UPDATE PACKAGE

UPD4303-192

for

PASCAL REFERENCE GUIDE, DOC4303-191

June 1983

This Update Package, UPD4303-192, is Update 1 for the December 1982 Edition of the <u>Pascal Reference Guide</u>, DOC4303-191. This package contains 264 pages. A list of effective pages appears on the next page.

Changes made to the text since the last printing are identified by vertical bars in the margin. Change bars with numbers identify new Pascal features of Software Release 19.2. Change bars without numbers identify documentation corrections and clarifications.

Copyright © 1983 by Prime Computer, Incorporated Technical Publications Department 500 Old Connecticut Path Framingham, MA 01701

The information contained on these updated pages is subject to change without notice and should not be construed as a commitment by Prime Computer Corporation. Prime Computer Corporation assumes no responsibility for any errors that may appear in this package.

PRIME and PRIMOS are registered trademarks of Prime Computer, Inc. PRIMENET, RINGNET, Prime INFORMATION and THE PROGRAMMER'S COMPANION are trademarks of Prime Computer, Inc. (Pages with changes, enclosed with this package, are <u>underlined</u>.)

Effective Pages for the <u>Pascal Reference Guide</u> at Software Release 19.2.

Pages	Pages
ii to v vi to ix x to xiii	8-1 to $8-2\frac{8-3}{8-4} to 8-16$
1-1 to 1-2 $\frac{1-3}{1-4}$ to 1-6	9-1 to 9-8 9-9 to 9-9A 9-10 to 9-11 9-12
2-1 to 2-2 $\frac{2-3}{2-4}$ to 2-6	9-13 to 9-17 9-18 to 9-18A 9-19 to 9-22
$\frac{2-7}{2-8} \text{ to } 2-10$ $\frac{2-11}{2-12} \text{ to } 2-13$ $\frac{2-14}{2-15} \text{ to } 2-17$	$ \begin{array}{r} 10-1 \\ \underline{10-2} \text{ to } 10-3 \\ 10-4 \\ \underline{10-5} \text{ to } 10-5A \\ \underline{10-6} \text{ to } 10-10 \end{array} $
3-1 $3-2$ $3-3$ to 3-8	$\frac{10-11}{10-11} \text{ to } 10-12$ $\frac{10-13}{10-23} \text{ to } 10-23$
4-1 to $4-4\frac{4-5}{1-6} to 4-8$	<u>11-1</u> 11-2 to 11-4 <u>11-5</u>
$ \frac{4-6}{4-9} \\ \frac{4-9}{4-10} \\ \frac{4-11}{4-12} $	A-1 to A-3 A-4 to A-4A A-5
5-1 to $5-35-45-5$ to $5-15$	<u>B-1</u> B-2 to B-7 B-8 to B-9
$\frac{6-1 \text{ to } 6-2}{6-3 \text{ to } 6-8}$ $\frac{6-9}{6-9}$	D-1 D-2 D-3 to D-8 D-9
$\begin{array}{r} 6-10 \text{ to } 6-13 \\ \underline{6-14 \text{ to } 6-14L} \\ \overline{6-15 \text{ to } 6-16} \\ \underline{6-17 \text{ to } 6-17A} \\ \overline{6-18 \text{ to } 6-33} \end{array}$	<u>X-1 to X-16</u>

7-1 7-2 to 7-4 7-5 to 7-67-7 to 7-7

7-7 to 7-7A 7-8

Pascal Reference Guide

DOC4303-191

Second Edition

by A. Paul Cioto

This guide documents the software operation of the Prime Computer and its supporting systems and utilities as implemented at Master Disk Revision Level 19.1 (Rev. 19.1).

Prime Computer, Inc. 500 Old Connecticut Path Framingham, Massachusetts 01701

COPYRIGHT INFORMATION

The information in this document is subject to change without notice and should not be construed as a commitment by Prime Computer Corporation. Prime Computer Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

> Copyright © 1982 by Prime Computer, Incorporated 500 Old Connecticut Path Framingham, Massachusetts 01701

PRIME and PRIMOS are registered trademarks of Prime Computer, Inc.

PRIMENET, RINGNET, PRIME INFORMATION, PRIMACS, MIDASPLUS, and THE PROGRAMMER'S COMPANION are trademarks of Prime Computer, Inc.

HOW TO ORDER TECHNICAL DOCUMENTS

U.S. Customers

Prime Employees

Software Distribution Prime Computer, Inc. 1 New York Ave. Framingham, MA 01701 (617) 879-2960 X2053, 2054

Customers Outside U.S.

Contact your local Prime subsidiary or distributor.

Communications Services MS 15-13, Prime Park Natick, MA 01760 (617) 655-8000 X4837

PRIME INFORMATION

Contact your Prime INFORMATION dealer.

PRINTING HISTORY - PASCAL REFERENCE GUIDE

Edition	Date	Number	Software Release
First Edition	October 1980	IDR4303	17.6
Update 1 Update 2	December 1980 Julv 1982	PPU2600-080 PPU2600-086	18.1
Second Edition	December 1982	DOC4303-191	19.1

This edition is a complete revision of IDR4303. It incorporates update material up to and including software release 19.1, corrects all known errors, and has been revised for clarity.

Changes made to the text since the last printing have been indicated with change bars in the margin. Change bars with numbers indicate technical changes. Those without numbers indicate rewrites for clarification or additional information. Appendixes A and D are new.

SUGGESTION BOX

All correspondence on suggested changes to this document should be directed to:

A. Paul Cioto Technical Publications Department Prime Computer, Inc. 500 Old Connecticut Path Framingham, Massachusetts 01701

Contents

ABOUT THIS BOOK

xi

PART I - OVERVIEW

1 INTRODUCTION TO PRIME PASCAL

The Pascal Language	1-2
Prime Pascal	1-2
Contents of This Book	1-2
Related Documents	1-4
Interface to Other Languages	1-6

PART II - COMPILING, LOADING, AND EXECUTING PROGRAMS

2 USING THE PASCAL COMPILER

Introduct	zion	2-1
Invoking	the Compiler	2-2
Compiler	Error Messages	2-2
Filename	Conventions	2-4
Compiler	Options	2-6
Compiler	Option Abbreviations	2-13
Compiler	Switches	2-16

3 LOADING AND EXECUTING PROGRAMS

Loading Programs	3-1
Executing Programs	3-7

PART III - PRIME PASCAL LANGUAGE REFERENCE

4 PASCAL LANGUAGE ELEMENTS

4-2
4-4
4-7
4-7

Numeric Constants	4-8
Character-strings	4-11
Declarations and Statements	4-11
Line Format	4-11
Comments, Blanks, and Ends of Lines	4-11

5 PASCAL PROGRAM STRUCTURE

Program Heading	5-1
The Block	5-3
Declaration Part	5-4
LABEL	5-4
CONSTANT	5-6
TYPE	5-6
VARIABLE	5-7
PROCEDURE and FUNCTION	5-9
Executable Part	5-9
A Program Example	5-11

