR I = 1 TO 50
130 FOR J = 1 TO 50
140 A(I,J) = 1.0
150 NEXT J
160 NEXT I
170 FOR I = 1 TO 50
180 FOR J = 1 TO 50
190 S = 0.0
200 FOR K = 1 TO 50
210 S = S + A(I,K) * A(K,J)
220 B(I,J) = S
230 NEXT K
240 NEXT J
250 NEXT I
260 STOP
270 END
```

BenchMark 2: Warshall's Algorithm

Warshall's algorithm provides a fast way of calculating the Boolean sum of the Boolean matrices A**i (summing over i = 1, 2, ..., N), where A is an n by n Boolean matrix. (Warshall, 1962)

Pascal's set data structure provides a natural and efficient way of dealing with such structures. FORTRAN is at a significant disadvantage in this test, as it must do each Boolean operation individually (using a 3-word representation for each array element). If the matrix size exceeded Pascal-OS/8's set size limit, 96 elements, then Pascal-OS/8 would have to be re-programmed, no doubt using a slower technique.

---------- BenchMark 2 in Pascal: Warshall's Algorithm

```
program Warshall;
  const n = 50;
  var   a, b: array[1..n] of packed set of 1..n;
        i, j: 1..n;
begin
  a[1] := [1..n];
  for j := 2 to n do a[j] := [1,j];
  for i := 1 to n do
    for j := 1 to n do
      if i in a[j] then a[j] := a[j] + a[i]
end.
```

---------- BenchMark 2 in FORTRAN: Warshall's Algorithm

```
      LOGICAL A(50,50), B(50,50)
      DO 200 J = 1, 50
      DO 100 I = 2, 50
100   A(I,J) = .FALSE.
      A(J,1) = .TRUE.
      A(1,J) = .TRUE.
200   A(J,J) = .TRUE.
```

```
        DO 400 I = 1, 50
        DO 400 J = 1, 50
        IF (.NOT. A(J,I)) GOTO 400
        DO 300 K = 1, 50
300     A(J,K) = A(J,K) .OR. A(I,K)
400     CONTINUE
        STOP
        END
```

---------- BenchMark 2 in BASIC: Warshall's Algorithm

```
100 REM BM2.BA
110 DIM A(50,50)
120 FOR J = 1 TO 50
130 FOR I = 2 TO 50
140 A(I,J) = 0
150 NEXT I
160 A(J,1) = 1
170 A(1,J) = 1
180 A(J,J) = 1
190 NEXT J
200 FOR I = 1 TO 50
210 FOR J = 1 TO 50
220 IF A(J,I) = 0 GOTO 290
230 FOR K = 1 TO 50
240 IF A(J,K) = 1 GOTO 270
250 IF A(I,K) = 1 GOTO 270
260 GOTO 280
270 A(J,K) = 1
280 NEXT K
290 NEXT J
300 NEXT I
310 STOP
320 END
```

BenchMark 3: Sorting an array of integers.

This is adapted from Wirth, 1971.

---------- BenchMark 3 in Pascal: Sort, N = 500

```
program sort;
  const n = 500;
  var   i, j, k: 1..n;  m: integer;
        a: array [1..n] of integer;
begin
  for i := 1 to n do a[i] := i;
  for i := 1 to n-1 do
    begin  k := i;  m := a[i];
      for j := i+1 to n do
        if a[j] > m then begin k := j; m := a[j] end;
      a[k] := a[i];  a[i] := m
    end
end.
```

---------- BenchMark 3 in FORTRAN: Sort, N = 500

```
      INTEGER A(500)
      N = 500
      DO 10 I = 1, N
10    A(I) = I
      N1 = N - 1
      DO 30 I = 1, N1
      K = I
      M = A(I)
      I1 = I + 1
      DO 20 J = I1, N
      IF( A(J) .LE. M) GOTO 20
      K = J
      M = A(J)
20    CONTINUE
      A(K) = A(I)
30    A(I) = M
      STOP
      END
```

---------- BenchMark 3 in BASIC: Sort, N = 500

```
100 REM BM3.BA
110 DIM A(500)
120 N = 500
130 FOR I = 1 TO N
140 A(I) = I
150 NEXT I
160 N1 = N - 1
170 FOR I = 1 TO N1
180   K = I
190   M = A(I)
200   I1 = I + 1
210   FOR J = I1 TO N
220     IF A(J) <= M GOTO 250
230     K = J
240     M = A(J)
250   NEXT J
260   A(K) = A(I)
270   A(I) = M
280 NEXT I
290 STOP
300 END
```

BenchMark 4: Reading Real numbers

The file being read resided on an RK05 disk, and contained 3600 lines, with 600 repetitions of the following set of six lines:

```
  25345.67
1235902.45
    123.23
     45.2
  63423.5
 -72854.3
```

---------- BenchMark 4 in Pascal: Read Reals (N = 3600)

```
program RealInput(input);
  var r: real;
begin
  while not eof do readln(r)
end.
```

---------- BenchMark 4 in FORTRAN: Read Reals (N = 3600)

```
        LOGICAL EOF
100     CALL CHKEOF(EOF)
        READ(1,1) R
1       FORMAT(F10.2)
        IF(EOF) STOP
        GOTO 100
        END
```

---------- BenchMark 4 in BASIC: Read Reals (N = 3600)

```
100 REM BM4.BA
110 FILE#1:"DSK:BM4.IN"
120 INPUT #1:A
130 IF END #1 GOTO 150
140 GOTO 120
150 STOP
160 END
```

BenchMark 5: Write Reals

The file being written resided on an RK05 disk.

---------- BenchMark 5 in Pascal: Write Reals (N = 3600)

```
PROGRAM REALOUTPUT(OUTPUT);
  VAR R: REAL;  I: INTEGER;
BEGIN
  R := 123456.78;
  FOR I := 1 TO 3600 DO WRITELN(R:15:2)
END.
```

---------- BenchMark 5 in FORTRAN: Write Reals (N = 3600)

```
        R = 123456.78
        DO 100 I = 1, 3600
100     WRITE(1,1) R
1       FORMAT(F15.2)
        STOP
        END
```

---------- BenchMark 5 in BASIC: Write Reals (N = 3600)

```
100 REM BM5.BA
110 A=123456.78
120 I = 0
```

```
130 FILEV#1:"DSK:BM5.OU"
140 PRINT #1:A
150 I = I + 1
160 IF I <= 3600 GOTO 140
170 STOP
180 END
```

BenchMark 6: Count characters in a text file.

This is adapted from Wirth, 1971. This is another example of where FORTRAN is at a disadvantage to Pascal. In particular, the timings for Pascal are sensitive to the actual line-lengths on the input file, whereas the FORTRAN program always sees 80 characters per line.

Three data files were tested, each containing 500 lines. The first had 2 characters on each line, the second had 40, and the third had 80. All files resided on an RK05 disk, with the outputs displayed on a 9600 baud crt screen.

---------- BenchMark 6 in Pascal: Character Count

```
PROGRAM CHARACTERCOUNT(INPUT, OUTPUT);
  VAR I: CHAR;
      C: ARRAY[CHAR] OF INTEGER;
BEGIN
  FOR I := ' ' TO '_' DO C[I] := 0;
  WHILE NOT EOF DO
    BEGIN READ(INPUT, I);  C[I] := C[I] + 1 END;
  FOR I := ' ' TO '_' DO WRITELN(' ', I, C[I])
END.
```

---------- BenchMark 6 in FORTRAN: Character Count

```
        INTEGER A(80), C(64)
        LOGICAL EOF
        DO 10 I = 1, 64
10      C(I) = 0
20      CALL CHKEOF(EOF)
        READ(1,1) A
1       FORMAT(80A1)
        IF (EOF) GOTO 50
        DO 40 J = 1, 80
        CALL CGET(A(J), 1, I)
        I = I + 1
40      C(I) = C(I) + 1
        GOTO 20
50      DO 60 I = 1, 64
        K = I-1
        CALL CPUT(CH, 1, K)
60      WRITE(4,2) CH, C(I)
2       FORMAT(1X, A1, I10)
        STOP
        END
```

---------- BenchMark 6 in BASIC: Character Count

```
100 REM BM6.BA
110 DIM L$(80), A(64)
120 FOR I = 1 TO 64
130   A(I) = 0
140 NEXT I
150 FILE#1:"DSK:BM6.2"
160 INPUT #1:L$
170 IF END#1 GOTO 250
180 L=LEN(L$)
190 FOR I = 1 TO L
200   X$ = SEG$(L$,I,I)
210   T = ASC(X$)
220   A(T) = A(T) + 1
230 NEXT I
240 GOTO 160
250 FOR I = 1 TO 64
260   PRINT CHR$(I);" ";A(I)
270 NEXT I
280 STOP
290 END
```

BenchMark 7: SIN/COS Repetition

---------- BenchMark 7 in Pascal: Sin/Cos (N = 1000)

```
PROGRAM FUNDAMENTAL(OUTPUT);
CONST N = 1000;
VAR X, Y: REAL; I: INTEGER;
BEGIN
  FOR I := 1 TO N DO
    BEGIN
      X := COS(I / N);
      Y := SIN(I / N)
    END
END.
```

---------- BenchMark 7 in FORTRAN: Sin/Cos (N = 1000)

```
      DO 100 I = 1, 1000
      X = SIN(I / 1000)
      Y = COS(I / 1000)
  100 CONTINUE
      STOP
      END
```

---------- BenchMark 7 in BASIC: Sin/Cos (N = 1000)

```
100 REM BM8.BA
110 FOR I = 1 TO 1000
120   X = COS(I / 1000)
130   Y = SIN(I / 1000)
140 NEXT I
150 STOP
160 END
```

Summary of Benchmarks

Two summary tables follow. The first compares the non-FPP products OS/8 BASIC and Pascal-S with non-FPP FORTRAN and non-FPP Pascal-OS/8. The second table compares FPP and non-FPP Pascal-OS/8 with FPP and non-FPP FORTRAN. All times are wall clock seconds to execute. "rel." stands for relative times, with FORTRAN IV (no FPP) being 1.00.

Table 3.5-1 non-FPP Comparison

BenchMark	Pascal-OS/8		OS/8-FORTRAN		OS/8-BASIC		Pascal-S	
	sec.	rel.	sec.	rel.	sec.	rel.	sec.	rel.
1. Matrix Multiply	1282	3.17	404	1.00	971	2.40	1: 781	1.93
2. Warshall's algorithm	26	0.05	480	1.00	727	1.51	830	1.73
3. Sort n=500	500	1.81	277	1.00	407	1.47	527	1.90
4. Read reals	39	0.53	73	1.00	63	0.86	19	0.26
5. Write reals	87	1.89	46	1.00	57	1.24	59	1.28
6.a Char count length=2	13	0.04	342	1.00	15	0.04	8	0.02
6.b Char count length=40	103	0.30	347	1.00	183	0.53	81	0.23
6.c Char count length=80	196	0.55	354	1.00	486	1.37	160	0.45
7. Sin/Cos N=1000	74	3.70	20	1.00	24	1.20	71	3.55

Notes:

1: The Pascal-S implementation would only support an N=45 matrix multiply, instead of the N=50 for the other tests.

Table 3.5-2 FPP Comparison

BenchMark	Pascal-OS/8 (with FPP)		Pascal-OS/8 (no FPP)		OS/8-FORTRAN (no FPP)		OS/8-FORTRAN 2: (with FPP)	
	sec.	rel.	sec.	rel.	sec.	rel.	sec.	rel.
1. Matrix Multiply	923 (3: 687	2.28 1.70)	1282	3.17	404	1.00	65	0.16
2. Warshall's algorithm	---		26	0.05	480	1.00	60	0.13
3. Sort n=500	---		500	1.81	277	1.00	27	0.10
4. Read reals	26	0.36	39	0.53	73	1.00	37	0.51
5. Write reals	40	0.87	87	1.89	46	1.00	39	0.85
6.a Char count length=2	---		13	0.04	342	1.00	58	0.17
6.b Char count length=40	---		103	0.30	347	1.00	62	0.18
6.c Char count length=80	---		196	0.55	354	1.00	67	0.19
7. Sin/Cos N=1000	10	0.50	74	3.70	20	1.00	3	0.15

Notes

"---" denotes timings that would not be different with an FPP, because there are no real numbers in the algorithm.

2: FORTRAN FPP timings are using an FPP-8a on a PDP8e bus, with lock-out set, available for FPP-8a only.

3: This timing is with range tests turned off. All other Pascal-OS/8 timings are with range tests enabled.

3.6. Hints and Notes.

3.6.1. Overcoming Memory Limits.

The amount of data that a Pascal-OS/8 program can handle is limited by the amount of memory available. The program code and data space share memory, of course. The data space is that space left over after loading the (largest segment of the) program. The data space is divided into stack and heap. Some ways to fit more data into your program are:

- Recompile with a smaller B option, or T- (run-time range tests off) or P- (no PMD information). See section 3.1.1.
- Use more physical memory, up to 32k in this version.
- If you have no FPP and do not need real numbers in your program, do not use any real numbers of any sort, and 1280 words (2400 octal) of memory will not be taken up by the FPP emulator. This does not apply if you have an FPP.
- Check into segmenting your program code. See section 3.1.3 for a guide on designing and implementing segmentation.
- Consider algorithm changes. Either a sequential data file or an indexed (direct access) data file can greatly enlarge data capacity, especially on a hard-disk system, but at a trade-off both in processing time and in programming effort. The program ldMap uses a direct file on 'DSK:' to accumulate identifier usage references during scanning of a Pascal program, and can handle quite a large Pascal program as a result.
- Obtain a KT8A and 64k or 128k of compatible memory, and inquire of the implementers/authors about implementing the KT8A for Pascal-OS/8. The addressing design will support a KT8A, and there are many hooks, but we did not implement actual support, due to the fact we did not have a KT8A and compatible memory.

3.6.2. Non-Text Files

Files declared as other than text-type (all three-way packed files including Text, file of char, file of ASCII, file of EightBit --see section 3.1.2.1) are not packed. That is, a physical block can contain only one element of the file. So a file declared as