6 DATA TYPES

Scalar Data Types	6-1
Standard Scalar Data Types	6-2
INTEGER	6-3
LONGINTEGER	6-4
REAL	6-6
LONGREAL	6-7
BOOLEAN	6-8
CHAR	6-8
User-defined Scalar Data Types	6-10
Enumerated	6-10
Subrange	6-12
Structured Data Types	6-14
The ARRAY Type	6-14
The RECORD Type	6-20
The SET Type	6-25
The FILE Type	6-27
TEXT	6-29
The Pointer Type	6-31

7 EXPRESSIONS

Operands	7-1
Operators	7-2
Arithmetic Operators	7–2
Relational Operators	7–3
SET Operators	7–5
BOOLEAN Operators	7-6
Integer Operators	7-7
Operator Precedence	7-7

8 STATEMENTS

Summary of Statements	8-1
Assignment Statement	8-2
Procedure Statement	8–3
Compound Statement	8-4
Empty Statement	8–5
Control Statements	8–5
Repetitive Statements	8 6
REPEAT	8–6
WHILE	8–7
FOR	8-8
Conditional Statements	8-10
IF	8–10
CASE	8-11
Unconditional Statement	8-14
GOTO	8-14
WITH Statement	8-16

9 PROCEDURES AND FUNCTIONS

Parameters	9-2
Procedures	9-9
Functions	9-12
Forward Procedures and Functions	9-14
External Procedures and Functions	9-15
Recursive Procedures and Functions	9-19

10 INPUT AND OUTPUT

Inputting and Outputting Data	
at the Terminal	10-2
Inputting and Outputting Data	
with PRIMOS Files	10-6
Creating and Using Input	10 0
Data Filog	10-6
Data FILES	10-0
ine RESET Procedure	10-7
Creating and Using Output	
Data Files	10-11
The REWRITE Procedure	10-11
I/O Procedures and Functions	10-14
Input File-handling Procedures	10-15
GET	10-15
READ	10-15
READLN	10-17
Output File-handling Procedures	10-18
PUT	10-18
WRITE	10-18
WRITELN	10-22
BOOLEAN Functions	10-22
EOF	10-22
FOLN	10-23

Auxiliary	Procedures	10-23
PAGE		10-23
CLOSE		10-24

11 STANDARD FUNCTIONS

Arithmetic Functions	11-1
ABS	<u>11–1</u>
SQR	11-1
SIN	11-1
COS	11-1
EXP	11-2
LN	11-2
SQRT	11-2
ARCTAN	11-2
Transfer Functions	11-2
TRUNC	11-2
ROUND	11 - 2
Ordinal Functions	11-3
ORD	11-3
CHR	11-3
SUCC	11-3
PRED	11-4
BOOLEAN Functions	11-5
ODD	11-5
EOF	11-5
EOLN	11-5

APPENDIXES

A SUMMARY OF PRIME EXTENSIONS AND RESTRICTIONS

Prime	Extensions	A-1
Prime	Restrictions	A-5

B DATA FORMATS

Overview	B-1
INTEGER Type Data	B-2
LONGINTEGER Type Data	B-2
Subrange Type Data	B-3
REAL Type Data	B3
LONGREAL Type Data	B3
CHAR Type Data	B-4
BOOLEAN Type Data	B-4
Enumerated Type Data	B-4
ARRAY Type Data	B5
RECORD Type Data	B-5
SET Type Data	B5
FILE Type Data	B6
Pointer Type Data	B-8

C ASCII CHARACTER SET

Prime Usage	C-1
Special Characters	C-2
Keyboard Input	C-2

D INTERFACING PASCAL TO OTHER LANGUAGES

Overview	D-1
Interfacing INTEGER, BOOLEAN,	
and Enumerated	D-3
Interfacing LONGINTEGER	D-4
Interfacing REAL	D-4
Interfacing LONGREAL	D-5
Interfacing CHAR and ARRAY	
OF CHAR	D5
Interfacing Pointer	D-6
Interfacing SET	D-7
Interfacing RECORD	D7

INDEX

X-1

Part II -- Compiling, Loading, and Executing Programs

- Chapter 2 provides information on the use of Prime's Pascal compiler, including compiler options.
- Chapter 3 provides information on loading and executing programs with Prime's SEG utility.

Part III -- Pascal Language Reference

- Chapter 4 provides brief descriptions of Pascal language elements and of terms used throughout Part III.
- Chapter 5 lists the fundamental elements of the Pascal program structure.
- Chapter 6 describes the data types available in Pascal, including two Prime extension data types called LONGINTEGER and LONGREAL.
- Chapter 7 describes the use of Pascal expressions.
- Chapter 8 describes the use of executable Pascal statements.
- Chapter 9 describes the use of procedures and functions, including external procedures and functions, which are declared with Prime's EXTERN attribute.
- Chapter 10 offers a detailed discussion of how to input and output data in Prime Pascal.
- Chapter 11 lists standard Pascal functions.

Appendixes

- Appendix A summarizes Prime extensions and restrictions to standard Pascal. It also references the chapter in which each extension or restriction is discussed.
- Appendix B illustrates how Prime Pascal data types are represented in storage.
- Appendix C lists the ASCII character set, which Prime Pascal uses.
- Appendix D lists guidelines for interfacing Pascal to some of Prime's other high-level languages.

Error Messages

Pascal compiler error messages, which were designed to be self-explanatory, appear on your terminal at compile time, and in the listing file if one is created. Therefore, the messages are not listed in this book.

RELATED DOCUMENTS

In addition to the <u>Pascal Reference Guide</u>, you will most likely need other documents to help you take full advantage of Prime's powerful utilities, which are separately priced products. These documents are listed below.

Prime User's Guide

Complete instructions for creating, loading, and executing programs in Prime Pascal or in most Prime languages, plus extensive additional information on Prime system utilities for programmers, are found in the <u>Prime User's Guide</u>. The <u>Prime User's Guide</u> and the <u>Pascal Reference</u> <u>Guide</u> are both essential to the Pascal programmer.

The <u>Prime User's Guide</u> also contains a complete guide to all Prime documentation.

Draft Proposal "X3J9/81-093" Programming Language Pascal

The definitive reference for standard Pascal is <u>The Draft Proposal</u> "X3J9/81-093" Programming Language Pascal. Every installation that uses Pascal extensively should have a copy of this proposed standard, which may be obtained from American National Standards Institute, 1430 Broadway, New York, NY 10018.

New User's Guide to EDITOR and RUNOFF

Prime's EDITOR is an interactive line-oriented text-editing utility. It is used to enter and modify text in the computer. New programs that do not rely on cards or tapes can be input to the system at a terminal using EDITOR.

The <u>New User's Guide to EDITOR and RUNOFF</u> contains a complete description of the EDITOR, and describes RUNOFF, Prime's text-formatting utility. It also provides a basic introduction to the Prime system for those with little or no computer experience. The caret (or arrow) that appears just above the error message points to the actual error on the line of code. The following is an example of an error message:

OK, <u>PASCAL TEST.PASCAL</u> [PASCAL Rev. 19.1] 14 END {main program} ERROR 31 SEVERITY 3 BEGINNING ON LINE 14

Missing dot at program end.

When compilation is complete and all the error messages have been listed on the terminal, the compiler tells you how many errors were encountered and the maximum severity. For example:

0013 ERRORS (PASCAL-REV. 19.1) MAXIMUM SEVERITY IS 3

The significance of the severity code is:

Severity

Description

- 1 Warning
- 2 Error that the compiler has attempted to correct
- 3 Uncorrected error (prevents optimization, code generation, and therefore successful compilation)
- 4 Error that immediately halts compilation

A severity 1 or 2 error will not prevent execution of your program, but the output may be unpredictable.

Error Messages Involving %INCLUDE Files

A %INCLUDE file is a Prime extension. It is an external file that is compiled with the main program after the %INCLUDE statement. The %INCLUDE statement is followed by the name of the file to be included. The format is:

%INCLUDE 'filename';

%INCLUDE files can hold any legal Pascal code -- declarations as well as executable statements. The files could, for example, contain long lists of variable declarations. (For more information on %INCLUDE files, see Chapter 5.) 19.1

18.3

If you compile a program that inserts a %INCLUDE file, and there are compile-time errors in that file, a special type of error message format is printed at the terminal:

line-number> line-of-code

ERROR xxx SEVERITY y BEGINNING ON LINE line-number IN FILE 'filename' explanation

line-number The number of the line in the %INCLUDE file where the error occurred. (Lines of code in %INCLUDE files are numbered separately, and the numbers are enclosed in angle brackets in the listing file.)

line-of-code The actual erroneous line of code in the %INCLUDE file.

xxx The error code number.

y Severity code number.

'filename' The name of the %INCLUDE file.

explanation Description of the error and possible remedies.

The caret points to the erroneous line of code.

Here is an example of a %INCLUDE file error message:

<23> VAR a : integer;

ERROR 2 SEVERITY 3 BEGINNING ON LINE 23 IN FILE 'test-1' This item in a variable definition list is already defined in this block.

The compiler adds the number of errors from the %INCLUDE file to the number of errors in the main program, and gives the total number of errors at the end of compilation.

FILENAME CONVENTIONS

When you compile a program with the PASCAL command, and there are no severity 3 or 4 errors, the compiler creates an object (binary) file. It also creates a source listing file if the -LISTING option is specified on the command line. In order for you and the compiler to identify and compile the source file and create the object and listing files, the "suffix" conventions, which are described below, should be used to name these files on Rev. 18 (or higher) systems.

Table 2-1

Options Commonly Used and Not Commonly Used (Defaults are underlined.)

Options Commonly Used	Options Not Commonly Used
-BINARY [argument]	-BIG and -NOBIG
-DEBUG and -NODEBUG	-64V and $-32I$
	-EXPLIST and <u>-NOEXPLIST</u>
-ERRITY and -NOERRITY	-EXTERNAL and -NOEXTERNAL
-LISTING [argument]	-FRN and <u>-NOFRN</u>
-MAP and -NO_MAP	-INPUT pathname
	-OFFSET and -NOOFFSET
-OPTIMIZE, -OPT1, -OPT3, and -NOOPTIMIZE	-PRODUCTION and <u>-NOPRODUCTION</u>
-RANGE and -NORANGE	-SILENT and <u>-NOSILENT</u>
-UPCASE	-SOURCE pathname
-XREF and -NOXREF	-STANDARD and -NOSTANDARD
	-STATISTICS and <u>-NOSTATISTICS</u>

-BIG and <u>-NOBIG</u>

-BIG and -NOBIG determine the type of code generated for references to ARRAY or RECORD formal variable parameters in a subprogram.

With -BIG, an ARRAY or RECORD formal variable parameter can become associated with any ARRAY or RECORD, even if the ARRAY or RECORD crosses a segment boundary.

With -NOBIG, an ARRAY or RECORD formal variable parameter can be associated only with an ARRAY or RECORD that does not cross a segment boundary.

See ARRAY or RECORD Type Variable Parameters in Chapter 9 for details.

<u>-BINARY</u> [argument]

The -BINARY option generates an object (binary) file. If this option is not given, -BINARY YES will be assumed. The argument may be:

pathname Object code will be written to the file <u>pathname</u>.

YES Object code will be written to the file named program.BIN, or B_program, in the user's UFD, where <u>program</u> is the name of the source file. (This is the default.)

NO No object file will be created. Specified when only a syntax check or listing is desired.

-DEBUG and -NODEBUG

18.2

The -DEBUG option generates code for Prime's source level debugger. With -DEBUG, the object file is modified so that it will run under the debugger. Execution time increases, and the code generated will not be optimized.

-NODEBUG, the default, causes no debugger code to be generated.

See the <u>Source Level Debugger Guide</u> for information about debugging programs.

<u>-ERRITY</u> and -NOERRITY

18.0 The -ERRTTY option prints error messages at the user's terminal. -NOERRTTY suppresses this function.

-OPTIMIZE, -OPT1, -OPT3, and -NOOPTIMIZE

These options control the optimization phase of the compiler.

-OPTIMIZE, the default, will cause the object code to be optimized. Optimized code runs more efficiently than nonoptimized code, but takes somewhat longer to compile.

The -OPT1 option optimizes less code and generates less efficient code than -OPTIMIZE, but compilation time is faster than -OPTIMIZE.

The -OPT3 option optimizes more code and generates more efficient code than -OPTIMIZE, but compilation time is slower than -OPTIMIZE.

When -NOOPTIMIZE is invoked, optimization does not occur. Execution time is slowest, and compile time is fastest.

-PRODUCTION and <u>-NOPRODUCTION</u>

-PRODUCTION produces alternative option-controlling code for the debugger.

-PRODUCTION is similar to DEBUG, except that the code generated will not permit insertion of statement breakpoints. Execution time is not affected.

-NOPRODUCTION will cause no production-type code to be generated.

-RANGE and <u>-NORANGE</u>

-RANGE checks for out-of-bounds values of array subscripts and character substring indexes. Error-checking code is inserted into the object file. If an array subscript or character substring index takes on a value outside the range specified when the referenced data item was declared, a runtime error will be generated. Range checking decreases the efficiency of the generated code.

With -NORANGE, out-of-bounds values will not be detected. The program will be more vulnerable to errors, but will execute more quickly.

19.1

19.1

-SILENT and -NOSILENT -SILENT suppresses severity 1 error messages. Severity 1 error messages will not be printed at the terminal and will be omitted from any listing file. -NOSILENT causes severity 1 error messages to be retained. -SOURCE pathname The -SOURCE option, which is identical to the -INPUT option, is obsolete and not useful. -SOURCE designates the source file pathname to be compiled: PASCAL -SOURCE pathname It is not useful because it produces the same results as: PASCAL pathname pathname must not be designated more than once on the command line. -STANDARD and <u>-NOSTANDARD</u> The -STANDARD option generates a severity 1 error message when your code's syntax is non-ANSI standard Pascal. -NOSTANDARD does not cause a severity 1 error to be generated. -STATISTICS and <u>-NOSTATISTICS</u> The -STATISTICS option lists compilation statistics at the terminal after each phase of compilation. For each phase the list contains: DISK Number of reads and writes during the phase, excluding those needed to obtain the source file SECONDS Elapsed real time Internal buffer space used for symbol table, in 16K byte SPACE units Disk I/O time used PAGING CPU time used in seconds, followed by the clock time CPU when the phase was completed -NOSTATISTICS causes no statistics to be printed.

-UPCASE

The -UPCASE option causes the compiler to map lowercase variables to uppercase. With -UPCASE, the compiler does not distinguish between lowercase variables and uppercase variables, except within character strings.