```
F: file of integer;
```

will take up one disk block (256 words) for every file element -- every integer! A file element will take up as many blocks as needed to store it, so large elements (>256 words) are supported.

As a result of this implementation, we have found ourselves declaring a file element as a structured type, usually either

an array of 12-bit words, or an array of records,
and end up manipulating two levels of addressing to access words or records in
a file; first the element number on the file (block number) and then the array index within the block (word or record number). Programs IdMap,
SvUtil, INDXED and CopyV all use this approach.

This was not our desired implementation, but time constraints have limited us.

3.6.3 Bartering for Time

There are a few ways to speed up the execution of Pascal-OS/8 programs.

- Use the compiler B option to allocate longer Input/Output buffers. This can improve the processing of text and non-text sequential files. This uses more data space. See section 3.1.1.

- Allocate files on more than one physical device to avoid excessive head movement, if the devices have the same data transfer rate.

- The compiler (and any segmented program) will run faster with more memory available (up to the point where all active segments simultaneously fit in memory).

- Use the compiler T- option to turn off run-time tests. This saves about 10% of the T+ execution time. See section 3.1.1.

- Use the compiler N- to turn off line-number references. The compiler generates code to keep track of the current line number while the program is executing, unless the N-switch (or P-) is in effect. This save a slight amount of execution time. Note that P+,N- (PMD enabled but without line numbers reports by address) is equivalent in execution speed to P-. See section 3.1.1.

- Use the compiler I- option if your program uses text files but does not do any interactive I/O. This saves a slight amount of time by turning off interactive I/O support, saving one call the to the run-time-system I/O routines for each character read by the program. See section 3.1.1.

- Array referencing is slow in this implementation of Pascal. Avoid arrays if any other data structure will solve the problem. If your algorithm is suitable for a dynamic data structure (built with pointers) rather than an array, this implementation of Pascal will do considerably better.

- Finally, study your algorithms and search for more efficient ones. This implementation of Pascal is very good at exposing inefficient algorithms, and encouraging one to improve them, if at all possible.

5. Installation and Support.

This section describes the installation procedures, hardware adaptation and configuration procedures, and support procedures for Pascal-OS/8.

5.1. Parts List: Files and Document in Pascal-OS/8 Kit.

	Source	Binary
Pascal-OS/8 System:		
Run-Time System (P8RTS)	-	P .SV
Pascal Compiler	-	PASCAL.PB
Post-Mortem Display (error traceback)	-	PMD .PB
Compiler Error Message Texts	PASCAL.TX	
Save File Header for code files	-	P8HEAD.SV
Test program, simple	TEST01.PS	TEST01.PB
Pascal Tools:		
Pascal Program Cross Referencer	IDMAP .PS	IDMAP .PB
Procedure and Function Mapper	PFMAP .PS	PFMAP .PB
Decode a Pascal-OS/8 .PB file	-	PASCOD.PB
Binary Editor for .SV files	SVUTIL.PS	SVUTIL.PB
Input for SVUTIL to setup P for SSRFC	SSRFC .OD	
Example Programs:		
Compare two text files	COMPAR.PS	COMPAR.PB
Copy OS/8 file with verify	COPYV .PS	COPYV .PB
FORTRAN IV cross-referencer (simple)	F4REF .PS	F4REF .PB
Hilbert curves on TEK scope	HILBER.PS	
Character count	CHCNT .PS	
Copy eight-bit bytes	COPY8 .PS	
Copy text file	COPYTX.PS	
Long Lines reporter	FINDLL.PS	
Formatted read real & integer	FREAD .PS	
Rename identifiers in a Pascal source	ID2ID .PS	
Test program for Indexed Files, example	INDXED.PS	
Process LINK maps for intelligibility	LINKMP.PS	
Unformatted read real & integer	READNU.PS	
General Symbol Cross Referencer	SYMREF.PS	
Transcendental function PDP8 algorithms	TRANSC.PS	
Toy for VT100 Scope	VTWALK.PS	
Binary tree algorithm	XREF45.PS	
Documentation Files		
Short list of files on floppy 1 of 2	README.1	
Short list of files on floppy 2 of 2	README.2	
Errata list (none if not included)	ERRATA	

Printed Documentation:

Pascal-OS/8 Reference Manual, about 134 pages with Appendices.

5.2. Installing Pascal-OS/8.

Installation consists of copying certain files from the distribution media (normally two floppy disks) onto your OS/8 system SYS: and DSK:. Only two files are absolutely necessary: P.SV (the run-time-system) and PASCAL.PB (the Pascal compiler). Two additional files, PMD.PB and PASCAL.TX enhance error reporting a great deal. PMD is a post-mortem display (error traceback report), and PASCAL.TX is a text file that the compiler uses for reporting its error message explanations. A fifth file, IDMAP.PB, is useful for creating a cross-reference map (IDentifier MAP) of Pascal program sources.

To install Pascal-OS/8, do: (assuming RXA0: is the distribution floppy)

```
.COPY SYS:<RXA0:P.SV,PMD.PB,PASCAL.TX
.COPY SYS:<RXA0:PASCAL.PB,IDMAP.PB
```

Pascal-OS/8 is now ready for use.

Substitute your own device name for RXA0: in the above command lines.

Three new OS/8 file name extensions are used for Pascal-OS/8:

.PS	PascalSource (text file, editable with TECO, EDIT, etc.)
.PB	PascalBinary (code file, ready for RTS to execute)
.PM	PostMortem Dump file, PMD trace-back upon error.

Also, Pascal-OS/8 uses some standard OS/8 extensions:

.LS	Listing files (from compiler, IDMAP, PFMAP, others).
.TM	Temporary files (internal files, scratch files).

It is recommended that you make a back-up copy of all files in the Pascal-OS/8 distribution kit.

Table 5-1 specifies recommended device residencies for the Pascal-OS/8 kit. It is possible to run Pascal-OS/8 with nothing on SYS: (although the run-time system must then be on a device that is co-resident with SYS:). However, the recommended residencies will provide the easiest usage.

Table 5-1

Recommended Device Residency Table

File	Rename? (#4)	Mandatory Device	Recommended Device	Alternate Device
P .SV	ok (#6)	SYS: or Co-Resident (#5)	SYS: (#1)	Co-Resident with SYS:
PMD .PB	no (#2)	SYS: (#1)		none
PASCAL.PB	ok	-	SYS: (#1)	any (#1)
PASCAL.TX	no (#3)	SYS:		none
utilities (IDMAP.PB,etc.)	ok	-	any (#1)	any (#1)

Notes:

#1: All code files must reside on a file-structured device (random access device). A random access device would be a RX01, RX02, RK05, RL01, or any similar disk normally set up as a file-structured device. The requirement of random-access for Pascal-OS/8 code files (.PB) is due to the Segmented code design, and is enforced by checking the file-structured bit for the device at the time the .PB file is opened. A fast disk (RK05, RL01 or equivalent) is recommended.

#2: SYS:PMD.PB is looked up by the run-time-system (P.SV) to compute the Post-Mortem Display. If not present on SYS:, no traceback is performed.

#3: SYS:PASCAL.TX is looked up by the compiler (PASCAL.PB) to provide error message explanations. If not present on SYS:, no explanations are printed.

#4: "Rename?" column states if a file can be renamed without affecting the Pascal-OS/8 system workability.

#5: If the RTS (P.SV) is on SYS: (recommended), then calling the RTS looks like:

```
.R P                    -- or --        .R P,prog
%prog
```

But if the RTS is on a device co-resident with SYS:, such as RKB0:, the calling statements will be:

```
.RU RKB0:P              -- or --        .RU RKB0:P,prog
%prog
```

where prog is the name of a Pascal-OS/8 code file (.PB).

#6: Unless P8HEAD feature is desired. This is a .SVfile header that can be concatenated onto .PB files (in front) and enable .RUNing of Pascal-OS/8 programs. See sections 3.2.3. and 5.4.2 P8RTS Patch to P8HEAD.

Example Programs.

There are several example Pascal programs with the kit, some of which are accompanied by their code files (.PB). These programs range from simple (COPYCH) to complex (IDMAP), and from portable, standard Pascal (COMPARE) to specialized programs using Pascal-OS/8 extensions (COPYV).

You may get a useful cross-referenced listing of any program with IDMAP (IDentifier MAP):

```
.R P,SYS:IDMAP,program,LP:
```

5.3. Adapting Pascal-OS/8

The Pascal-OS/8 Run-Time-System (P8RTS, file P.SV) makes automatic adaptations to the host machine each time it is run. The five major adaptations are: (1) memory size, (2) use of an FPP (floating point processor), (3) control-C traps, (4) managing a two-page system handler, and (5) OS/8 BATCH management.

5.3.1. Memory Adaptation.

Pascal-OS/8 RTS will use the OS/8 memory size as set by the MEMORY command, unless that setting is zero, in which case the actual amount of memory available is used. The actual amount is measured by a routine provided by Jim Van Zee.

If the OS/8 memory setting is non-zero, and if the actual amount of memory differs from the OS/8 memory setting, P8RTS will use the OS/8 memory setting, if there is enough hardware memory, otherwise it will use the hardware amount.

Three fields (12k) of memory are the minimum amount needed for P8RTS execution. Field 0 is mostly I/O support, field 1 is mostly the interpreter, and field 2 and up are used for the user program and data. If real numbers are used and there is no FPP on the bus, then 2400 (octal) words are used to contain the FPP simulator.

5.3.2. FPP Adaptation

When the RTS first executes, it tests for an FPP present on the host machine, and remembers this fact for the duration of the RTS execution.

Each time a program is executed, the run-time checks the program header to see if the program uses real numbers (determined at compile time). If the program does not use real numbers, nothing further happens. Otherwise the FPP or an FPP simulator is used. If a hardware FPP (FPP-8A or FPP-12) is available, it is initialized and used for all real number operations. Otherwise, a FPP simulator is loaded into memory to be resident during the life of the current program. This simulator takes up approximately 2400 base 8 (1280 base 10) words of memory that would otherwise be available to the user program.

5.3.3. Control-C Traps

P8RTS sets execution traps at OS/8 locations 07600 and 07605 in order to regain control if an OS/8 handler intercepts a control-C (P8RTS does not enable interrupts), and perform clean-up operations, such as restoring a two-page system handler, or aborting OS/8 BATCH.

5.3.4. Two-Page System Handler

If the RTS thinks the system device handler is a two-page handler, such as an RX02 system handler, it will re-locate the second page of the handler to the highest used memory field (top page, of course)

in order to have the largest possible continuous memory for the Pascal program. The two-page system handler will be restored upon all exit paths: normal program completion, program error, program 'Halt', RTS errors and control-character aborts (^A, ^D, ^C).

5.3.5. BATCH Management

Pascal-OS/8 is designed to be executable under OS/8 BATCH Version 7A.

If OS/8 BATCH is running, P8RTS saves the state of BATCH, and restores it when returning to OS/8. This is simply saving and restoring the four-word BATCH state block at n7774. Upon any error, BATCH is aborted via the documented entry point, but control-C turns off BATCH completely.

P8RTS can find its command lines from the BATCH stream, but cannot offer user program access to read the BATCH stream due to the complexities of memory management that would be necessary.

Executing under BATCH affects the internal device handler IO: (the default binding device for program files 'input' and 'output'). IO: is equivalent to TT:, the internal console handler, if not under BATCH, or under BATCH but not sending the BATCH log to the line printer. If under BATCH with the log going to the printer, the device IO: is equivalent to LP:, the internal line printer handler. Table 5-2 summarizes this.

Not under BATCH:

	Effective Binding
Reset(input,'IO:')	TT: keyboard
Rewrite(output,'IO:')	TT: display

Under OS/8 BATCH control:

BATCH log on:	Console	Line Printer
Reset(input,'IO:')	NULL:	NULL:
Rewrite(output,'IO:')	TT: display	LP:

Table 5-2. IO: Device Binding Rules.

5.4. Configuring Pascal-OS/8

There are no mandatory "patches" for Pascal-OS/8. While Pascal-OS/8 is designed to be as easy and automatic as possible to install, there may be needs for specific PDP8 (or PDP12) sites and machines which are addressed here.

Configuration consists of manually patching the run-time-system (P.SV, called P8RTS) using the utility SVUTIL, which is provided. If you are unfamiliar with patching a .SV file, I suggest you try only one item at a time, as needed.

P.SV is an overlaid OS/8 save file, generated with MACREL (V2C) and LINK (V2AG). SVUTIL was created to perform correct overlay addressing and modification, and is essentially a variation of ODT and FUTIL. If you have a version of FUTIL (OS/8 extension kit) that you know performs correct address calculations for OS/8 SAVE files that have overlays, you can use it in place of SVUTIL for the patches described here.

5.4.1. Using SVUTIL

You should avoid modifying the run-time-system that is executing/interpreting SVUTIL!

** Always keep a "stock" (un-patched) copy of the run-time-system as a backup **.

SVUTIL (or FUTIL) modifies the SAVE file by using random-access rewrite-in-place on the disk file you specify to it.

I make my patches by:

```
.COPY SYS:X.SV<SYS:P.SV

.R P,SVUTIL,SYS:X.SV
```

One copy of the run-time-system (SYS:P.SV) is executing SVUTIL, while SVUTIL can modify another copy (SYS:X.SV, the default file name for SVUTIL to operate on).

The control statement:

```
.R P,SVUTIL,SYS:X.SV
```

will execute SVUTIL, and it will set up to access SYS:X.SV . The first thing printed will be the name and length of the file to be patched. SVUTIL commands parallel those of OS/8 ODT (or FUTIL ODT) very closely. Use Q (quit) to leave SVUTIL; the last modified block will be written out automatically. Patch 1 shows a complete example of SVUTIL usage. See Appendix T.7 for a more complete description of SVUTIL.

The file SSRFC.OD in the distribution kit is an example input to SVUTIL that sets up the run-time-system for our PDP8 at SSRFC. This patch file takes advantage of a pointer at location 10006 in the run-time-system which gives the address of the ENVIRON data sect.

ENVIRON begins with four version words:

```
word1:  bit0-5: release;       bit6-11: update
word2:  bit0-11: level
word3:  bit0-5: fix#;          bit6-11: patch#
word4:  bit0-5: code version;  bit6-11: operating system
```

The patch file then uses known offsets into ENVIRON. This allows the patch file to work after the RTS is re-LINKed, which causes shifts in SECT addresses. You won't have to worry about these shifts, because you will not be relinking the RTS, and you can use the absolute addresses given here.

5.4.2. Configuration Patches

Most (but not all) locations in the run-time-system that are usefully patched for configuration are in one data section known as ENVIRON, located in field 0 (level 0 root).

The configuration patches are:

```
Patch 1: FPP Disable/Enable (HDW1A)
Patch 2: RIO: Device Codes (RIODEV)
Patch 3: TEK: Device Codes (TEKDEV)
Patch 4: PSW2 (Program Status Word 2)
Patch 5: PSW4 (Program Status Word 4)
Patch 6: MEMLWA Memory Last Word Available
Patch 7: Prompt character for RTS
Patch 8: TMDV - Temporary File Name
Patch 9: PMD Related File Names
Patch 10: PMD Width of Output Device
Patch 11: PRS (PreSet) Installation message
Patch 12: Execute CCL File Name
Patch 13: Internal Handler Name Table
Patch 14: LP: Device Codes
Patch 15: P8HEAD name of P8RTS
```

In the following, a '#' marks a patch, whose address is double checked before distribution.

Patch 1: FPP Disable/Enable (HDW1A)

After making a copy of P.SV, as described above, to disable P8RTS use of FPP hardware when it exists:

```
        .R P,SVUTIL,SYS:X
#       0.4723/ 7676  7600
        Q
```

To re-enable P8RTS use of FPP hardware:

```
        .R P,SVUTIL,SYS:X
```

```
#       0.4723/ 7600  7676
        Q
```

Patch 2: RIO: Device Codes (RIODEV)

The internal device handler RIO: can be re-directed to the KL8 device of your choice; the default is device 40 keyboard and device 41 printer-display. This device pair is expressed in one word as [6/keyboard IOT, 6/display IOT], and is used to code-modify the IOTs used for the RIO: handler each time the RIO: handler is first bound to a file during a program execution.

(The TT: and LP: internal handlers are permanently fixed to the device code pairs 0304 and 6566, respectively. But see patch 14 for LP:.)

The RIO: handler can be patched with this one-word patch to any reasonable device code pair that is actually a TTY-like controller board (KL8, KL8E, KL8J, etc.) by:

```
#       0.4727/ 4041  nnnn
```

where nnnn could be 3031 for a KL8 setup with keyboard IOT device = 30, display IOT device = 31.

Patch 3: TEK: Device Codes (TEKDEV)

The internal device handler TEK: can be re-directed to the KL8 device of your choice; the default is device 42 keyboard and device 43 printer-display. This device pair is expressed in one word as [6/keyboard IOT,A6/display IOT], and is used to code-modify the IOTs used for the TEK: handler each time the TEK: handler is first bound to a file during a program execution.

The TEK: handler can be patched with this one word patch to any reasonable device code pair that is actually a TTY-like controller board (KL8, KL8E, KL8J, etc.) by:

```
#       0.4730/ 4243  nnnn
```

where nnnn could be 3031 for a KL8 setup with keyboard IOT = 30, display IOT = 31.

Patch 4: PSW2 (Program Status Word 2)

Various bits in PSW2 and PSW4 control switch settings within the run-time-system software. To alter these bits requires real bit-twiddling. To alter a bit, you must use the current value of the word (PSW2 or PSW4) and compute the difference when changing one (or

more) bits. For instance, to enable the OS/8 date format, do:

```
        .R P,SVUTIL,SYS:X
#       0.4732/3606  3607          (selects OS/8 date format)
        Q
```

The new value combines BITDAT (0001) with PSW2 (3606) giving 3607. Note this is a Boolean operation first masking out the bit position (0001) and then adding (or ORing) in the desired bit value (0001).

Following is a description of the bits.