18.2

-XREF and <u>-NOXREF</u>

The -XREF option appends a cross-reference to the source listing. A cross-reference lists, for every variable, the number of every line on which the variable was referenced.

-NOXREF causes no cross-reference listing to be generated.

▶ <u>--64V</u> and -321

These determine the addressing mode to be used in the object code. -64V is a segmented virtual addressing mode for 16-bit machines. -32I is a segmented virtual mode, which takes maximum advantage of the 32-bit architecture of Prime's more advanced models (P450 and up).

COMPILER OPTION ABBREVIATIONS

Most compiler options have abbreviations that are accepted by the compiler. For example, instead of typing -LISTING on the command line, you could simply type -L. A list of Prime's recommended abbreviations, along with a summary of options in straight (nonpaired) alphabetical order, is given in Table 2-2.

Table 2-2

-			
	Option	Abbreviation	Significance
	-BIG	-BIG	Generate boundary-spanning code
	-BINARY	-В	Create object file
18.2	-DEBUG	-DE	Generate debugger code
18.0	-ERRITY	-ERRT	Print error messages at terminal
	-EXPLIST	-EXP	Generate an expanded source listing
	-EXTERNAL	-EXT	Generate external procedure definitions
	-FRN	-FRN	Generate floating-point round instructions
	-INPUT	-I	Designate source file
	-LISTING	-L	Create source listing
18.3	-MAP	-MA	Print listing file with map
	-NOB IG	-NOB	Don't generate boundary-spanning code
18.2	-NODEBUG	-NOD	Don't generate code for debugger
18.0	-NOERRITY	-NOERRT	Don't print error messages at terminal
	-NOFRN	-NOFRN	Don't generate FRN instruction
18.3	-NO_MAP	-NOM	Don't include a map in listing file
	-NOOFFSET	-NOOF	Don't append an offset map to source listing
	-NOOPTIMIZE	-NOOP	Don't optimize object code

Summary of Compiler Options and Abbreviations (Defaults are underlined.)

Table 4-1 Pascal Punctuation Symbols

Symbol	Description
+	Addition Identity Set union
-	Subtraction Sign-inversion Set difference
*	Multiplication Set intersection
/	Division (real)
=	Equal to Set equality Type identifier and type separator Constant identifier and constant separator
<	Less than
>	Greater than
[]	Subscript list or set constructor delimiters
•	Decimal point Record selector Program terminator
r	Parameter or identifier separator
:	Variable name and type separator Label and statement separator
;	Statement separator Record field separator Declaration separator
^	File or pointer variable indicator
()	Parameter list, identifier list, or expression delimiters
<>	Not equal to Set inequality

Tab.	Le 4-1	(cont	inued)
Pascal	Punctu	ation	Symbols

Symbol	Description
<=	Less than or equal to Set inclusion ("is contained in")
>=	Greater than or equal to Set inclusion ("contains")
:=	Assignment Operator
• •	Subrange Specifier
{}	Comment delimiters
/* */	Comment delimiters (Prime extension)
(* *)	Comment delimiters
T	Character-string delimiter (apostrophe)
&	Bit Integer AND operator (Prime extension)
I	Bit Integer OR operator (Prime extension)

18.3

Table 4-3 Standard Identifiers

Constants					
FALSE	TRUE	MAXINT			
Types					
INTEGER BOOLEAN	LONG INT CHAR	EGER* REAL TEXT	LONGREAL*	l	
Files					
INPUT	OUTPUT				
Directives					
FORWARD	EXTERN*				
Functions					
ABS ARCTAN CHR COS EOF EOLN	EXP LN ODD ORD PRED ROUND	S IN SQR SQRT SUCC TRUNC			
Procedures					
CLOSE* DISPOSE GET NEW	PAGE PUT READ READLN	RESET REWRITE WRITE WRITELN			
* Prime extension identifiers					

19.1

19.1 There are two ways of expressing real and longreal numbers:

- 1. In decimal notation, the number is expressed by an optional sign, a whole number part, a decimal point, and a fractional part. There must be at least one digit on each side of the decimal point.
- 2. In scientific notation, the number is represented by a value, followed by the letter E or D, which is followed by an exponent. The letter E is used if the number is REAL. The letter D is used if the number is LONGREAL. The value consists of an optional sign, one or more digits, and an optional decimal point and fractional part. The exponent must be an integer with an optional sign. The letter E or D is read as "times 10 to the power of". This is a convenient way to represent very large or very small numbers.

No comma may appear in a number. Examples:

Valid Integer/Longinteger	Invalid Integer
23	-32,768 (No comma allowed)
-100	
+40000 (longinteger)	
Valid Real/Longreal Number	Invalid Real Number
-0.1	.1 (Must be a digit to
1E6 (1000000)	point)
5E-8 (0.0000005)	1. (Must be a digit to the
-87.35E+15 (-8735000000000000)	right of the decimal point)
-7.0E-6 (-0.000007)	-8.0E-6.3 (Only whole number
2.1D01 (longreal)	1,234D+20 (No comma allowed)

19.1

19.1

19.1

1.234567 (longreal)

6 Data Types

Every constant, variable, function, or expression must have a data type. The data type determines the set of values a variable may assume or a function or an expression may generate. The data type also determines which operations may be performed on the values and how these values are represented in storage.

This chapter summarizes the data types available in Prime Pascal — standard Pascal data types as well as Prime extensions. There are two Prime extension data types, LONGINTEGER and LONGREAL. Each of these data types is described later in this chapter.

Figure 6-1 illustrates all of the data types in Prime Pascal. The internal representations of data types are illustrated in Appendix B. Appendix D offers guidelines for interfacing Pascal data types with those of other languages. For more information about Pascal data types, consult a commercially available text.

SCALAR DATA TYPES

Scalar data types are the fundamental data types in Pascal. All other data types must be built from scalar data types.

Each scalar data type has a group of distinct values, called constants, which have a defined linear ordering. Thus, each scalar type is ordered. Any two of these constants can be compared by asking if one is less than, equal to, or greater than the other. The total number of constants in a type is called the cardinality of that type. 19.1



The Hierarchy of Data Types in Prime Pascal19.1*Prime extensions are flagged with an asterisk.

Figure 6-1

Scalar data types are divided into two classes: standard scalar data types and user-defined scalar data types. The standard scalar types are the predefined, built-in data types provided by Pascal. The user-defined scalar types are data types that you create and define in a program.

Standard Scalar Data Types

19.1 There are four standard scalar types -- INTEGER, REAL, BOOLEAN, and CHAR -- plus two Prime extension scalar types called LONGINTEGER and LONGREAL. The following program compares all of the printable characters (decimal 160-255) in Prime's character set, using relational operations:

```
PROGRAM Karacter;
VAR
  I : INTEGER;
BEGIN
  FOR I := 160 TO 255 DO
    BEGIN
      WRITE(CHR(I));
      IF ((CHR(I) \geq 'A') AND (CHR(I) <= 'Z')) THEN
        WRITELN(' This is a capital letter')
      ELSE
      IF ((CHR(I) \geq 'a') AND (CHR(I) \langle = 'z')) THEN
        WRITELN(' This is a small letter')
      ELSE
      IF ((CHR(I) \geq '0') AND (CHR(I) \leq '9')) THEN
        WRITELN(' This is a printable number')
      ELSE
        WRITELN(' This is punctuation or other character')
    END
```

END.