```
#       0.4732/ 3606          PSW2 original value
```

word	bit	name	default	description
PSW2	& 4000	BITINF	0	enables informative messages if set.
	& 2000	BITVMS	1	enables RTS version message if set.
	& 1000	BITLBM	1	enables load of BATCH monitor if set; this occurs when aborting the current BATCH job; if BITLBM is not set, BATCH is aborted by turning it off completely.
	& 0400	BITEXC	1	enables Execute of CCL statements if set.
	& 0200	BITEXR	1	enables Execute of RTS statements if set.
	& 0100		0	do not alter
	& 0040		0	do not alter
	& 0020		0	do not alter
	& 0010		0	do not alter
	& 0004	BITPRR	1	enables pre-read of directory devices to determine readability before continuing with I/O; takes time but allows better diagnostic messages when opening a file on a device that is not ready.
	& 0002	BITPRW	1	enables pre-write of directory devices to determine if the device is write-locked before continuing with Input/Output; takes time but allows better diagnostic messages when opening an extend-mode file on a write-locked device.
	& 0001	BITDAT	0	selects date format for date procedure. 0=ISO 'yyyy-mm-dd'; 1=OS/8 ' dd-mmm-yy'.

Patch 5: PSW4 (Program Status Word 4)

See discussion under PSW2, above.

```
#       0.4734/ 1200          PSW4 original value.
```

word	bit	name	default	description
PSW4	& 1000	BITCSI	1	enable CS: handler if set; does not save CS: on a scratch block if BITCSI is 0, saving one I/O

```
                    access during RTS startup.
      & 0200 BITLOG 1  if set, use LP: for echoing if BATCH is running
                       on the line printer else use BATCH log device to
                       echo the command statement and use LP: for
                       program Input/Output.
      & 0020 BITPMC 0  use the load file name for .PM (PMD dump file)
                       if set, else use :PASCAL.PM.  PMD expects
                       the dump on :PASCAL.PM
```

Do not change other bits in PSW4.

Patch 6: MEMLWA Memory Last Word Available

MEMLWA is zero by default but if patched to a non-zero (and sensible value) will limit the run-time-system to the amount of memory given. The OS/8 .MEMORY command is normally used for this purpose; MEMLWA had uses during debugging.

```
#       0.4740/ 0000              12/low bits
#       0.4741/ 0000              7/0,5/high
```

For example, to prevent the RTS from using any address above 57577, do:

```
#       0.4740/ 0000      7577
#       0.4741/ 0000      0005
```

Patch 7: Prompt character for RTS

The prompt character for the RTS in job-step mode is '%' (ASCII= 045 octal). To change it:

```
#       0.5000/ 0045 xxxx
```

where xxxx is the octal code for the ASCII character desired. For example

```
#       0.5000/ 0045 0077
```

changes to a question mark '?'.

Patch 8: TMDV - Temporary File Name

The run-time-system uses a fixed device to allocate temporary files (program file variables that are not named in the program statement or created with the form "reset(f,string)" or "rewrite(f,string)"). This device is the same as the device from which the run-time-system

is executed. If P8RTS is run from SYS: (.R P) then temporary files (.TM files) are allocated on SYS:. If P8RTS is run from non-SYS co-resident (.RU RKB0:P) then temporary files are allocated on the non-SYS co-resident device.

The temporary-device may also be set to any file-structured device in the OS/8 system by patching TMDV as follows:

```
#       0.4772/ 0000      nnnn
#       0.4773/ 0000      nnnn
```

where the two numbers are the packed 6-bit representation of the device name, such as 2303; 2200 for "SCR:". If these two locations are zero (default), the temporary-device is the run-time-system device.

Further, the temporary file name may be changed, if desired. It immediately follows the temporary-device words:

```
#       0.4774/ 2001    'PA
#       0.4775/ 2303    'SC
#       0.4776/ 0114    'AL
#       0.4777/ 2415    'TM
```

This file name, 'PASCAL.TM' is used for temporary file names, non-external files that are allocated by the user program. These temp file names start with this name and are incremented by the run-time-system in the sequence PASCAM.TM, PASCAN.TM, PASCAO.TM, ..., ZZZZZZ.TM. The name is actually incremented to PASCAM.TM before the first use. The device and extension may be changed to any sensible names, and the start temp file name may be changed, but it must consist of letters only, and be a full 6 characters. Note that the name 'PASCAL.PM' is used by PMD for the Post-Mortem-Dump file.

Patch 9: PMD Related File Names

Other file names may be substituted here if necessary; after an error, the RTS will use these names to build the control statement string. Possibly one or both of these files could be on a scratch space other than SYS:.

```
        PMDFIT+FITDN:
#  3.13025/ 2331     'SY       'SYS:PASCAL.PM' -- name of dump file.
#  3.13026/ 2300     'S@       becomes 'SYS:loadfilename.PM' if
#  3.13027/ 2001     'PA       PSW4&BITPMC is set.
#  3.13030/ 2303     'SC
#  3.13031/ 0114     'AL
#  3.13032/ 2015     'PM

        PMDPBN:
#  3.13040/ 2331    'SY       'SYS:PMD.PB' is the expected file
#  3.13041/ 2300    'S@       name for the PMD program.
#  3.13042/ 2015    'PM
```

```
#  3.13043/ 0400    'D@
#  3.13044/ 0000    '@@
#  3.13045/ 2002    'PB
```

Patch 10: PMD Width of Output Device

The PMD report is produced in multiple columns if there is enough room. The line width PMD will write is controlled by three numeric ASCII characters in the RTS PMD overlay. These three characters are set to '071' as the default width (0060;0067;0061 in octal ASCII). If both the console (TT:) and line printer (device 66, LP: internal handler) have at least 80 columns, a patch would be:

```
#       3.13267/ 0060
#       3.13270/ 0067 "8
#       3.13271/ 0061 "0
```

after which the three characters are '080'. Reasonable values for these three characters would be '032' up to perhaps '133'.

These ASCII characters become part of the control statement constructed internally by the RTS/PMD to invoke PMD.PB. (The next six characters are blanks, then a zero word to terminate the string.)

Patch 11: PRS (PreSet) Installation message

```
# 1.12703/ 0000          1.12703 through 1.12714 are available for
                         a customized version message. Place ASCII
                         characters, one per word, here. There is
                         room for ten (decimal) characters. The last
                         word plus one:
# 1.12715/ 0000 ** must remain zero ** to terminate the message.
```

Patch 12: Execute CCL File Name

The file that the Execute('.string') will call, normally SYS:CCL.SV, can be patched in the following locations.

```
# 3.17010/ 0303   'CC
# 3.17011/ 1400   'L@
# 3.17012/ 0000   '@@
# 3.17013/ 2326   'SV
```

Patch 13: Internal Handler Name Table

The names of the internal handlers may be changed with patches to this table. The following gives the correspondence of addresses with names. The table ends with a zero word (the last entry may be deleted by putting a zero word in the first word of its file name). A device name may be from one to four characters long, using alphabetic and numeric characters only (A-Z, 0-9). The device codes are listed with the handler names/descriptions for references; no IOT codes are in this table. (This is Dsect IHTABL in module LDHAND.MA).

```
  addr     octal   packed      handler name

# 5.05764/ 1117   'IO         IO:     Input/Output stream
  5.05765/ 0000   '@@

# 5.05774/ 2424   'TT         TT:     Console (03,04)
  5.05775/ 0000   '@@                 buffered input

# 5.06004/ 2424   'TT         TT:     Console (03,04)
  5.06005/ 6000   '@@                 immediate input

# 5.06014/ 1420   'LP         LP:     Line Printer (65,66)
  5.06015/ 0000   '@@                 buffered input

# 5.06024/ 1420   'LP         LPO:    Line Printer (65,66)
  5.06025/ 6000   'O@                 immediate input

# 5.06034/ 0323   'CS         CS:     control statement
  5.06035/ 0000   '@@

# 5.06044/ 2024   'PT         PT:     Paper Tape (01,02)
  5.06045/ 0000   '@@

# 5.06054/ 1625   'NU         NULL:   Null Device
  5.06055/ 1414   'LL

# 5.06064/ 2211   'RI         RIO:    Remote Input/Output (40,41)
  5.06065/ 1700   'O@                 buffered input

# 5.06074/ 2211   'RI         RIOO:   Remote Input/Output (40,41)
  5.06075/ 1760   'OO                 immediate input

# 5.06104/ 2405   'TE         TEK:    Tek Scope (42,43)
  5.06105/ 1300   'K@                  buffered input

# 5.06114/ 2405   'TE         TEKO:   Tek Scope (42,43)
  5.06115/ 1360   'KO                 immediate input

# 5.06124/ 0711   'GI         GIN:    Tek Scope Graphic Input (42,--)
  5.06125/ 1600   'N@                  buffered input, no echo

# 5.06134/ 0000       --- end of Internal Handler Table ---
```

Patch 14: LP: Device Codes

The LP: handler is really a normal ASCII terminal handler, very much like RIO: or TEK:. The device codes for those handlers is changed in one place, however, and the run-time spreads the desired IOT around when initializing. The LP: handler has IOT codes that are more fixed, but highly localized. The following patches will change the LP: handler to another terminal-like device. Assuming xx is the keyboard IOT code,

```
#       0.5132/ 6652 6xx2
#       0.5110/ 6651 6xx1
#       0.5111/ 6656 6xx6
```

and yy is the display IOT code:

```
#       0.5115/ 6661 6yy1
#       0.5116/ 6666 6yy6
#      1.14537/ 6666 6yy6
#      1.14543/ 6661 6yy1
```

(Note: the last two in overlay 1, field 1, are where we test for the existence of a line printer, in a test similar to OS/8 FORTRAN IV.)

Patch 15: P8HEAD name of P8RTS

Words 12-15 in P8HEAD.SV contain the expected P8RTS file name (in FILENAME format) on SYS:. In order to rename the run-time-system and still allow the P8HEAD feature to function, P8HEAD.SV (and each already concatenated .SV/.PB file) must be patched as follows.

```
        .R P,SVUTIL,P8HEAD
#       12/ 2000
#       13/ 0000
#       14/ 0000
#       15/ 2326
        Q
```

Patch 16: RTS Tracer

This patch is only for using the execution tracer with the RTS during debugging. ION=6001.

```
#       1/7000 6273
#       2/4427 5403
#       3/0060 6600
```

5.5. Installation Checkout Test

The distribution of Pascal-OS/8 contains the program "TEST01.PS" with a code file. To test that the Pascal-OS/8 system is working minimally, do:

```
.COPY DSK:<distribution-device:TEST01.PS,TEST01.PB
.R P,TEST01
```

The greeting "Welcome to Pascal-OS/8!" should be printed on the console.

To ensure the compiler is working minimally, do:

```
.R P,SYS:PASCAL,TEST01
.R P,TEST01
```

and the same greeting should print.

To find out if the Pascal run-time can print on your line printer, do:

```
.R P,TEST01,LP:
```

or try:

```
.R P,TEST01,LPT:
```

5.6. Service Policy and Procedures

If problems arise that indicate a failure in the Pascal-OS/8 system, we invite you to write to us with a description of the problem. We do not guarantee support, however, due to limited resources at our office. We may fix a problem or incorporate a suggestion into a future release of the software, but we do not guarantee any future version.

The following information should be included in your letter:

- -- The version number of the Pascal-OS/8 system you are using.
- -- The version of OS/8 you are running.
- -- The hardware configuration you are running (processor type, PDP8, 8-L,I,E,F,A, PDP12 or other; amount of memory; FPP12 or FPP8a).
- -- A detailed description of the problem.

If the problem involves a complex program, please try to find a simple case example that demonstrates the failure.

Also, we would be happy to receive comments about any of the following areas.

1. Bugs and problems.

2. Design quirks and goofs.

3. Hardware incompatibilities.

4. Documentation weaknesses. Due to limitations, we have been unable to do as much work on documentation as we wanted to.

5. Comments on our extensions to Pascal; suggestions for extensions.

6. Comments on the limitations of this implementation.

7. Compatibility with other implementations you are using. Note that our major design goal was to implement a nearly complete, standard Pascal.

8. Is it too slow for some of the applications you try? We have had to get used to slow compilation by thinking and editing carefully, which is actually a good discipline.

9. Whatever other comments you care to make.

To contact us:

Pascal Group c/o John Easton
Social Science Research Facilities Center
25 Blegen Hall
269 19th Avenue South
University of Minnesota
Minneapolis, Minnesota 55455

(612) 373-7525 John Easton
(612) 373-5599 SSRFC

ASCII Character Codes

Octal

	00	20	40	60	100	120	140	160
0	NUL	DLE		0	@	P	¢	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
10	BS	CAN	(	8	H	X	h	x
11	HT	EM	)	9	I	Y	i	y
12	LF	SUB	*	:	J	Z	j	z
13	VT	ESC	+	;	K	[	k	{
14	FF	FS	,	<	L	\	l	¦
15	CR	GS	-	=	M	]	m	}
16	SO	RS	.	>	N	^	n	~
17	SI	US	/	?	O	_	o	DEL

Decimal

	00	10	20	30	40	50	60	70	80	90	100	110	120
0	NUL	LF	DC4	RS	(	2	<	F	P	Z	d	n	x
1	SOH	VT	NAK	US	)	3	=	G	Q	[	e	o	y
2	STX	FF	SYN		*	4	>	H	R	\	f	p	z
3	ETX	CR	ETB	!	+	5	?	I	S	]	g	q	{
4	EOT	SO	CAN	"	,	6	@	J	T	^	h	r	¦
5	ENQ	SI	EM	#	-	7	A	K	U	_	i	s	}
6	ACK	DLE	SUB	$	.	8	B	L	V	¢	j	t	~
7	BEL	DC1	ESC	%	/	9	C	M	W	a	k	u	DEL
8	BS	DC2	FS	&	0	:	D	N	X	b	l	v	
9	HT	DC3	GS	'	1	;	E	O	Y	c	m	w	

ACK	Acknowledge
BEL	Bell
BS	Backspace
CAN	Cancel
CR	Carriage Return
DC1	Device Control 1 (X-ON)
DC2	Device Control 2
DC3	Device Control 3 (X-OFF)
DC4	Device Control 4
DEL	Delete
DLE	Data Link Escape
EM	End of Medium
ENQ	Enquiry
EOT	End of Transmission
ESC	Escape
ETB	End of Transmission Block
ETX	End of Text
FF	Form Feed
FS	File Separator
GS	Group Separator
HT	Horizontal Tab
LF	Line Feed
NAK	Negative Acknowledge
NUL	Null
RS	Record Separator
SI	Shift In
SO	Shift Out
SOH	Start of Heading
STX	Start of Text
SUB	Substitute
SYN	Synchronous Idle
US	Unit Separator
VT	Vertical Tab

B. Known Bugs in Version 1-0-F

The following problems are known to exist in the current release, Version 1-0-F, of Pascal-OS/8.

B.1. Compiler Bugs

-- the compiler parses real numbers with slightly less precision than the read-real routine in the run-time-system (RDRRT).

B.2. Run-Time-System Bugs

-- CLOSE(F,I) on a reset indexed file is considered an error due to the second (index) parameter but this error is not detected; the parameter is ignored and harmless.

-- Severe P8RTS system errors during OS/8 BATCH operation can cause infinite re-processing of some errors.

-- Use of OS/8 LPT: handler may cause re-loading and re-initializing of LPT: upon every print buffer. Use LP: instead, if it drives your printer adequately. Or use fewer simultaneous devices, and the LPT: handler may not get re-loaded. See also OS/8 DSN for October/November 1981, patch "SEQ 73.40.03 N Lineprinter Handler Modification (EL)".

-- If in job-step mode and a .SV/.PB (P8HEAD) file is given, after the program terminates the RTS will return to the KBM instead of to job-step mode unless ^A, ^D, halt or error.

B.3. Tools Bugs

(none listed)

C. Restrictions

The following are the system restrictions of importance in OS/8 Pascal.

C.1. Compiler Restrictions

-- The compiler limits identifier significance to eight characters. Longer identifiers are allowed but are not distinguished beyond eight characters.

-- The word 'otherwise' is a reserved word.

-- The ordinal of the first character in the type char is not zero (ord(firstch) <> 0). Ord(firstch) = 32.

-- A pointer to a file is not allowed.

-- Sets may have at most 96 members. Set of char is supported.

-- See also Appendix V for detailed restrictions in Conformance, and Error Handling tests.

C.2. Run-Time-System Restrictions

-- .PB files (code files) must be generated (by the compiler) and executed (by the run-time-system) on a file-structured-device. (The code file is accessed randomly.)

-- Memory adaptation will not currently go beyond 32k (8 fields). The KT8A is not supported.

-- Separate compilation and libraries are not supported.

-- There is no method of access to assembly language level of programming from a Pascal program.

-- OS/8 does not allow a delete of one file on a device that already has a temporary file open without losing the temporary file.

-- OS/8 does not allow more than one file open in extend mode (USR enter function) on a device at one time.

-- Temporary files are not deleted if there is an error termination.

C.3. Tools Restrictions

(none listed)

E. Run-Time-System Messages

This appendix gives explanations for various messages that the Pascal-OS/8 run-time system may issue.

E.1. Explanation of version numbers

The version numbers of Pascal-OS/8 have the general form

digit - digit - letter

possibly followed by additional characters. For example, the current version is "1-0-F". The first digit defines the release number, the second digit defines the update number, and the letter defines a level of development of the release and update. Thus, the current version is release 1, update 0, level F.

The digit-digit-letter for the run-time-system and the compiler must agree for compatibility of the system. Any additional fields in a version number are relevant to the component (such as the run-time-system) that is reporting. Compatibility is also checked for PMD and for the code files. If you are receiving a second version of Pascal-OS/8, you may need to recompile all programs to use them (your old code files may not be accepted by later versions).

E.2. Informative Messages

'P8RTS 1-0-F E'

This is the version message that the run-time-system issues when it is run. If a third letter is printed (such as 'V1-0-F EC'), a SVUTIL/FUTIL patch has been applied to fix bugs, and the letter indicates the latest patch.

'OS/8 memory is not defined (0).'

This is a debug facility message, and normally will not occur. OS/8 .MEM command has not been set; OS/8 memory amount is set to zero. P8RTS will use the actual hardware amount available.

'OS/8 memory differs from hardware.'

This is a debug facility message, and normally will not occur. There is either more or less actual memory than the OS/8 .MEM command has set; P8RTS uses the actual amount if OS/8 says there is more than actual, or uses the OS/8 amount if it is less than or equal to the actual.

'MEMLWA jam-set'

This is a memory facility debug message only and occurs if ENVIRO.MEMLWA is patched to a non-zero value.

E.3. Batch Error Messages

'#Batch Abort'

Result of a control-C while executing under BATCH, or severe BATCH monitor error. Indicates that BATCH has been completely shut off.

'#Batch Job Aborted'

Normal abort of BATCH (jmp to BATCH Monitor abort entry point). This occurs upon a control-D, control-A, Halt, program error or system error, and causes BATCH to skip the rest of the current job and attempt to start a subsequent job ('$JOB') in the BATCH stream.

E.4. Severe Run-Time-Errors

These are some severe conditions that cause an abbreviated error message from the RTS. These will occur when the overlay driver is impaired, and an error is encountered.

'***P8RTS ERR A'

System error: the P8RTS internal overlay driver was called but was not initialized.

'***P8RTS ERR B'

System error: the overlay driver encountered a read error from the overlay device (SYS: or co-resident).

'***P8RTS ERR C'

Too many P8RTS errors. If the RTS gets into a bad state and is processing too many errors in a row, we abdicate.

'***P8RTS ERR D'

Reserved.

'***P8RTS ERR E'

System error: during initial command statement processing, the run-time-system name was not found on the device indicated.

'***P8RTS ERR F'

System error: the Keyboard Monitor command is unclassifiable (not '.R', '.RU', or '.GE').

'***P8RTS ERR G'

System error: the RTS device is unknown in this system.

'***P8RTS ERR H'

The Run-Time-System is not being called from SYS: or a device that is co-resident with SYS:.

'***P8RTS ERR I'

Invalid chain to P8RTS. Usually incompatible P8HEADer. See also ERR J.

'***P8RTS ERR J'

P8HEADer is incompatible with RTS -- reassemble P8HEAD, P8RTS, or both; or use the correct version of P8HEAD for P8RTS, and re-concatenate P8HEAD.SV and your code files (.PB). See section 3.2.3.

'***Too Many Errors'

Too many errors in succession have occurred, probably because of recursive or cyclic error processing. This will be followed by '***P8RTS ERR C'.

'BOOT' <HLT>

If a severe RTS error occurs and the normal exit paths from the RTS are impossible, this message requests the computer user to re-boot-strap OS/8 -- there is no other way to recover from this error. The only intentional HLT in P8RTS is executed after this message.

E.5. Run-Time System Error Messages Explained

Most error messages from the run-time-system will be of the form:

```
'*** Error: message'
```

or:

```
'*** Error: message'
'*** file   dev:name.ex'
```

and are (we hope) self-explanatory. If a file name is given as the second line of the error message, it is the OS/8 name of a file bound to a Pascal-OS/8 file variable that is involved in the error situation.

The error message is followed by a PMD (Post Mortem Display) to report the program state at the time of the error. If no PMD appears, a user program was not executing.

Here is presented a complete list of error messages, with explanations. The numbers on the left hand side are the error numbers and are normally not needed by the user. If there is no error number (---), the command statement processor is reporting the error.

The messages are ordered by (uppercased ASCII) alphabetical value. Error messages that begin with a name in parenthesis announce the name of the internal run-time-system routine that is signaling the error.

All errors that include the phrase "internal run-time error" indicate serious problems in the run-time system that the user program has stimulated. Internal errors are normally not the fault of the user program if the run-time tests (T+ compiler option) are on. But if the run-time tests are turned off (T-) then array references out of range, pointer references out of range, case statement jumps out of range and other possibilities can give uncontrollable results. Internal errors may also indicate a damaged .PB file (program code file).

Otherwise, logic or coding errors (bugs) in the run-time-system may show up as one of these internal errors. These errors can be reported in full to the supporters of Pascal OS/8, once you have determined that your program tests were not turned off.

051 (ABT) P8RTS system error

An internal run-time error has occurred while trying to abort a program run. (ABORT) Possibly incompatible with OS/8 BATCH.

005 Actual parameter is out of range

The parameter to a function or procedure is out of range of the defined type for the parameter.

001 Array index is out of range

The index being used to select an array element is outside of the index type of the array type; the result would reference an address out of the allocated memory space for an array, either calling for too low or too high an address.

006 Assigning FOR statement index out of range value

The index variable of a FOR statement has a defined range of values that cannot allow either the initial value or the final value to be assigned in the FOR loop statement.

142 Attempt to execute an un-defined FPP simulator instruction

An internal run-time system error has occurred. Not all possible FPP instructions are defined.

141 Attempt to execute an un-implemented FPP simulator instruction

An internal run-time system error has occurred. Not all possible FPP instructions are implemented in the FPP simulator.

264 Attempt to extend a Reset indexed file

An indexed (random-access) file that has been bound to the user program with a reset statement is a pre-existing file with a fixed size; elements within the length of the file may be read and written, but an attempt to write an element beyond the size of the file cannot be supported in OS/8.

265 Attempt to extend an indexed file by more than one component

An indexed (random-access) file is defined as extendable by one element at a time in a sequential fashion in this implementation. The reason for this is to avoid accidental creation of undefined elements if a new element were to be written more than one component index higher than the previous highest component index. It is possible to use the non-standard procedure CLOSE to reserve a fixed amount of disk space.

270 Attempt to get an indexed file with index too large

An attempt to read an element of an indexed file with an index that is larger than the maximum element on the file has caused an error. The non-standard function LENGTH can be used to determine the current size in elements of the indexed file. The non-standard function MAXLENGTH can be used to determine the maximum size in elements of the indexed file.

373 Attempt to open an input file on a write-only device

Some OS/8 devices are write-only; reading from these devices does not make sense. A paper tape punch is write-only.

324 Attempt to open read-only file for write

A file can be specified as read-only through the command statement or through reset with the form: 'filename[r]'. If a rewrite is executed upon such a file, the read-only lock is violated and this error results.

366 Attempt to open-write file on a read-only device

Some OS/8 devices (and internal) are read-only; writes to these devices do not make sense. A paper tape reader is read-only.

341 Attempt to read number past End-of-File

The user program has tried to read an integer or real but there is no more data available on the file named.

361 Attempt to read past End-of-File

The user program has tried to read data but there is no more data available on the file named.

266 Attempt to write on a read-only indexed file

A file can be specified as read-only through the command statement or through reset with the form: 'filename[r]'. If a rewrite is executed upon such a file, the read-only lock is violated and this error results. Also, the sequence:

```
reset(f,'f[r]') {f is indexed};
put(f,index)
```

will cause this error.

132 Attempted divide by zero detected by FPP
144 Attempted divide by zero detected by FPP simulator

The user program is attempting to divide by zero.

--- Bad delimiter

The command statement contained a delimiter character that did not match the syntax of command statements.

--- Batch command, Error: in

the '%' prefix was missing from the BATCH command line that the RTS was trying to read in job-step mode.

053 (BATPR) P8RTS system error

An internal run-time error has occurred while setting up the run-time system to run under OS/8 BATCH. (BatchPreset) Possibly incompatible with OS/8 BATCH.

311 (BEFRT) Invalid file-specification

The user program has passed a character string which is not a valid file-specification to the run-time system. This can usually happen while using the reset or rewrite procedures with the non-standard second string parameter. (BindExternalFileRunTime)

276 Cannot open a file for PMD

The Run-Time-System is processing an error and needs to open a file to write out the Post-Mortem-Display information, but there is no room on the assigned OS/8 device (SYS: normally).

003 Case index has no matching statement part

The value of the case-selector expression does not match any of the case labels in the case statement. Perhaps a case was omitted, or the Otherwise extension may be desired.

101 (CIP) Intrinsic procedure number is out of range

An internal run-time error has occurred while attempting to call an intrinsic (predefined or-built in) procedure. It is possible this error could occur with a damaged Pascal Binary (.PB) file. (CallIntrinsicProcedure)

343 (CLOSE) FITFSF open mode is bad

An internal run-time error has occurred while closing a file; an invalid open mode is set in the File-Information-Table. (CloseFile)

317 Close index length larger than actual file

The non-standard procedure Close is being used to close a file with an explicit length (second parameter to Close) but the length given overflows the space available on the mass storage device for the file.

--- Code file length is wrong

The code file is in a funny state, the internal length does not match the directory length; usually a damaged code file. Recompile the program and re-run it.

--- Command statement too long

The command statement length has exceeded the length limit and boy are we surprised!

--- Control-character in command

Control characters are not used in command statements.

105 (CPR) Invalid comparison type

An internal run-time error has occurred while comparing two items in the interpreter. (ComPare)

104 (CPR) Undefined relational code

An internal run-time error has occurred while comparing two items in the interpreter. (ComPare)

043 CTRL-A abort

The user has pressed Control-A upon the console in order to abort the execution of the program with no traceback, but closing any open files.

044 CTRL-D abort/PMD

The user has pressed Control-D upon the console in order to abort the execution of the program and produce a Post-Mortem-Display.

321 Device already has an open-write file (OS/8 restriction)

OS/8 allows only one temporary file at a time on any one device (one open-write file, or one extend mode file). A second Rewrite has been attempted on the same OS/8 device. Re-run the program with one of the files assigned to another device.

312 Device is not readable

The device being accessed is not ready, or returns a fatal error when the file is being looked up. This usually occurs when a program tries to read a file on a disk that is not ready to read (not set to "run").

ready to read (not set to "run").

313 Device is not writeable

The device being accessed in order to write a new file is ready but is write-locked. The device should be made write-ready.

372 Device not available in this system

The device named does not exist within the Pascal-OS/8 Run-Time-System or within OS/8.

057 Device unit number out of range

The first argument to DevGet, DevPut or DevReady is a unit that does not exist.

340 Digit expected for numeric read

A non-digit has been encountered on the text file named while looking for a digit to read an integer or real. Either a digit or a sign character ('+', '-') was expected.

124 Execute prefix character not recognized

The non-standard Execute procedure takes a string which must start with '.' or with '%' (the OS/8 Keyboard Monitor signal or the Pascal-OS/8 prompt character). Any other first character in the string will result in this error. (The '.' causes the rest of the string to be issued to CCL, and the '%' causes the rest of the string to be given to the RTS for another program execution.

123 Execute string is too long

The string parameter to the non-standard execute procedure is longer than 36 characters, a length which is determined by the CCL command buffer.

047 (EXERT) Execute is not enabled

The non-standard procedure EXECUTE(string) has been called but the run-time system is not enabled to perform this function. See installation notes for the run-time system. (ExecuteRunTime)

137 EXP (exponential) operand is too large
140 EXP (exponential) operand is too small

134 Exponent overflow detected by FPP
135 Exponent underflow detected by FPP

The Floating Point Processor has detected an exponent going out of the designed range of real numbers. The minimum exponent is -617 and the maximum is +616. The smallest non-zero number that the FPP represents is 3.09431E-617, and the largest is 1.61587E+616.

277 Failure while writing PMD file

This error occurs if an device error happens while writing out or closing the PMD (Post-Mortem-Display) dump file. The PMD for the current program is lost. The device may be write-locked, not ready, full, or too fragmented.

122 Fatal device read error during segment load

While loading a program segment for the currently executing Pascal-OS/8 program, a fatal device Input error has occurred. This might happen if the device were taken off line during the execution of a program, or if there were an actual read failure.

--- Fatal error loading program

A serious device read error has occurred while loading (the initial segment of) the users program. Clear up the device problem and try again.

262 Fatal handler error on indexed file

The device handler assigned to an indexed file has signaled a fatal (non-recoverable) Input or Output error while trying to perform Input or Output.

375 Fatal USR file close error

The OS/8 User Service Routine has signaled a fatal error while closing a file, perhaps because the device involved encountered an Input/Output error, or because the directory is not in the correct condition to close the file. This is an internal run-time error.

064 Field address (CDF) out of range

An internal run-time system error. Every address is stored as a 17-bit integer and the upper 5 bits are translated to a CDF instruction before access. If the high order word of an address is would translate to a non-existent field, this error results.

306 Field width is negative or zero

The field width expression for an item in a write (or writeln) list has resulted in a negative or zero field width. Field widths must be positive, non-zero integers.

007 File index is out of declared range

The second parameter to get or put for a non-standard indexed file has a value which is outside of the declared range for the indexed file.

066 (FIOBIO) Attempt to read/write block 0..6

An internal run-time error threatened the OS/8 device directory during an Input/Output operation.
(FileInputOutputBufferInputOutput)

374 (FIOOPR/FIOOPW) Open of already open file

An internal run-time error has occurred while opening a file: attempt to open a file that is already open.
(FileInputOutputOpenRead/FileInputOutputOpenWrite)

364 (FIOOPW) USR ENTER function error

An internal run-time error has occurred while opening a new file for write. (FileInputOutputOpenWrite)

325 (FIORDB) No instream is established

An internal run-time error has occurred while reading a buffer of data but the file is not opened properly.
(FileInputOutputReadBuffer)

360 (FIORDB) Sequential input invalid from indexed file

An internal run-time error has occurred while reading a buffer. An indexed file cannot be used as a sequential file.
(FileInputOutputReadBuffer)

350 (FIORDC) FITYPE invalid

An internal run-time error has occurred while reading a character - the file type was not a character type.
(FileInputOutputReadCharacter)

056 (FIOSET) File sub-system setup error

An internal run-time error has occurred while initializing the File Input-Output sub-system. (FileInputOutputSetup)

326 (FIOWTB) No outstream is established

An internal run-time error has occurred when attempting to write a buffer with no output stream defined. (FileInputOutputWriteBuffer)

355 (FIOWTB) Sequential write invalid to indexed file

An internal run-time error has occurred when attempting to write sequentially to a file defined as indexed. (FileInputOutputWriteBuffer)

345 (FIOWTC) FITYPE invalid

An internal run-time error has occurred while writing a character to a file that is not a character type file. (FileInputOutputWriteCharacter)

261 (FIOXIO) Handler signaled End-of-File on indexed file

An internal run-time error has occurred on an indexed file. Assignment of an indexed file to a non-file-structured device is not allowed. (FileInputOutputIndexedIO)

274 (FIOXIO) No outstream defined for indexed file

An internal run-time error has occurred when attempting to write on an indexed file that has no data stream assigned.

273 (FIOXIO) Not an indexed file, can't do indexed I/O

An internal run-time error has occurred when trying to do indexed (random) Input or Output (read or write) on a non-indexed file type. (FileInputOutputIndexedIO)

126 Floating Point Processor won't start

The Run-Time system thought there was an Floating Point Processor on the PDP8 bus, but it could not start it up. See the section on configuring and patching the RTS for a way to shut off FPP usage (and an internal FPP simulator will then be used if the program uses real numbers).

131 FPHLT instruction caused an FPP exit

This is an internal run-time error. FPHLT is FPP Halt.

143 (FPPSIM) Impossible FPP field

An internal run-time error has occurred in the Floating Point Simulator. A data field out of the range of 0..7 is not possible for the FPP. (FloatingPointProcessorSimulator)

133 Fraction overflow in DP mode detected by FPP

This Pascal-OS/8 Run-Time System does not use DP mode, so if this error occurs, welcome to the twilight zone. Please report.

307 Fraction width is negative or zero

The fraction width expression for a real number item in a write (or writeln) list has resulted in a negative or zero fraction width. Fraction widths must be positive, non-zero integers.

002 Function result is undefined

No statement has assigned a result to the function identifier.