<u>Caution</u>

Prime's character set is represented by the decimal numbers 128 to 255. You should <u>not</u> use the CHR function on integers less than 128 or greater than 255. Any such attempt will produce unpredictable results.

To indicate a constant of the CHAR type, place an apostrophe (a single quote) on each side of the character. To indicate an apostrophe, write it twice. Examples:

'A'

171

1,1

- {Single quote}
- ' {Blank is considered a printable character.}

Note

A constant of the CHAR type is always a <u>single</u> character. Constructs such as '123' or 'STRING' are not constants of this type but are constants of a more complex type called ARRAY OF CHAR or "string", which is described later in this chapter. As was explained earlier, each character corresponds to its own internal integer, which is called the <u>ordinal number</u> of the character. Using the standard function ORD — the opposite of CHR — you can get a character's ordinal number. For example:

ORD('A') yields 193{Octal value 301}ORD('a') yields 225{Octal value 341}ORD('l') yields 177{Octal value 261}

There are two more standard functions particularly useful for processing character data -- PRED (predecessor function) and SUCC (successor function). Given a value, PRED produces the next lesser value and SUCC gives the next greater value. For example:

PRED('E') yields 'D' {The predecessor of 'E' is 'D'}
SUCC('E') yields 'F' {The successor of 'E' is 'F'}
PRED(8) yields 7 {The predecessor of 8 is 7}
SUCC(8) yields 9 {The successor of 8 is 9}
PRED(ORD('G')) yields 198 {The predecessor of G's ordinal
value is 198}
SUCC(ORD('F')) yields 199 {The successor of F's ordinal
value is 199}

Functions are described in detail in Chapter 11.

The relational operators =, $\langle \rangle$, $\langle \rangle$, $\langle \rangle$, $\langle =$, and $\rangle =$ can be used with all character constants. For more information, see Chapter 7.

User-defined Scalar Data Types

There are two user-defined scalar types -- enumerated and subrange.

<u>The Enumerated Types</u>: An enumerated type defines an ordered set of values by listing these values.

To create an enumerated type, use the following type definition:

TYPE type-identifier = (identifier-1, identifier-2 [,identifier-3]...);

<u>Array of Characters</u>: A line of text can be represented as an array of characters. This particular array is called ARRAY OF CHAR or "string".

A typical VAR declaration of an ARRAY OF CHAR would be:

VAR

A : ARRAY[1..60] OF CHAR;

The identifier "A" is an array with 60 character elements. A[1] is the first character, and A[60] is the last. Any character string value assigned to A must have 60 characters.

Here is an example of how an ARRAY OF CHAR (string) type is declared within a TYPE declaration:

TYPE STRING1 = ARRAY[1..10] OF CHAR;

Two more examples follow:

```
TYPE

STRING1 = ARRAY[1..10] OF CHAR;

VAR

STRING2 : STRING1;

BEGIN

STRING2 := 'ABCDEFGHLJ';

STRING2 := 'AB'

{This is an invalid assignment.}

{The string must contain 10}

{characters.}
```

END.

Here is another example:

```
TYPE

LENGTH = 1..30;

STRING30 = ARRAY [LENGTH] OF CHAR;

VAR

ALPHA : STRING30;

I : LENGTH;

BEGIN

FOR I := 1 TO 30 DO

READ (ALPHA[I])

END.
```

Note

Although Prime Pascal does not support the keyword PACKED in type definitions, an ARRAY OF CHAR is always stored as a packed ARRAY OF CHAR on Prime computers. 19.1

Array of Characters (the Prime Extension): At Rev. 19.1, the ARRAY OF CHAR was enhanced into a Prime extension that makes reading of these arrays much easier. On Rev. 19.1 (or higher) systems, you can read an array of characters as one unit, instead of reading one character at a time. For example, Prime's ARRAY OF CHAR function allows you to declare and read character arrays like this: **PROGRAM** Primearray; VAR A : ARRAY[1..10] OF CHAR; B : ARRAY[1..60] OF CHAR;BEGIN READLN(A); READLN(B) END. Previously, reading was done one character at a time within a loop: BEGIN FOR I := 1 TO 10 DO READ(A[I]);FOR I := 1 TO 60 DO READ(B[I])END. You can still use loops to read an ARRAY OF CHAR; however, it is easier and more efficient to use the Prime extension. Note If you do not have a Rev. 19.1 (or higher) system, then you must use the loops. You can read or write a Prime character array up to 256 characters long. Consider the following: VAR A : ARRAY[1..30] OF CHAR; BEGIN READLN(A); With READLN, if fewer than 30 characters are typed in, the remaining

in, only the first 30 characters will be assigned. You will not be warned that you have typed in extra characters.

7 Expressions

An expression is a single operand or a combination of operands and operators that are evaluated to produce a value.

OPERANDS

An operand may be any of the following expressions:

- A variable
- An unsigned or signed number
- A character string
- A constant identifier
- A function designator (explained in Chapter 9)
- NIL
- A set

Here are some examples of valid operands:

15 (x+y+z) SIN(x+y) [RED, C, GREEN] [1, 5, 10..19, 23] NOT P I * J + 1 -N

OPERATORS

Operators modify an operand or combine two operands. Operators can be classified as arithmetic, relational, set, Boolean, or integer. (Integer operators are Prime extensions.)

Arithmetic Operators

An arithmetic operator specifies computation to be performed on its operands to produce a single numeric value. Table 7-1 lists the binary and unary arithmetic operators and the data types of operands and results.

Table 7-1 Arithmetic Operators

Binary Operators	Type of Operands	Type of Result	
+ (add) - (subtract) * (multiply)	INTEGER/LONGINTEGER REAL/LONGREAL	INTEGER/LONGINTEGER if both operands are INTEGER/LONGINTEGER; otherwise REAL/LONGREAL	
/ (divide)	INTEGER/LONG INTEGER REAL/LONGREAL	REAL/LONGREAL	
DIV (divide with truncation)	INTEGER OF LONG INTEGER	INTEGER/LONG INTEGER	
MOD (modulus or remainder)	INTEGER OR LONG INTEGER	INTEGER/LONG INTEGER	
Unary Operators			
+ (identity) - (sign-inversion)	INTEGER/LONG INTEGER REAL/LONGREAL	Same as operand	

Relational Operators

The relational operators are used to compare values of data types -scalar, ARRAY OF CHAR (string), pointer, or SET. In any given comparison, both operands must be of the same type, except that INTEGER can be compared with LONGINTEGER, and REAL with LONGREAL. The result of the comparison is a BOOLEAN value, TRUE or FALSE. Table 7-2 lists the legal relational operators and data types of operands.

19.1

Operator	Operation	Type of Operands	
= <>	equality inequality	SET, scalar, pointer, or ARRAY OF CHAR	
< >	less than greater than	scalar or ARRAY OF CHAR	
<=	less or equal	scalar or ARRAY OF CHAR	
<=	set inclusion ("is contained in")	SET	
>=	greater or equal	scalar or ARRAY OF CHAR	
>=	set inclusion ("contains")	SET	
IN	set membership	first (left) operand is any scalar type (except REAL and LONGREAL), second (right) operand is a set of that type	

Table 7-2 Relational Operators

Here are some examples of relational operators.