```
1 program m;
2   var b:integer;
3   function f:integer;
4   begin
5     if false then f := 0
6   end;
7 begin
8   b := f
9 end.
```

This error will happen while executing line 6 (return from function) because f never was assigned a value.

102 (GBA) Bad static level

An internal run-time error has occurred in the Pascal P-code interpreter. (GetBaseAddress)

046 Halt.
046 Halt(message)

The user program has executed the non-standard Halt procedure to terminate execution. If PMD was enabled (default P+) in the user program, a PMD report will follow. If running under OS/8 Batch, the current Batch job will be aborted. If a message is given with the Halt, it is from the user program.

344 Impossible I/O field

An internal run-time system error. Input/Output cannot be performed outside of fields 0..7.

--- Incompatible load file

The file specified to the run-time-system to be executed has been compiled with a different version of Pascal-OS/8. Recompile with the current version, or run on the old version of the run-time-system.

354 Incomplete last record on sequential file

If a non-text, non-indexed file has elements that take more than one OS/8 mass storage block (256 words) to store, and if the last element on the file is missing one or more blocks of the element, the file is mal-formed and this error will result. Possibly the wrong file was bound to the program, or the file was created incorrectly by an aborted program execution or by a non-Pascal program.

275 Index too large or negative

The second parameter to get or put for an indexed file is either too large for the declared range of the file, or is negative.

263 Indexed extend would overflow space available for file

While extending an open-extend (newly created) indexed file, the current extend request using put(f,index) would overflow the space available on the mass storage device for the file. There is no more room in this file hole.

370 Indexed file must be on a file-structured device

An indexed (direct-access) file must be allocated on an OS/8 file-structured device because these are the only devices that have the consistent property of non-sequential access to storage blocks. In particular, the compiler builds the program code file on a file-structured device, and gets the default device from the device of the source file specified. See section 2.2.2.

272 Indexed file not open

An Input or Output operation has been attempted on an indexed file which has not been opened for read/write or extend. This is an Input/Output sequence error: the file must be opened before I/O can be performed.

336 Input/Output sequence error or file not opened

This error indicates one of three possible conditions:

1. the file has not been reset or rewritten (has not been opened for read or extend) and an Input/Output operation (read, readln, write, writeln, eof test, eoln test, get, put, or reference to the file buffer, page, etc.) has been attempted, or:

2. the file is opened for read (reset) and a write operation (write, writeln, put, etc.) is attempted), or:

3. the file is opened for write (rewrite) and a read operation (read, readln, get, etc.) is attempted.

This error check is performed for every Input/Output operation to ensure the integrity of files and the file system, so the error possibilities are combined into one check for more efficiency.

362 Instream Handler Fatal Error

The Instream is the current input stream of data; an OS/8 handler has signaled a fatal (non-recoverable) error while reading from a file.

267 Insufficient data on indexed file for get

If a non-text indexed file has elements that take more than one OS/8 mass storage block (256 words) to store, and if the last element on the file is missing one or more blocks of the element, the file is mal-formed and this error will result. Possibly the wrong file was bound to the program, or the file was created incorrectly by an aborted program execution or by a non-Pascal program.

337 Integer too large

An integer has been read with an absolute value larger than MAXINT (8388607, or 2 ** 23 - 1).

353 Internal handlers cannot be allocated for non-character files

The device handlers that are provided internal to P8RTS such as TT:, LP:, PT: are all character-only (text, file of char, file of ASCII, file of EightBit) handlers and may not be bound to non-character files such as file of integer, or indexed files.

314 Invalid file name

A string that fails to match the syntax for an OS/8 filename has been given as the second parameter to reset or rewrite, or has resulted from the program statement and command statement binding process. Often, this is a blank file name. In particular, the compiler has blank ('dsk:.ps') for a default

source file name and if the user gives no file name, this error results.

271 Invalid index for file is zero

An indexed file always has bounds of 1 and a non-negative, non-zero integer less than 4095. The second parameter to get or put for a non-standard indexed file was zero.

110 Invalid operand of DIV or MOD

The second operand of DIV (the divisor) must not be zero. The second operand of MOD (modulus, remainder) must be positive and non-zero.

127 Invalid pointer passed to Release

An invalid pointer would be a pointer with a value of nil or a value that lies outside of the allocated range for the program.

342 (IOERR) Unexpected error path

An internal run-time error has occurred in the error trace-back mechanism. (InputOutputError)

170 KLUDGE procedure number is too big

The maximum KLUDGE procedure number is set by assembly constants within the run-time system. This is an experiment. See section 3.2.5.

063 (LBM) Batch Monitor Load Error

An internal run-time error has occurred when attempting to load the OS/8 Batch Monitor to abort Batch. (LoadBatchMonitor)

052 (LBM) P8RTS system error

An internal run-time error has occurred when attempting to load the OS/8 Batch Monitor. (LoadBatchMonitor)

113 (LDHAND) Zero IOT code for TEK: or RIO:

An internal run-time error has occurred when the TEK: or the RIO: handler is referenced by a program but the device code patched in for TEK: or RIO: is zero (0000). See section 5.4.2 for setting the IOT codes for TEK: or RIO:. (LoadHandlers)

117 LN (natural LOG) operand is zero or negative

The argument (operand) to LN must be non-zero and non-negative. LN is defined over the range of positive non-zero real numbers.

--- Load device not available

The device given with the load file name does not exist in this OS/8 system. Use RESORC/E in OS/8.

--- Load file not found

The load file name specified in the command statement was not found on the device given (DSK default, .PB extension default).

071 (MEMPR) Not enough real memory for jam-set MEMLWA

An internal run-time error. MEMLWA is MEMory Last Word Address, and is normally set automatically. (MemoryPreset)

116 (MFTRT) Parameter error

An internal run-time error when a new file is being created. (MakeFitRunTime)

045 (NDH) Handler problem: bad #/ nonexistent/ too big

An internal run-time error while loading a handler. The handler called for does not exist or is too large to fit in the allocated space for handlers. (NeeDHandler)

125 No CCL found to Execute

SYS:CCL.SV was not found. A call to the non-standard procedure Execute with a string parameter starting with '.' will attempt to chain to CCL with the string parameter as a command to CCL.

nnn No error message assigned
nnn No error message found

The error being reported has no error message text to go with it. An error number will be reported, however.

--- No load file name

There was no load file name given on a command statement.

--- Not a recognizable load file

The load file given on the command statement exists but is not recognizable as a Pascal-OS/8 compiled program (.PB file). This could be a damaged code file, an accidentally renamed file, or an anciently compiled code file.

073 Not enough hardware memory for P8RTS (12k minimum)

The Pascal-OS/8 Run-Time System requires 12k minimum to execute, but can start in 8k, giving this message. The compiler takes 20k minimum, and can compile larger programs with more memory, up to the maximum allowed by P8RTS.

072 Not enough OS/8 memory for P8RTS (12k minimum)

The OS/8 CCL command .MEMORY has been used to set the amount of memory less than 12K. (.MEM 1 is too small. Set .MEM 2 or greater.)

346 Outstream file overflow
357 Outstream file overflow

The file being written would exceed the space available on the OS/8 file-structured device allocated by the rewrite (OS/8 enter) procedure. Squish the device or bind the file to a more spacious device, or reduce the amount of output being generated.

356 Outstream handler fatal error

The handler got a write error while trying to write out data. Check for write-lock, try running the program again. This message could also be given if the handler were trying to write outside of the bounds of the device.

111 Overflow during integer multiply

The result of a multiply of two integers became too large to express as a double-precision (23 bit signed) number. Perhaps the computation requires real numbers, or perhaps it simply exceeds the limitations of this implementation of Pascal.

376 Overflow while closing file

When closing a file at the end of a procedure, function or program block, or as a result of the non-standard Close statement, the remaining data were being written out from buffer to mass-storage, but the buffer of data would not fit into the remaining hole on disk. A larger space for the file is needed, then re-run the program.

060 (P8RTS) Jump to address 00000
061 (P8RTS) Jump to address 10000
062 (P8RTS) Jump to address 20000

An internal run-time error has occurred with PDP8 execution going out of control and eventually resulting in a jump (or jms) to address 0000 in one of the memory fields where the run-time-system has code. (Pascal0S/8RunTimeSystem) Possibly the result of the interrupt system being enabled.

065 (PMD) PMD command statement is too long

An internal run-time error has occurred while processing a prior run-time error. P8RTS constructs a new command statement in order to invoke PMD to analyze an error. (PostMortemDisplay)

171 Program Aborted

The program has called the non-standard KLUDGE 2 procedure to abort itself (and BATCH, if running) without a PMD report.

076 (PRS) EAE required but not available

An internal run-time error has occurred during initialization of a version of the run-time-system that was assembled with code depending on the presence of an EAE (Extended Arithmetic Element). (PreSet)

077 (PRS) FPP required but not available

An internal run-time error has occurred during initialization of a version of the run-time-system that was assembled with code depending on the presence of a FPP (Floating Point Processor). (PreSet)

041 (PRS) Preset called twice

An internal run-time error. PreSet performs once-only initializations. (PreSet)

070 (PSP) Attempt to increase SP

An internal run-time error. SP is the stack pointer. This is an attempt to increase it beyond the currently allocated stack space. (PutTopinStackPointer)

106 Ran out of memory during program or segment load

This can occur either at the initial start-up of any program, in which case there is not enough memory to load the program,

or anytime during program execution of a segmented code file such as the compiler, which indicates there is not enough memory to run the program to completion. See section 3.1.3 for a discussion of segmentation.

347 Read failure during program load

While reading the first program segment for the currently executing Pascal-OS/8 program, a fatal device Input error has occurred. This might happen if the device were taken off line during the execution of a program, or if there were an actual read failure.

050 Real numbers are not available (no FPP or FPP simulator)

The version of Pascal-OS/8 Run-Time System that you are using does not support real numbers in the absence of a hardware FPP, and the program uses reals. The RTS has been configured with no FPP simulator if this message occurs.

310 Real too large or small

A real number being read has gone out of the range represented by the FPP. The smallest non-zero number that the FPP represents is 3.09431E-617, and the largest is 1.61587E+616. The read real routine within the run-time system does not achieve quite this large range in order to try to detect going out of range. Read real accepts: -1e+614 ... -1e-614, 0, 1e-614 ... 1e+614.

112 Real too large to trunc or round

The value of a real number after trunc or round must not be larger than MAXINT, which is 8388607 in this implementation.

100 (RNI) Undefined P-machine operation

An internal run-time error. The Pascal Code file has called for an undefined P-machine operation. (ReadNextInstruction)

040 Second operand of MOD is negative

The MOD (modulus) operator is not defined for negative modulus in Pascal. See also error # 110 "Invalid operand of DIV or MOD".

107 Set element is out of range

An attempt was made to create a set element that is outside the ordinal range 0..95 (except for characters, which allow the range chr(32) .. chr(127) (or ' ' .. '~').

055 (SETRAP) P8RTS system error

An internal run-time error during PreSet. Traps are set for control-C from the console so the run-time system can restore the OS/8 environment properly. See section 5.3.3. (SetTraps)

120 SIN or COS operand magnitude is too large

The SIN and COS functions require the absolute value of their operand (argument or parameter) to be less than or equal to 9099.0242, which is equal to (pi * 2 ** (23/2)). See section R.3.1 for a reference.

114 SQRT (Square Root) operand is negative

The SQRT function is not defined for negative operands.

121 String argument is too long

In general, no string argument to a predeclared procedure or function can be over 4095 characters long.

042 (SYSPR) Cannot relocate SYS: handler above field 7

An internal run-time error. If the PDP8 has more than 32k memory, and the SYS: handler is two pages long, the second page cannot be successfully relocated. (SystemPreset)

054 (SYSPR) Logic error

An internal run-time error. If the SYS: handler is two pages long, the second page has to be relocated to the high field to free up field 2. Some logic error is preventing this operation. Possibly, P8RTS does not recognize the two-page System handler on the host system. (SystemPreset)

316 Too many digits in real number

A real number is being read that has an integer part with so many digits that it will produce a number larger than can represented within the FPP (over 616 digits!).

--- Too many files named or implied

The number of files named or implied (,,) on the command statement exceeds the number of file parameters defined in the program statement of the program to be executed. See T.4, PasCode, for some help.

115 Too many local file names generated

When a local file variable is created by a Pascal-OS/8 program, it is assigned a name from the sequence PASCAL.TM, PASCAM.TM, PASCAN.TM, ..., ZZZZZZ.TM. If a program actually creates a very large number of local files, the name cannot be incremented past the last value, and this error results.

130 Trapped instruction caused an FPP exit

An internal run-time system error has occurred. Please report.

136 Unknown FPP error

The FPP has signaled an error condition but does not admit to any specific error bits! An internal error, probably hardware.

103 Use of an undefined pointer

An attempt was made to use a pointer which has an undefined value. Not all undefined values can be caught, however. Pointer values of 00000 to 17777 are always disallowed if tests are turned on (compiler option T+).

074 Use of Nil pointer

An attempt was made to use a pointer which has the value nil to reference a data item.

004 Value being assigned is out of range

An attempt to assign a value that is outside the declared range of a variable will result, in most cases, in this error message, if tests are turned on (compiler option T+).

E.6. Run-Time-System Error Messages, Numerical Order

001 Array index is out of range
002 Function result is undefined
003 Case index has no matching statement part
004 Value being assigned is out of range
005 Actual parameter is out of range
006 Assigning FOR statement index out of range value
007 File index is out of declared range
040 Second operand of MOD is negative
041 (PRS) Preset called twice
042 (SYSPR) Cannot relocate SYS: handler above field 7
043 CTRL-A abort
044 CTRL-D abort/PMD
045 (NDH) Handler problem: bad #/ nonexistent/ too big
046 Halt.
047 (EXERT) Execute is not enabled
050 Real numbers are not available (no FPP or FPP simulator)
051 (ABT) P8RTS system error
052 (LBM) P8RTS system error
053 (BATPR) P8RTS system error
054 (SYSPR) Logic error
055 (SETRAP) P8RTS system error
056 (FIOSET) File sub-system setup error
057 Device unit number out of range
060 (P8RTS) jump to address 00000
061 (P8RTS) jump to address 10000
062 (P8RTS) jump to address 20000
063 (LBM) Batch Monitor Load Error
064 Field address (CDF) out of range
065 (PMD) PMD command statement is too long
066 (FIOBIO) Attempt to read/write block 0..6
070 (PSP) attempt to increase SP
071 (MEMPR) Not enough real memory for jam-set MEMLWA
072 Not enough OS/8 memory for P8RTS (12k minimum)
073 Not enough hardware memory for P8RTS (12k minimum)
074 Use of Nil pointer
075 (MPR) KT8A required but not available
076 (PRS) EAE required but not available
077 (PRS) FPP required but not available
100 (RNI) Undefined P-machine operation
101 (CIP) Intrinsic procedure number is out of range
102 (GBA) Bad static level
103 Use of an undefined pointer
104 (CPR) Undefined relational code
105 (CPR) Invalid comparison type
106 Ran out of memory during program or segment load
107 Set element is out of range
110 Invalid operand of DIV or MOD
111 Overflow during integer multiply
112 Real too large to trunc or round
113 (LDHAND) Zero IOT code for TEK: or RIO:
114 SQRT (Square Root) operand is negative
115 Too many local file names generated
116 (MFTRT) Parameter error
117 LN (natural LOG) operand is zero or negative
120 SIN or COS operand magnitude is too large

121 String argument is too long
122 Fatal device read error during segment load
123 Execute string is too long
124 Execute prefix character not recognized
125 No CCL found to Execute
126 Floating Point Processor won't start
127 Invalid pointer passed to Release
130 Trapped instruction caused an FPP exit
131 FPHLT instruction caused an FPP exit
132 Attempted divide by zero detected by FPP
133 Fraction overflow in DP mode detected by FPP
134 Exponent overflow detected by FPP
135 Exponent underflow detected by FPP
136 Unknown FPP error
137 EXP (exponential) operand is too large
140 EXP (exponential) operand is too small
141 Attempt to execute an un-implemented FPP simulator instruction
142 Attempt to execute an un-defined FPP simulator instruction
143 (FPPSIM) Impossible FPP field
144 Attempted divide by zero detected by FPP simulator
170 KLUDGE procedure number is too big
171 Program Aborted
261 (FIOXIO) Handler signaled End-of-File on indexed file
262 Fatal handler error on indexed file
263 Indexed extend would overflow space available for file
264 Attempt to extend a Reset indexed file
265 Attempt to extend an indexed file by more than one component
266 Attempt to write on a read-only indexed file
267 Insufficient data on indexed file for get
270 Attempt to get an indexed file with index too large
271 Invalid index for file is zero
272 Indexed file not open
273 (FIOXIO) Not an indexed file, can't do indexed I/O
274 (FIOXIO) No outstream defined for indexed file
275 Index too large or negative
276 Cannot open a file for PMD
277 Failure while writing PMD file
306 Field width is negative or zero
307 Fraction width is negative or zero
310 Real too large or small
311 (BEFRT) Invalid file-specification
312 Device is not readable
313 Device is not writeable
314 Invalid file name
316 Too many digits in real number
317 Close index length larger than actual file
321 Device already has an open-write file (OS/8 restriction)
324 Attempt to open read-only file for write
325 (FIORDB) No instream is established
326 (FIOWTB) No outstream is established
335 Error closing file
336 Input/Output sequence error or file not opened
337 Integer too large
340 Digit expected for numeric read
341 Attempt to read number past End-of-File
342 (IOERR) Unexpected error path
343 (CLOSE) FITFSF open mode is bad

344 Impossible I/O field"
345 (FIOWTC) FITYPE invalid
346 Outstream file overflow
347 Read failure during program load"
350 (FIORDC) FITYPE invalid
353 Internal handlers cannot be allocated for non-character files
354 Incomplete last record on sequential file
355 (FIOWTB) Sequential write invalid to indexed file
356 Outstream handler fatal error
357 Outstream file overflow
360 (FIORDB) Sequential input invalid from indexed file
361 Attempt to read past End-of-File
362 Instream Handler Fatal Error
364 (FIOOPW) USR ENTER function error
366 Attempt to open-write file on a read-only device
370 Indexed file must be on a file-structured device
372 Device not available in this system
373 Attempt to open an input file on a write-only device
374 (FIOOPR/FIOOPW) open of already open file
375 Fatal USR file close error
376 Overflow while closing file

F. Pascal Compiler Messages: Version 1-0-F B 1982-07-05.

F.1. General Compiler Messages

*** UNDECLARED PROCEDURE: name
A procedure heading followed by the compiler directive "forward" was not subsequently defined as a procedure block.

**** INCOMPLETE PROGRAM.
The program "end." symbol was not seen when expected.

ERRORS DETECTED.
The compiler is unhappy.

Invalid options!
The option-sequence on the command statement had malformed options.

NO ERRORS DETECTED.
The compiler is happy.

NOT ENOUGH ROOM FOR CODE FILE
The device where the code file is being produced does not have enough contiguous space to contain it. Squish the device, or re-allocate the code file (section 2.2.2.3).

P8COM V1-0-F F
The compiler always prints its version message. The 1-0-F must match the version message of the run-time system for compatibility.

SOURCE FILE = OBJECT FILE!
The source file name is identical to the code file name, and the compiler refuses to over-write the source file.

SOURCE FILE (filename) IS EMPTY.
The source file does not exist, or has an immediate end-of-file.

SOURCE FILE NAME = LIST FILE NAME!
The source file name is identical to the listing file name, and the compiler refuses to over-write the source file.

UNDECLARED EXTERNAL FILE: filename
The external file 'filename' was first named in the program statement, but was never declared in the global var section.

UNDEFINED TYPE ID: name
A pointer declaration to a as-yet-unnamed type was not followed by a declaration of that type.

UNDEFINED LABEL: nnnn
A label has been declared but never defined within the proper scope.

F. Pascal Compiler Messages: Version 1-0-F B 1982-07-05.

F.2. Numbered Messages