First, let
 x := ['A', 'D', 'C', 'B']
 y := ['A', 'E']

then

19.1

x = ['A', 'B', 'C', 'D']	{true }
y <= x	{falæ}
у <> х	{true }
'B' IN X	{true }

Integer Operators

The integer operators & and ! are Prime extensions. They perform Boolean AND and OR operations on integers respectively. These operators also work on longintegers. For example, if you wanted to perform AND and OR operations on the two numbers 10 and 12, you could say:

```
VAR
    A,B,C,D : integer;
BEGIN
    A := 10;
    B := 12;
    C := A & B; {AND operation}
    D := A ! B; {OR operation}
    WRITELN(C);
    WRITELN(D);
END.
```

At the machine level, the two binary numbers that stand for decimal 10 and 12 are 1010 and 1100 respectively. (The 12 leading zeros are omitted.) During the AND and OR operations, the digit 1 means TRUE and 0 means FALSE. The first digit of 1010 is compared with the first digit of 1100, and so on, to produce new binary (and hence decimal) numbers C and D. The machine, therefore, calculates:

1010 AND 1100 = 1000 {decimal 8} 1010 OR 1100 = 1110 {decimal 14} C = 8 D = 14

Integer operators can be useful when you need a lot of Boolean TRUE and FALSE values or "switches" that can be set to 1 (TRUE) or 0 (FALSE) in the internal binary representation of any decimal number.

OPERATOR PRECEDENCE

The precedence among operators determines the order in which expressions are evaluated. The precedence of operators is as follows:

1.	Operations in parentheses	Highest precedence (done first)
2.	NOT, unary - and +	
3.	*, /, DIV, MOD, AND, &	
4.	+, -, OR, !	Ļ
5.	=, <>, <, >, <=, >=, IN	Lowest precedence (done last)

Order of Evaluation

When there are several operations at the same level of precedence, the operations are performed from left to right.

Parentheses may be used to override the normal evaluation order. An expression enclosed in parentheses is treated as a single operand, and is evaluated first. When expressions are contained within a nest of parentheses, evaluation proceeds from the innermost set to the outermost set (inside out).

For example:

7 +	A *	2 -	5 DIV	3 +	A	{Numbers below the operators indicate the order in which
2	1	4	3	5		the operations are performed.}

((7 + A) * 2 - 5) DIV 3 + A

1 2 3 4 5
The following are some guidelines for using assignment statements:

- The variable or function identifier and the expression must be of compatible types.
- Neither the variable/function identifier nor the expression should be a FILE type or a structured type with a FILE element.
- The variable or function identifier can be of type REAL and the expression can be of type INTEGER; however the converse is not possible. (You can assign an integer to a real, but not a real to an integer unless the TRUNC function is used.)
- The variable or function identifier can be of type LONGINTEGER and the expression can be of type INTEGER, but the converse may cause your program to fail. (You may assign an integer to a longinteger, but a longinteger will be truncated when assigned to an integer.) This rule also applies to REAL and LONGREAL for the same reason.

19.1

- Any element, group of elements, or expression that is of a particular SET type must be assigned to a variable or function identifier of the <u>same</u> SET type.
- The variable or function identifier and expression can be type ARRAY OF CHAR (string) as long as both arrays have the same number of elements.
- The variable or function identifier and expression can be subranges of each other.

PROCEDURE STATEMENT

A procedure statement activates the execution of a procedure. A procedure is a subprogram, which is declared in the main program.

The format of the procedure statement is:

procedure-identifier [(parameter-list)];

The <u>procedure-identifier</u> is the name of the procedure. When the procedure statement is encountered in the main program, the procedure is executed. The <u>parameter-list</u> is optional. If you want to pass values to and from the main program and the procedure, you would use parameters. The parameter-list is enclosed in parentheses, and the parameters are separated by commas.

Here are some examples of procedure statements:

PRINTHEADING;

TRANSPOSE(A, N, M);

BISECT(FCT, -1.0, +1.0, X);

For more information on procedures and functions, including external procedures and functions, see Chapter 9.

COMPOUND STATEMENT

A compound statement is a sequence of statements separated by semicolons. The general form of a compound statement is:

```
BEGIN
    statement-1 ; statement-2;...[statement-n]
END;
```

The keywords BEGIN and END must designate the start and the end of the sequence of a compound statement. They are not statements themselves. BEGIN and END should not be used on a single statement. <u>statement-1</u>, <u>statement-2</u>, etc. can be any Pascal statements. A compound statement can appear anywhere a single statement is allowed.

```
Example 1:
```

```
BEGIN

Z := X;

X := Y;

Y := Z

END;
```

```
Example 2:
```

```
IF FLAG = 1 THEN
BEGIN
COUNTER := 0;
READ (CHARACTER);
WHILE (CHARACTER);
WHILE (CHARACTER <> BLANK) DO
BEGIN
COUNTER := COUNTER + 1;
READ (CHARACTER)
END;
WRITELN (' THE NUMBER OF CHARACTERS = ', COUNTER)
END
ELSE
FLAG := 0;
```

```
Example 5:
   VAR
      I, J : INTEGER;
    PROCEDURE ADD2 (PROCEDURE A1);
      BEGIN {procedure ADD2}
        Al;
        Al
      END;
    PROCEDURE ADD1;
      BEGIN {procedure ADD1}
        I := I + 1
      END;
    PROCEDURE CALLPROC (PROCEDURE X (PROCEDURE Y); PROCEDURE Z);
      BEGIN {procedure CALLPROC}
        Z;
        X(Z)
      END:
    BEGIN {main program}
      I := 0;
      CALLPROC(ADD2, ADD1)
    END. \{I = 3\}
```

PROCEDURES

A procedure is a user-written independent program unit that performs a set of operations. A procedure must be declared in a procedure declaration, a forward procedure declaration, or an external procedure declaration before the procedure can be called by a procedure statement.

Procedure declarations are discussed below. Forward and external procedure declarations are discussed later in this chapter.

The external procedure declaration is a Prime extension to standard Pascal.

Procedure Declarations

A procedure declaration defines and names a procedure. The form of a procedure declaration is:

PROCEDURE identifier [(formal-parameter-list)]; block;

The keyword PROCEDURE begins a procedure declaration. The <u>identifier</u> is the name of the procedure. The list of <u>formal parameters</u>, if any, enclosed in parentheses, specifies the name of each formal parameter followed by its type-identifier. If you choose to use them, parameters can be passed by value or by reference to the subprogram. Parameters are discussed earlier in this chapter. 19.1

Except in forward or external declarations, the procedure heading described above is immediately followed by the <u>procedure block</u>.

A procedure block has the same general form as a program block. It may contain declarations for labels, constants, types, variables, procedures, and functions and a sequence of executable statements surrounded by a BEGIN and END pair. However, the procedure block ends with a semicolon instead of a period.

Unlike a function, the name of a procedure must not be assigned a value. Therefore, do not specify a data type for a procedure itself.

<u>Note</u>

Identifiers and labels declared in the main program are <u>global</u>. That is, they can be referenced throughout the entire program, including these procedures (or functions), so long as the procedures are contained within the main program (are not external). However, those identifiers and labels applying only to a particular procedure (or function) but not to the program as a whole should be declared within that procedure (or function). These identifiers and labels are <u>local</u>.

Invoking Procedures

L

A procedure statement invokes, or calls, a procedure. A procedure statement has the form:

procedure-identifier [(actual-parameter-1 [,actual-parameter-2]...)]

The <u>procedure-identifier</u> is the name of the called procedure. When the called procedure has one or more formal parameters defined in its heading, the procedure statement must contain the corresponding <u>actual</u> <u>parameters</u> along with the procedure-identifier.

```
Example 1:
    PROGRAM TEST;
       .
    PROCEDURE INDATA;...BEGIN...END;
    PROCEDURE SORT; ... BEGIN... END;
    PROCEDURE CUTDATA;...BEGIN...END;
    {Main program begins here.}
    BEGIN
       INDATA;
       SORT;
       OUTDATA
    END.
Example 2:
    PROGRAM CURVE(INPUT, CUTPUT);
    VAR
      X, Y : REAL;
      I : INTEGER;
    PROCEDURE PLOT(A, B: REAL; J: INTEGER); {A, B, & J are formal value
                                               parameters.}
      .
    BEGIN...END;
    PROCEDURE ENDPLOT;
      .
      ٠
    BEGIN...END;
    {Main program begins here.}
    BEGIN
      X := 0.0;
      Y := 1.0 + SIN(X);
      READLN(I);
      I := I + 2;
      PLOT(X, Y, I); {X, Y, and I are actual parameters.}
      .
      .
      ENDPLOT;
      ٠
      -
    END.
```

Standard Procedures

A standard procedure, denoted by a predefined identifier, is a built-in procedure supplied by the Pascal language.

Prime Pascal supports the following standard procedures:

- File Handling Procedures: RESET, GET, REWRITE, PUT, READ, READLN, WRITE, and WRITELN. (See Chapter 10.)
- I/O Auxiliary Procedures: PAGE and CLOSE. (See Chapter 10.)
- Dynamic Allocation Procedures: NEW and DISPOSE. (See Chapter 6.)

Note

The CLOSE procedure is a Prime extension to standard Pascal.

Use of the standard transfer procedures PACK and UNPACK in Prime Pascal will generate an error message and cause your program to fail because PACK and UNPACK are not supported in Prime Pascal. This is a Prime restriction.

FUNCTIONS

Functions are also user-written subprograms. Here are some characteristic traits of functions:

- The keyword FUNCTION is used instead of PROCEDURE.
- Similar to a procedure, a function is a subprogram.
- Unlike procedures and standard functions, the names of user-written functions must represent values. Procedure names and standard function names cannot represent values.
- Unlike a procedure, a data type must be specified for the function itself in the function heading.

A function is an independent program unit that accepts zero or more parameters to produce a single output value. A function must be declared in a function declaration, a forward function declaration, or an external function declaration before the function can be invoked.

Function declarations are discussed below. Forward and external function declarations are discussed later in this chapter.

The external function declaration is a Prime extension to standard Pascal.

<u>Using the -EXTERNAL Option Instead of $\{\$E+\}$:</u> An alternative to using the $\{\$E+\}$ switch in the subprogram is to use the -EXTERNAL option every time you compile the file of subprograms. For example:

PASCAL filename -EXTERNAL

The <u>filename</u> is the name of the file that contains the external subprograms. (See Chapter 2 for more information on compiling programs.)

<u>Defining External (Global) Variables with $\{\$E+\}$:</u> If you want your external subprograms to reference the variables that are declared in the calling program, you must use the $\{\$E+\}$ and $\{\$E-\}$ switches in the VAR declaration of the calling program. For example:

```
VAR

I, J : INTEGER;

{$E+}

X, Y, Z : INTEGER;

{$E-}
```

Here is an example of a program that calls an external procedure. It has one variable, ADDSUM, that is used externally:

```
PROGRAM File 1;
VAR
    I, J : INTEGER;
{$E+}
    ADDSUM : INTEGER;
{$E-}
PROCEDURE ADD(A, B : INTEGER); EXTERN;
BEGIN {main program}
    I := 23;
    J := 45;
    ADD(I, J); {external procedure is called here}
    WRITELN(ADDSUM)
END.
```

Here is the external procedure ADD, which the above program calls. Notice that the external variable ADDSUM <u>must also</u> be declared in the subprogram at the <u>top</u> of the file, <u>outside</u> the procedure or function block:

```
{$E+}
VAR
ADDSUM : INTEGER;
PROCEDURE ADD(A, B : INTEGER);
BEGIN
ADDSUM := A + B
END;
```

<u>Compiling and Loading Subprograms</u>: Remember that each external subprogram file must be compiled and loaded separately. After you have entered SEG's LOAD subprocessor, the main program must be loaded before the separately compiled subprograms. For more information on compiling, loading, and executing programs, see Chapters 2 and 3.

External subprogram names, as well as the names of main programs, cannot be more than 32 characters long.

Subprograms Written in Other Languages

Subprograms declared in external procedure or function declarations in the main program can be written in any Prime high-level language or Prime Macro Assembly (PMA) language with certain restrictions:

- There must be no conflict of data types for variables being passed as parameters. For example, a FIXED BINARY(15) in PL/I is equivalent to an INTEGER in Pascal.
- Programs compiled in either 64V or 32I mode cannot reference or be referenced by programs compiled in R mode. Programs in 64V or 32I mode may reference each other.

For more information on interfacing Pascal with other languages, see Appendix D.

10 Input and Output

In Prime Pascal, data can either be input from your terminal or be input from a PRIMOS input data file. Similarly, the output can either be written out to your terminal or to a PRIMOS output data file.

This chapter explains how to input and output data in Prime Pascal, using both of these methods.

Throughout this chapter, various built-in I/O (input/output) functions and procedures that manipulate data are discussed. These include eight file-handling procedures (RESET, GET, READ, READLN, REWRITE, PUT, WRITE, and WRITELN), two BOOLEAN functions (EOF and EOLN) and two auxiliary procedures (PAGE and CLOSE).

Note

Prime Pascal performs I/O operations only on data stored in disk files or data supplied at the terminal.

INFUTTING AND OUTPUTTING DATA AT THE TERMINAL

When you execute a program, and your program requests data at execution time, it can wait for you to input the data at your terminal. For example:

```
PROGRAM Add;
VAR
A, B, C : INTEGER;
BEGIN
READLN(A);
READLN(B);
C := A + B;
WRITELN(C)
END.
```

In the example above, the computer expects you to enter two integers at your terminal upon execution. The execution would look like this, where user input is underlined:

OK, <u>SEG ADD</u> <u>30</u> <u>50</u> 80 {computer writes out result here}</u> OK,

```
For more information on executing programs, see Chapter 3.
```

If you were using READs instead of READLNs in the example above, you could place the integers on the same line, separated by spaces or a comma. For example, given the following statements:

READ(X, Y); Z := X + Y; WRITELN(Z);

your terminal input and execution would look like this:

A space placed after the 30 and after the 50 signals the end of each integer. It also tells the computer that each integer has two digits. Notice that with READs, the computer outputs the sum on the same line as your input.