```
  1:  Error in simple type.
  2:  Identifier expected.
  3:  "Program" expected.
  4:  ")" Expected.
  5:  ":" Expected.
  6:  Unexpected symbol.
  7:  Error in parameter list.
  8:  "Of" expected.
  9:  "(" Expected.
 10:  Error in type.
 11:  "[" Expected.
 12:  "]" Expected.
 13:  "End" expected.
 14:  ";" Expected.
 15:  Unsigned integer expected.
 16:  "=" Expected.
 17:  "Begin" expected.
 18:  Error in declaration part.
 19:  Error in field-list.
 20:  "," Expected.
 21:  ".." Expected.

 50:  Error in constant.
 51:  ":=" Expected.
 52:  "Then" expected.
 53:  "Until" expected.
 54:  "Do" expected.
 55:  "To" or "downto" expected.
 58:  Error in factor.
 59:  Error in variable.

100:  Use of extension makes program nonstandard.
101:  Identifier declared twice.
102:  Low bound exceeds highbound.
103:  Identifier is not of appropriate class.
104:  Identifier not declared.
105:  Sign not allowed.
106:  Number expected.
107:  Incompatible subrange types.
108:  File not allowed here.
110:  Tagtype must be ordinal.
111:  Case constant is of wrong type or is out of range.
113:  Index type must be ordinal.
115:  Base type must be ordinal.
116:  Error in type of predeclared procedure parameter.
117:  Undefined type, procedure, or function identifier.
119:  Forward declared; repetition of parameter list not allowed.
120:  Function result type must be ordinal, real, or pointer.
121:  File value parameter not allowed.
122:  Forward declared; repetition of result type not allowed.
123:  Missing result type in function declaration.
124:  Expression to be written must be real.
```

125: Error in type of predeclared function parameter.
126: Number of parameters does not agree with declaration.
127: Invalid substitution, incompatible procedures or functions.
129: Type conflict of operands.
130: Expression is not of set type.
131: Only "=" and "<>" allowed with pointers.
132: Neither "<" nor ">" allowed with sets.
134: Invalid type of operand(s).
135: Type of operand must be Boolean.
136: Set element type must be ordinal.
137: Set element types not compatible.
138: Type of variable is not array.
139: Index type is not compatible with declaration.
140: Type of variable is not record.
141: Type of variable must be file or pointer.
142: Invalid parameter substitution.
143: Type of variable must be ordinal.
144: Invalid type of expression.
145: Type conflict.
146: Assignment of files not allowed.
147: Type of constant is incompatible with case-index expression.
148: Subrange bounds must be ordinal.
150: Assignment to this function is not allowed here.
152: No such field in this record.
154: Actual parameter must be a variable.
155: Control variable must be local.
156: Multidefined case label.
157: Implementation limit: range of case-constants is too large.
158: No such variant in record declaration.
160: Previous declaration was not forward.
161: Previously declared forward.
163: Unspecified tag type value(s).
165: Multidefined label.
166: Multideclared label.
167: Undeclared label.
168: Undefined label.
169: Set base-type exceeds implementation limits.
171: Attempting to redeclare standard file ("input" or "output").
172: Undeclared external file.
175: Missing file "input" in program heading.
176: Missing file "output" in program heading.
178: Value was previously specified.
180: Control variable must not be formal.
181: Implementation limit: array too large.
182: Implementation limit: record too large.
183: Implementation limit: too many large local variables, parameters, or expressions.
184: Invalid file index range.
185: Indexed character file not allowed.
187: Invalid directive or mis-spelled reserved word.
188: Label is not accessible from here.
189: Label is not accessible to previous goto(s) that used it.
190: Identifier being defined was used already in this scope.
191: Type identifier expected.
192: Invalid use of for-statement control variable.
193: Invalid use of tag field.
194: Control variable is threatened by nested procedures or

functions.
195: Already a control variable for an enclosing for-statement.
196: Function result is nowhere defined.

200: Character is not valid in Pascal.
201: Error in real constant; digit expected.
202: String constant must not exceed source line.
203: Implementation limit: integer constant exceeds maxint.
204: 8 Or 9 in octal number.
205: Null string not allowed.
206: Implementation limit: real constant exceeds range.
207: Invalid character code.
208: Error in compiler option.
209: Error in file-binding specification.
210: Value of label is too large.
211: Unexpected comment terminator.

250: Implementation limit: identifier scopes nested too deeply.
251: Implementation limit: procedures and functions nested too deeply.
252: Implementation limit: too many identifier scopes.
253: Implementation limit: code segment too long.
255: Implementation limit: too many errors on this source line.
259: Implementation limit: expression too complicated.
260: Implementation limit: too many segments.
261: Implementation limit: too many procedures, functions, and destination labels for non-local gotos.
263: Implementation limit: too many program parameters.
264: Implementation limit: too many formal parameters.

300: Second operand of div or mod is zero.
302: Index expression out of declared range.
303: Value to be assigned is out of declared range.
304: Implementation limit: set-member expression out of range.
305: Parameter value is out of declared range.

350: Kludge procedure index is out of range.
351: Implementation limit: pointers to files not allowed.
352: External variables must be file variables.

400: Compiler error (please report).

Compiling a Pascal program.
If your Pascal program is a file called "YourProgram.PS" then the command:

```
.R P,SYS:PASCAL,YourProgram
```

will run (execute) the Pascal compiler with your program as it's input. If there are no errors, the compiler will generate a "code file" called "YourProgram.PB" on the same device as your source which you can execute.

The compiler will also provide a listing if you specify a device and/or file after the program. For example:

```
.R P,SYS:PASCAL,YourProgram,LP:
```

will compile YourProgram.PS while listing it on the line printer. (LP: is an internal line printer handler, see below.)
To get a cross-referenced listing of your program, do:

```
.R P,SYS:IDMAP,YourProgram,LP:
```

which produces a cross-referenced listing on the line printer (LP:).

Executing a Pascal program
A "code file" called "YourProgram.PB" is executed by the command:

```
.R P,YourProgram
```

Device and/or file names can be "substituted" for program parameters by listing them after YourProgram:

```
.R P,YourProgram,param1,param2,...
```

To reproduce the last PMD (Post Mortem Display, execution trace-back) if you need a printed copy of the error trace-back report, do:

```
.R P,SYS:PMD,LP:
```

Internal Device Handlers (character oriented input/output). Not all are necessarily available (installation defined).

TT:	OS/8 console handler, input line buffered and edited.
TTO:	OS/8 console handler, immediate (no buffering or editing).
LP:	line printer handler, input line buffered and edited.
LPO:	line printer handler, immediate.
IO:	message output, equal to TT: unless BATCH is running on the line printer, then IO: is equal to LP:. Input, equal to TT: if not under BATCH, else equal to NULL:.
NULL:	null device (input: immediate eof, output: infinite sink).
CS:	control statement image (read-only).
PT:	paper tape (high speed, input: reader, output: punch).
RIO:	remote terminal device, input line buffered and edited.
RIOO:	remote terminal device, immediate.
TEK:	TEK 4015 scope, input line buffered and edited.
TEKO:	TEK 4015 scope, immediate.
GIN:	TEK 4015 scope, no echo (input for GIN mode only) input line buffered and edited.

Compiler Options control compiler behavior. Default settings are shown in parenthesis. These switch options appear in program comments as: {$L+,T-} or on the compiler call statement, as:

```
.R P,SYS:PASCAL,YourProgram/L+,T-
```

I+,I-	enable,disable interactive I/O for all text files	(I+)
N+,N-	enable,disable line number reporting within PMD	(N+)
L+,L-	enable,disable compilation listing (L- unless listing device/file given)	
P+,P-	enable,disable Post-Mortem-Dump information generation	(P+)
S+,S-	enable,disable segmentation of subsequent procedure or function blocks	(S-)
T+,T-	enable,disable compiler-generated run-time tests	(T+)
V+,V-	enable,disable variable reporting within PMD.	(V+)
X+,X-	allow,disallow use of extensions	(X-)
O+,O-	enable,disable options. If disabled, cannot be enabled	(O+)

The following numeric options can only appear in program comments:

Bn	set Input/Output buffer factor to n, where 1<= n <= 512 (n is number of blocks). (default is B1)
Wn	set workspace to n, where n is in words. (Default is all memory.) No effect in this release (V 1-0-F).

Terminal Controls (TT:, RIO:, LP:, TEK:) ('*' means TT: only)

Name	Display	Effect
Control-C	"^C"	abort program, stop BATCH, return to OS/8.
Control-D	"^D" *	abort program, close files, do PMD, abort BATCH.
Control-A	"^A" *	abort program, close files, abort BATCH.
Control-B	*	debug enable: interpreter state frames.
Control-E	*	debug disable: interp. and segment state frames.
Control-F	*	debug enable: segment loader state frames.
Control-S		X-OFF, hold output until X-ON. (DC3)
Control-Q		X-ON, resume output after X-OFF. (DC1)
Control-O		slough off (mute) output, any other key will resume.

The following are effective during line-buffered input.

BackSpace	delete one character, erase from screen (control-H).
Rubout or Delete	delete one character; if OS/8 scope bit not set, echoes as "\x\", the TTY convention; if OS/8 scope bit is set, same as BackSpace: erases from screen.
Control-U "^U"	delete current input line.
Line Feed	re-print current input line.
Tab	perform OS/8 tabulation; always expanded to blanks.
Carriage Return	terminate input line; see also section 3.2.2.
Control-Z "^Z"	cause eof(f) to become true (after end of current input line, if any).

I. Implementation-Defined Checklist

8388607	:	MaxInt (maximum integer value)
-8388607	:	least integer
E	:	Exponent char produced on write real is 'E' (upper case).
E or e	:	Exponent char accepted upon read of a real is 'E' or 'e' (upper or lower case).
1.61587E+616	:	greatest real
3.09431E-617	:	least non-zero real
1.19209E-007	:	Epsilon, largest number added to 1.0 which does not change the value of 1.0.
6 or 7	:	significant digits in reals.
13	:	default real floating-point field width ("sn.nnnnnesxxx").
8	:	default integer field width ("snnnnnnn").
4 True, 5 False	:	default boolean field width ("True","False").
ASCII	:	character set. Type char is all printing characters (' ' ... '~', or chr(32) ... chr(126). Type char includes no non-printable characters. Chr(0) is not of type char. Non-standard type ASCII is all characters in the range chr(1) ... chr(127). Non-standard type EightBit is all characters in the range chr(0) ... chr(255). See Appendix A for a table of ASCII character values.
48	:	ord('0') see Appendix A.
65	:	ord('A') see Appendix A.
97	:	ord('a') see Appendix A.
carriage-return form-feed	:	effect of procedure Page on a text file.
8	:	number of significant characters in an identifier.
equivalent	:	upper and lower case in identifiers
'_'	:	additional character allowed as extension in identifiers -- underline.
150	:	maximum length of source line and identifiers.
preserved	:	blanks at end of line are preserved.

96	:	maximum set cardinality.
allowed	:	"set of char" is supported.
@	:	is equivalent to ^ (pointer references).
(* *)	:	is equivalent to { } (comment markers).
(. .)	:	is equivalent to [ ] (array references).
none	:	unimplemented standard procedures.
KLUDGE	:	non-standard compiler directive (unfinished experiment).
otherwise	:	extension of reserved word list.
none	:	non-standard predefined file parameters.
none	:	non-standard predefined labels.
none	:	non-standard predefined constants.
ASCII, EightBit	:	non-standard predefined types.
none	:	non-standard predefined vars.

Non-standard predefined procedures:

Date, Mark, Release, Halt, GetFN,

Close, DevPut, Execute.

Non-standard predefined functions:

Length, MaxLength, ValidFN, PwrOf10,

DevGet, DevReady.

R. References.

R.1. Pascal: Standard, Textbooks, Design.

Cooper, Doug. Standard Pascal User Reference Manual. W. W. Norton & Co. New York, 1983.

Cooper, Doug and Michael Clancy. Oh! Pascal! W. W. Norton & Co. New York, 1982.

Findlay, W., and D. F. Watt. Pascal: An Introduction to Methodical Programming. London: Pittman, 1978.

Holt, Richard C., and J. N. P. Hume. Programming Standard Pascal. Reston, Va.: Reston Publishing Co., 1980.

Jensen, Kathleen, and Niklaus Wirth. Pascal User Manual and Report. New York: Springer-Verlag, 1974, 1978. This is a reference manual, not a good tutorial. (As of this writing, there is a third edition in preparation.)

Schneider, G. M., S. W. Weinhart, and D. M. Perlman. An Introduction to Programming and Problem Solving With Pascal. New York: John Wiley & Sons, second edition, 1982.

Specification for Computer Programming Language Pascal, BS6192: 1982, British Standards Institution. (This is the ISO standard. It is not a tutorial. There is also a French translation published by AFNOR - the French standards organization.)

Watt, David. Pocket Guide to Pascal. Addison-Wesley Co., 1982.

Welsh, Jim, John Elder. Introduction to Pascal. Prentice Hall Inc., Englewood Cliffs, N.J., 1979.

Wirth, Niklaus. Algorithms + Data Structures = Programs. Prentice Hall, Inc., Englewood Cliffs, N.J., 1976.

R.2. Hardware and OS/8.

OS/8 Handbook, Digital Equipment Corporation, 1974.

OS/8 Software Support Manual (Version 3), Digital Equipment Corporation, 1974.

PDP8/e 8/m & 8/f Small Computer Handbook, Digital Equipment Corporation, 1973.

PDP8/a Minicomputer Handbook, Digital Equipment Corporation, 1976.

FPP8-A Maintenance Manual, Digital Equipment Corporation, 1976.

R.3. Implementation Techniques.

W. Cody and W. Waite, Software Manual for the Elementary Functions, Prentice Hall, 1980.

James B. Saxe and Andy Hisgen, Lazy Evaluation of the File Buffer for Interactive I/O, Pascal News #13, December 1978, Pages 92-93.

T. Tools

The following programs are useful tools that accompany the Pascal-OS/8 distribution kit. The sources for many of them are included on the distribution media. Only a few programs are pre-compiled, in order to be able to include as many sources as possible. To compile a tool (or example), follow the normal compilation instructions (see section 2.2). To compile directly from the distribution floppy (or other media), do:

```
.R P,PASCAL,device:toolname,,DSK:toolname
```

and the code file will be produced on DSK:.

The following tools written in Pascal are described in this appendix:

T.1	Compare	- compare two text files.
T.2	CopyV	- copy OS/8 files with verify.
T.3	IdMap	- produce a map of identifiers used.
T.4	PasCode	- produce pseudo-assembly listing.
T.5	PFMap	- procedure and function usage map.
T.6	PMD	- post mortem display for Pascal-OS/8.
T.7	SvUtil	- save file utility (ODT/FUTIL).

T.1 Compare - Compare two text files and report their differences.

Compare is used to report the differences between two similar texts (FILEA and FILEB). Notable characteristics are:

- Compare is line oriented. The smallest unit of comparison is the text line (ignoring trailing blanks). The present implementation has a fixed maximum line length.

- Compare employs a simple backtracking search algorithm to isolate mismatches from their surrounding matches. This requires (heap) storage roughly proportional to the size of the largest mismatch, and time roughly proportional to the square of the size of the mismatch for each mismatch. For this reason it may not be feasible to use Compare on files with very long mismatches.

To use Compare, enter:

```
.R P,COMPAR,filea,fileb,listing
```

There are no options. The program heading for Compare is:

```
PROGRAM COMPARE(FILEA, FILEB, OUTPUT);
```

The default file names for the two files to be compared are FILEA and FILEB, respectively. There are no default extensions for these files. Device DSK: is assumed unless you specify a device with your file names. The output will be sent to the TT: console handler, or to LP: handler if running OS/8 BATCH on the line printer.

The source for Compare is included in the Pascal-OS/8 kit, and is in nearly standard Pascal (non-standard junk in the source is marked with $X+, $X- comments). Now I know that OS/8 has a fast COMPAR program, and I tend to use COMPAR.SV instead of Compare most of the time; but when the mis-matches are a little too big for COMPAR.SV, or I need less ambiguous output, and have a lot of time to spare, I use COMPAR.PB. Also, you can take the source to your favorite (other) computer with Pascal and no decent compare utility and get it going with a little effort. Good luck!

T.2 CopyV - copy OS/8 files with verify.

CopyV is a utility written in Pascal-OS/8 that can copy OS/8 files between directory devices like FOTP (except CopyV does not have any wild-card handling or any options at all!) CopyV performs a verify pass for each copy pass, ensuring the integrity of the copy operation. CopyV loses the old date, however, putting today's date on the result file.

To use CopyV, enter:

```
.R P,COPYV,datafilename,copyfilename
```

and messages will report progress of the copy operation. If the copy does not verify correctly, a message is printed and execution is halted.

If the copyfilename is omitted, no name is assumed, and a run-time execution error will result -- no copy is attempted.

If the copyfilename is a device name only, and it is a different device name that in the datafilename, the datafilename file-name part and extension-part are used for the copyfilename. That is,

```
.R P,COPYV,SYS:BATCH.SV,SCR:
```

will append 'BATCH.SV' to the 'SCR:' specification and perform the copy and verify.

The source for CopyV is included in the distribution kit. It is not entirely standard Pascal, but does work well enough to be useful.

T.3 IdMap - Produce a map of identifiers used in a Pascal program.

IdMap is a Pascal program cross-referencer written in Pascal. It is very useful to produce a listing of your program, with the identifier usage cross-referenced at the end of the listing.

To use IdMap, do:

```
.R P,IDMAP,source,listing
```

where source is your source program (the .PS extension is assumed), and listing is the listing device or file (the LP: line printer is default, and the .LS extension is default).

The IdMap program heading is:

```
Program IdMap(Source     : '.PS',
              Listing    : 'LP:LISTING.LS',
              Output     : 'TT:',
              UsageTable: 'TEMP.TM');
```

IdMap might write error messages to output on the OS/8 console. IdMap will use a scratch file to accumulate the references. This file will be on device 'DSK:' by default. If you wish to allocate this scratch file on another device, do:

```
.R P,IDMAP,source,listing,,scratchdevice:
```

and IdMap will use the file scratchdevice:TEMP.TM. The scratch file is always removed when IdMap finishes.

IdMap has a limited capacity of < 1499 identifiers; the number of references is limited only by the available disk space.

IdMap will handle a maximum source width of 150 characters. Identifiers may be of any length. Identifiers that are not unique in the first eight characters are noted. Identifiers that appear with inconsistent capitalization are also noted.

The source for IdMap is included in the distribution kit for Pascal-OS/8. If IdMap will not run on the amount of memory you have, edit the HashSize from 1499 to a smaller odd number, preferably a prime number. Few programs have over a few hundred identifiers. It is possible that a 32K system could support a larger HashSize.

T.4 PasCode - Produce pseudo-assembly listing from Pascal-OS/8 code files.

PasCode reads a Pascal-OS/8 code file and produces output that interprets the contents of the code file. The PasCode control statement is:

```
.R P,PASCOD,program,listing/options
```

The PasCode program heading is:

```
program PasCode(PB:'.PB', Listing:'IO:.LS');
```

The default code-file-name is '.PB'. A file name must be specified, but the .PB extension is default.

The default listing file name is 'IO:.LS'. This will produce a listing on the OS/8 console, or on the line printer if OS/8 BATCH is running on the line printer.

The most frequent use of PasCode for the general user of Pascal-OS/8 might be to find out the program file parameters of a Pascal-OS/8 code file that you are uncertain about, or for which you do not have documentation or source. To find out the files the program needs, do:

```
.R P,PASCOD,program,listing/H
```

which tells PasCode to produce a header report only. Another use of PasCode is to report the segment structure of a code file when building a code-segmented program, as an aid to understanding. See Figure T.4-1.) The S option is used:

```
.R P,PASCOD,program,listing/S
```

The last use of PasCode is to decode the entire program in the pseudo-assembly language that corresponds to the interpreter instruction set. This report will only be of interest to maintainers of Pascal-OS/8. Do:

```
.R P,PASCOD,program,listing
```

The header report is produced for each of the three possible uses of PasCode.

Figure T.4-1 Example Segmentation Pascode Report

```
PASCODE 1-0-F-2/2    1983-10-09
R P,PASCOD,SegmentationExample/S

PROGRAM NAME:      SEGMENTA
COMPILATION DATE: 1983-10-09
PFL COUNT:              12
SEGMENT COUNT:           7
MAX WORK SPACE:     377777
NO-PMD FLAG:       False
NO-REALS FLAG:     True
COMPILER VERSION: V1-0-F F
PROG PARAMETERS:         0



Segment Lengths in Blocks
SEG=  0  LEN=  2  FBA=  1
SEG=  1  LEN=  1  FBA=  3
SEG=  2  LEN=  1  FBA=  4
SEG=  3  LEN=  1  FBA=  5
SEG=  4  LEN=  1  FBA=  6
SEG=  5  LEN=  1  FBA=  7
SEG=  6  LEN=  1  FBA=  8

Procedures/Functions/Labels

PFL=  0  SEG=  1  LEV= 1  PROGRAM   SEGMENTA
PFL=  1  SEG=  1    LEV= 2  PROCEDURE A
PFL=  2  SEG=  2    LEV= 2  PROCEDURE B
PFL=  3  SEG=  2      LEV= 3  PROCEDURE B1
PFL=  4  SEG=  3      LEV= 3  PROCEDURE B2
PFL=  5  SEG=  2      LEV= 3  PROCEDURE B3
PFL=  6  SEG=  4    LEV= 2  PROCEDURE C
PFL=  7  SEG=  5      LEV= 3  PROCEDURE C1
PFL=  8  SEG=  6      LEV= 3  PROCEDURE C2
PFL=  9  SEG=  1    LEV= 2  PROCEDURE D
```

Notes on PasCode Output:

- The "/S" option to Pascode is necessary for this report. Without it, Pascode will decode all the code in the program, and not produce the segmentation report.
- There are three sections to this Pascode report: (1) heading, (2) Segment table, (3) proc/func segmentation report.
- The heading gives the Pascode name, Pascode version, and today's date; echoes the command statement; and gives several lines of general information about the program code file. If a program has

any files in the program heading, the last part of the header will name them and give the default file name pattern as specified in the program source. This is can be used if you forget the program parameters for a program.

- The segment table gives the length in disk-blocks, and the first disk-block address (FBA), of each segment. Segment zero is the header of the code file.

- The proc/func segmentation report describes the Procedure/Function/Label (PFL) number of each proc/func, the segment that the proc/func is in (SEG), the nesting level of the proc/func (LEV), the kind of the proc/func (PROGRAM, PROCEDURE, FUNCTION, or LABEL), and, if PMD was enabled, the name of the proc/func.

T.5. PfMap- Pascal Procedural Cross-Referencer by A.H.J. Sale, University of Tasmania.

PfMap reads Pascal source programs and produces two tables as output. These tables are procedural documentation and cross-references. One table documents all procedure and function headings in a format that illustrates lexical nesting. The other table gives the locations of heading, block, and body for each procedure and function, and what procedures and functions it immediately calls.

To use PfMap, enter:

```
.R P,PFMAP,source,listing
```

The program heading for PfMap is:

```
Program Referencer(input:'dsk:input.ps',output:'10.ls');
```

so the default input device is DSK:, extension is .PS, and the default output device is 10:. If an output file name is specified, the output device will be either DSK: or the device given with the filename; and the default output extension is .LS.

The source for PfMap is included in the distribution kit. It is one of the most formal (and portable) Pascal programs in the kit.

T.6 PMD - Post Mortem Display for Pascal-OS/8

PMD produces an analysis of the state of a Pascal program when it causes a run-time execution error. PMD processing is automatic, with the report presented on the OS/8 console, or upon the line printer, if running OS/8 BATCH on the line printer. The PMD program is written in Pascal using Pascal-OS/8 extensions.

The last PMD report may be re-produced very simply if the file SYS:PASCAL.PM has not been disturbed, and if the code file (.PB) that was executing when the error was reported is still on the same directory. Enter:

```
.R P,SYS:PMD
```

or enter the following for a printed copy:

```
.R P,SYS:PMD,LP:        --or--        .R P,SYS:PMD,LPT:
```

(LP: is a line printer handler internal to the run-time-system.)

The Pascal-OS/8 run-time-system does the initial error processing by signaling the error with a message, and creating the file:

```
device:PASCAL.PM
```

where device: is the same device from which the run-time-system was .RUN. The .PM file contains the state of the interpreter, the state of the PDP8 machine, the error number, and the program stack and heap (if any). The run-time-system then executes the PMD program to analyze the information provided. PMD.PB is expected to be on the same device from which the run-time-system was .RUN.

If you wish to save the program dump file, you may:

```
.COPY dev2:prog.PM<SYS:PASCAL.PM
```

Then, to re-produce the PMD report, PMD expects the two program heading files codefile and dumpfile to be specified, and further expects the name recorded in the dumpfile file to match the codefile file name (although it will try to analyze anyway).

The full PMD call statement is:

```
.R P,PMD,listing,codefile,dumpfile/options
```

The PMD program heading is:

```
program PMD (output: 'IO :PMD    .LS',
                PB:      '           .PB',
                PM:      'SYS:PASCAL.PM');
```

'IO:PMD.LS' is the default file name pattern for first parameter, the PMD output file. '.PB' is the default pattern for the second program

file, the executable program name. 'SYS:PASCAL.PM' is the default pattern for the third program file, the PMD dump file.

Options are any of the following.

/B - B+ asserts OS/8 BATCH is running, abort it after producing the PMD report. B- asserts BATCH is not running. This allows the RTS to tell PMD the state of BATCH. Default is B- (false).

/Wnnn - set Printer Width to nnn (an integer). Default is W071.

/D D+ sets debug output, the interpreter state vars and PDP8 machine state vars are reported. Default is D- (false).

T.7 SvUtil - Save File Utility (ODT/FUTIL)

SvUtil is a simple ODT-like octal debugger for overlaid save files. This program initially implemented accurate overlay editing as intended in FUTIL V7; then I got carried away and implemented more and more junk, until the present version (V1-0-F/18).

To use:

```
.R P,SVUTIL,savefile
```

The program heading of SvUtil is:

```
Program SvUtil(SaveFile : 'SYS:X.SV',
                  TTin : 'TTO: .OD',
                 TTout: '     .LS');
```

The control statement:

```
.R P,SVUTIL
```

will execute SvUtil interactively, and SvUtil will set up to access SYS:X.SV, the default .SV file name. Any other file may be specified as the first parameter to SvUtil, but only .SV format files will be accepted for editing/patching.

Alternatively, SvUtil may be given a file of patches as its second file parameter, and then it runs in "batch" mode. See file "SSRFC.OD" in the distribution kit for an example.

When it runs, SvUtil first prints the name and length of the file to be patched. SvUtil commands parallel those of OS/8 ODT (or FUTIL ODT mode) very closely. To open a location, type the desired address in octal followed by a slash:

```
nnnnn/ xxxx
```

SvUtil will respond by displaying the contents of the address in octal, which is the only available display mode.

To close a location, type a carriage return.

To modify a location, first open it, then type the new contents in octal, followed by a carriage return.

Typing a line feed to SvUtil will cause the current location to be closed, the address to be incremented by one, and the new location to be opened and contents displayed.

Modified blocks are written back to disk automatically when an address in a different block is opened, or when the 'Q' (quit) key command is typed.

Type 'Q' (the quit command) to leave SvUtil and return to OS/8.

A full address consists of an overlay part and a location in the overlay. For example, an address in the PMD overlay (in the RTS) used in a patch in section 5.4 is specified as:

```
3.13270/ 0067
```

This is in overlay 3 ('3.') of level 4 (but level is wholly determined by the address, and is never be specified by the user of SvUtil); address is 13270 (field 1 location 3270). This is a location in the P8RTS part of PMD.

The remainder are rough notes taken from the source of SvUtil.

Commands Implemented:

```
[overlay.]address / contents  replacement
[     oo.]aaaaaa  / cccc      rrrr

char  effect
----  ------

CR  - store rrrr if given, close current location
LF  - store rrrr if given, close current, increment address,
      open next and display
\   - store rrrr if given, close current, decrement address,
      open next and display. reverse LF with poke.
;   - store rrrr if given, close current, increment address,
      open next

      note: none of the rest 'poke' a value (store)

/   - open current location (display current
      address if none given)
n/  - set current location to n and open it for modification
*   - re-open the immediate last open location.
      (remembers up to 16 prior addresses!. No number is
      allowed before the '*'.)
n+  - close the current address, add n to the address,
      and open the new effective address. If no n was
      given, default is 1.
n-  - close the current address, subtract n from the
      current address, and open the new effective address.
      If no n is given, default is 1.
n(  - decrement address by one field (-n*4096) and open.
      (If no number is given, 1 is assumed.)
n)  - increment address by one field (n*4096) and open.
      (If no number is given, 1 is assumed.)
^   - (up arrow) (no number is allowed) close current
      location, use the contents as a memory reference
      instruction, and open the effective address.
_   - (left arrow or underline) (no number is allowed) close
      current location, use the contents as a 12-bit
      pointer to a new effective location in the same
      field, open it for modification.
.   - display current overlay number
n.  - set overlay number to n
```

```
nnnnnn  - assemble digit string as octal number (max 6 digits)
"c  - double quote: take one full ASCII character "c" and use
      it as a number to replace the contents of the currently
      open location.
'cc - single quote: take the next two characters and pack them
      into one word, and use the word to replace the contents
      of the currently open location.
!   - 17-bit indirect reference using current word and next
      word as a (low;high) address pair, the same as the
      Pascal-OS/8 interpreter often does.
=n  - ensure match of current location contents with the
      number given after the "=".  See SSRFC.OD.
d   - toggle debug output (gives addr/disk block mapping)
h   - report save file header block
l   - display length in blocks of savefile
q   - return to OS/8 (writes last changes)
t   - change terminal (as in 'TT:')
w   - reserved for possible implementation of word search
m   - reserved for possible implementation of mask.
z   - setup ccb for modification
^c  - (control-c) abort, all but last block addressed is modified
^w  - (control-w) toggle re-write enable
null - (000 or 200 code) ignored
^l  - (control-l, form feed) ignored
tab - equivalent to a space
^z  - (control-z) end-of-file, 'Q' cmd simulated.
c   - comment line
```

Items found in OS/8 ODT that are not implemented in SvUtil:

nnnng		nnnnb	
m	w	a	l
c	d	^o	f

--- End of SvUtil documentation.

Appendix V - Validation Suite V3.1 Results.

The Pascal-OS/8 processor was checked on the Pascal Validation Suite Version 3.1, which serves to monitor compliance with the ISO Pascal Standard. The result of the testing are presented here. After each test description is a number of the form (VSnn), which denotes the validation suite test program number in ascending order as tested, and as ordered on the file of test programs.

Pascal Processor Identification

Processor:	Pascal-OS/8 Version 1-0-F
Computer:	Digital Equipment Corporation PDP8/e with 32K words, FPP8a, and RK8E disks.

Test Conditions

Tester:	John Easton (Co-Implementer)
Date:	1984-02-01 and 1984-02-06.
Validation Suite Version:	3.1

Conformance Tests

Number of tests passed:	168
Number of tests failed:	16

Details of Failed Tests.

6.1.2-2 shows that 'otherwise' is not an allowed spelling for identifiers (it is a reserved word). (VS4)

6.1.3-2 shows that spellings of identifiers are distinguished on only the first eight characters. (VS7)

6.4.2.2-7 shows that the ord(firstch) <> 0. (ord(firstch) = 32, firstch = ' '.) (VS38)

6.4.3.2-2 shows that chr(0) cannot be represented in a set of char. (VS60)

6.4.3.5-4 checks that file types can appear in a record declaration; it fails due to having more than one file open for extend on a device, an OS/8 restriction. (VS64)

6.4.3.5-11 checks that two local files with similar ID's are distinguished. The test is prevented from running due to ID length

limit of 8 chars. (VS71)

6.4.3.5-12 checks for a small number of simultaneous open-extend files, and fails due to an OS/8 restriction on the number of open-extend files on the same device. (VS72)

6.5.1-1 shows that a pointer to a file is not allowed. (VS83)

6.6.3.1-1 shows small limits on the number of formal parameters (maximum 15) and on the number of actual parameters (maximum 16). (VS92)

6.7.1-8 fails because chr(0) is not a value of type char. Intended to test set-constructors. (VS129)

6.7.1-9 exceeds the implementation limits on set size. (VS130)

6.7.1-10 fails because chr(0) is not a value of type char. (VS131)

6.7.2.4-6 fails with sets exceeding implementation limits (96 set members maximum). (VS140)

6.8.3.5-2 shows that very large (sparse) case statements are not supported. (VS148)

6.9-1 shows that recursive I/O fails due to an OS/8 limitation on the number open-extend files on the same device. (VS165)

6.9.3.5-1 fails, showing inaccuracy in the 7th digit when a real number is written to a file. (VS175)

Deviance Tests

Number of deviations detected: 160

Number of exceptions: 1

Details of Exception.

6.4.3.1-4 test is flawed with the use of an un-defined type identifier. (VS237)

Error Handling Tests

Number of errors detected: 137

Number of errors not detected: 24

Number of exceptions: 3

Details of Exceptions.

6.5.4-2 fails yet seems to pass but only because a pointer was randomly undefined. (VS365)

6.6.5.3-5 fails yet seems to pass but only because a pointer was randomly undefined. (VS399)

6.6.5.4-3 fails yet seems to pass but only because a pointer was randomly undefined. (VS425)

Details of Errors Not Detected.

6.2.3.5-1: The use of an undefined variable is not detected. (VS339)

6.4.3.3-10 fails to detect the error of accessing an undefined variant. (VS341)

6.4.3.3-11 fails to detect the access to a variant field with an undefined value. (VS343)

6.4.3.3-12 fails to detect the mis-use of a variant. The test program accesses a field of a variant which is not the current variant, thereby changing the selected variant, which should cause an error. (VS345)

6.4.3.3-13 fails to detect the access to a variant field with an undefined value. (VS347)

6.5.5-2 fails to detect use of put(f) while f^ is passed to a procedure. (VS367)

6.5.5-3 fails to detect use of put(f) while f^ is within the scope of a with statement. (VS369)

6.6.5.2-9 fails to detect use of put(f) while f^ is undefined. (VS383)

6.6.5.2-10 fails to detect an error when reset(f) is performed but f is undefined. (In Pascal-OS/8, eof(f) is true upon reset(f) with no f defined.) (VS385)

6.6.5.3-6 fails to detect dispose of a pointer with an outstanding reference as an actual parameter. (VS401)

6.6.5.3-7 fails to detect dispose of a pointer with an outstanding reference as an element of the record-variable-list of a with statement. (VS403)

6.6.5.3-8 fails to detect an error as a variable created by the use of the long form of new is used as an operand in an expression. (VS405)

6.6.5.3-9 fails for the same reason as 6.6.5.3-8. (VS407)

6.6.5.3-10 fails for the same reason as 6.6.5.3-8. (VS409)

6.6.5.3-11 fails to detect an error as a pointer is used after a dispose(pointer). (VS411)

6.6.5.3-13 fails to detect an error in activating a variant which is different from those specified by the long form of new. (VS413)

6.6.5.3-14 fails to detect an error in applying the short form of dispose (dispose(p)) to an identifying-value which had been created with the long form of new (new(p,c1,,cn)). (VS415)

6.6.5.3-16 fails to detect the use of the long form of dispose with a different number of parameters than the corresponding application of the long form of new. (VS417)

6.6.5.3-17 fails to detect an error when calling dispose(q,k1,,km) when the variants in the variable identified by the pointer q are different from those specified by the case-constants k1,,km. (VS419)

6.6.5.3-21 fails to detect an error caused by dereferencing a dangling pointer. (VS421)

6.7.2.2-12 fails to detect the write of an integer expression which overflows the numeric capability of the processor. (VS467)

6.8.3.9-19 fails to detect the erroneous use of an undefined (left-over) for-statement control variable. (VS483)

6.8.3.9-21 fails to detect the erroneous use of an undefined (left-over) for-statement control variable. (VS485)

6.8.3.9-22 fails to detect the erroneous use of an for-statement control variable after an un-executed for-statement. (VS487)

Quality Tests

Number of tests passed:	51
Number of tests failed:	7
Number of exceptions:	2

Details of Exceptions.

6.1.5-9 shows that very large integer and real constants exceed the implementation limits of Pascal-OS/8, but the diagnostic messages are adequate to explain the error. (VS507)

6.9.3.2-5 shows that the default Boolean write field width does not agree with the test expectations. (VS569)

Details of Failed Tests.

6.1.5-11 fails on quality of conversion of real constants. (VS509)

6.1.5-12 fails on quality of conversion of real constants. (VS510)

6.6.1-8 shows that the Pascal-OS/8 processor limit on procedure and function nesting is 13. (VS527)

6.6.3.1-6 shows that the Pascal-OS/8 processor limit on formal parameters is 15, and the limit on actual parameters is 16 (the test expects 50 for quality). (VS528)

6.6.5.3-12 shows that dispose is implemented but does nothing. (VS529)

6.6.6.2-8 shows that exp fails when tested with an operand that is too large, x=1.41705E+003. (VS532)

6.9.1-8 shows that the write and read of real numbers is only approximately correct. (VS555)

Implementation Defined Tests

Number of tests run: 30

Details of Implementation Defined Tests

See also appendix I for an Implementation-Defined Checklist.

6.1.9-5 shows that alternate comment delimiters ((**) for {}) and alternate array brackets ((..) for []) are implemented. (VS562)

6.4.2.2-10 shows that MaxInt = 8388607. (VS563)

6.4.2.2-11 shows that real numbers have at most 8 significant digits. (VS564)

6.7.2.2-17 shows that real number have only 7 significant digits. (VS567)

6.6.2-11 shows that the Digital Equipment Corporation FPP8a firmware can not normalize certain floating-point quantities for which a valid normalized representation exists. After altering the test to avoid this problem, the following results were obtained. (VS566)

```
  beta =    2            t =   23
   rnd =    1         ngrd =    0
machep =  -23        negep =  -23
  iexp =   12       minexp =-2048
maxexp = 2047          eps = 1.19209e-007
epsneg = 1.19209e-007  xmin = 3.09431e-617
  xmax = 1.61587e+616
```

6.5.3.2-6 shows that the evaluation order of indexed-expressions of an indexed-variable is from left to right. (VS574)

6.6.5.2-16 and 6.6.5.2-17 show that the number of evaluations of the file parameter of the procedure read(f,v1,v2,v3) is one. (VS575, VS576)

6.7.2.3-3 and 6.7.2.3-4 show that boolean expressions are fully evaluated. (VS583, VS584)

6.8.2.2-1 and 6.8.2.2-2 show that the variable is accessed before the expression is evaluated in assignment statements. (VS586, VS587)

6.8.2.3-1 shows that the actual parameters are evaluated in left-to-right order. (VS588)

Level 1 Tests

Pascal-OS/8 does not support Level 1 of the ISO standard.

Extension Tests

6.8.3.5-16 shows that the case-statement completer ("otherwise clause") is supported. (VS634)