You can make the computer prompt you for input by putting WRITE or WRITELN statements in your program. For example:

```
VAR
  A,B,C : INTEGER;
BEGIN
  WRITELN('Enter two numbers:');
  READLN(A);
  READLN(B);
  C := A+B;
  WRITELN(C)
END.
```

Your input and execution would look like this:

```
OK,
SEG ADD
Enter two numbers:
10
20
30
OK,
```

If you were using READs on CHAR type data instead of INTEGER or REAL, you would not put spaces between the input characters. Therefore, with the following program:

```
PROGRAM Letters;
VAR
  X, Y, Z : CHAR;
BEGIN
  WRITE('Enter three letters: ');
  READ(X, Y, Z);
  WRITELN(X:10, Y, Z)
END.
```

your input and execution would look like this:

OK, <u>SEG LETTERS</u> Enter three letters: <u>POR</u> POR OK,

The 10 in the WRITELN statement formats the output so that nine spaces are placed before the P. Notice that the WRITE statement prompts you for input.

Using Erase and Kill Characters

PRIMOS provides two special character functions called <u>erase</u> and <u>kill</u>. The erase character (the double quotation mark) erases the immediately preceding character. For example, if you type 1235 when you wanted to type 1234, you can correct your mistake by typing the double quote followed by the correct input:

1235"4

The kill character (the question mark) deletes your entire current line. For example, if you mistakenly type this:

123456789

and were supposed to type this:

ABCDEFGHI

you can correct your mistake by typing the question mark followed by the correct input:

123456789?ABCDEFGHI

<u>Note</u>

Your System Administrator may have changed the Prime-supplied erase and kill characters to some other characters. If so, find out what they are. (You can change them yourself, too.)

<u>How to Use Erase and Kill on Terminal Input</u>: Before Rev. 19.1, use of Prime's erase and kill characters on input from the terminal was not possible because each character was assigned to the program as soon as it was typed. Not only was it too late to use an erase or kill character, but also an erase or kill character <u>itself</u> was assigned.

Now you can use the erase and kill characters by using the -INTERACTIVE switch in the RESET statement in your program. For example:

19.1

```
VAR

I, J : INTEGER;

BEGIN

RESET(INPUT, '-INTERACTIVE');

READLN(I);

READLN(J)

END.
```

The -INTERACTIVE switch is a Prime extension. When this switch is used, you can erase or kill anything on the current line — that is, <u>before</u> you enter a carriage return. The word -INTERACTIVE must be enclosed in single quotes.

<u>Caution</u>

You can only use READLNs with the -INTERACTIVE switch. Do not use READs. A READ will not work with -INTERACTIVE because a READ, by definition, still assigns a character as soon as it is typed at the terminal, even <u>before</u> the carriage return is hit. An attempt to use READs will generate an error message at runtime.

The RESET statement opens a PRIMOS data file for reading. RESET is usually used to open input data files; however, there are special cases, such as the example above, where RESET is used to manipulate input from the terminal. (RESET is fully discussed later in this chapter.)

The word INPUT in the RESET statement is a standard Pascal textfile identifier. -INTERACTIVE can only be used with the file INPUT. (For more information on the special functions of the file types INPUT and OUTPUT in Prime Pascal, see Chapter 6 and the discussion on data input files later in this chapter.)

<u>How to Turn the -INTERACTIVE Switch Off</u>: Since the -INTERACTIVE feature is a switch, you can turn it on or off within a program. If you want to turn the -INTERACTIVE feature off use the -TTY feature in another RESET statement. For example:

VAR A, B, C, D : INTEGER; BEGIN RESET(INPUT, '-INTERACTIVE'); READLN(A); READLN(B); RESET(INPUT, '-TTY'); READ(C); READ(D) END.

Use of -TTY lets you go back to inputting data from the terminal in the "normal" way, <u>without</u> the use of Prime's erase and kill characters. The -TTY switch must be used only with the standard file INPUT. (For information on the other uses of -TTY, see the discussion on input data files later in this chapter.)

Prime's -INTERACTIVE extension differs from standard Pascal in the following ways:

- There is no such feature in standard Pascal.
- READs are not allowed when using -INTERACTIVE.

- In standard Pascal, assignments are supposed to be done when a character is typed at the terminal. With the -INTERACTIVE switch, assignments are done only after the carriage return is hit.
- The erase and kill characters are given special meaning. In standard Pascal, the carriage return is the only special character.

INPUTTING AND OUTPUTTING DATA WITH PRIMOS FILES

In Prime Pascal, data can be input from an input data file. Similarly, the computer can output data to an output data file. These data files are PRIMOS files, similar to the PRIMOS file that contains your program. These PRIMOS files can be placed in any directory that you wish.

Upon execution of your program, the computer opens input and output files, retrieves the data from the input file, performs operations using that data, outputs results into an output file, and closes the input and output files.

<u>Note</u>

If you do not use input and output files, data will be input from and output to the terminal by default.

CREATING AND USING INPUT DATA FILES

When you want to place data in a file to be read and operated on by a program, you can create a new PRIMOS file and type your data into that file, using Prime's line editor, ED, or Prime's screen editor, EMACS. (See the <u>New User's Guide to EDITOR and RUNOFF</u>, the <u>EMACS Primer</u>, or the <u>EMACS Reference Guide</u>.)

Once your data has been typed into the file, you would name the file, as you would name any PRIMOS file.

Opening the Input File

In your program, you must tell the computer that the data your program needs is located in a PRIMOS data file. This is called <u>opening</u> the input file. All input files are opened with Pascal's RESET procedure.

19.1

When INPUT is used with a data file, the name of the file must be given as the second parameter in the RESET procedure, as shown above.

If a file is not specified in a READ or READLN statement, the standard textfile INFUT is assumed. For example, the following have the same effect, whether the standard textfile INFUT is a data file or the terminal:

READ(INPUT, A);

READ(A);

For more information on INPUT, see Chapter 6.

Switching from Standard INPUT File to Terminal

If you open an input data file with the standard textfile INPUT, and want to switch to inputting data from the terminal, use the -TTY switch in another RESET procedure. For example:

VAR A, B : INTEGER; INFILE : FILE OF CHAR; BEGIN RESET(INFUT, 'INDATA'); READLN(INFUT, A); RESET(INFUT, '-TTY'); READ(B) END.

18.3

The value of A will be read from an input file named INDATA, and the value of B will be read from the terminal. The standard file INPUT is the first parameter with -TTY. The -TTY switch must be enclosed in single quotes.

The -TTY switch also works with REWRITE and the standard textfile OUTPUT.

CREATING AND USING OUTPUT DATA FILES

When you want to write data out to an output file, simply open the file and name it using the REWRITE procedure.

The REWRITE Procedure

The format of the REWRITE procedure statement is:

REWRITE(file, 'filename');

The first parameter <u>file</u> is a Pascal file variable of a FILE type that is associated with the output file. The second parameter, '<u>filename</u>' is the actual name of the PRIMOS file. This name must be enclosed in single quotes. The inclusion of the second parameter is a Prime extension.

You do not have to create a PRIMOS output file beforehand. The REWRITE procedure will create a PRIMOS file for you upon execution. For example:

```
PROGRAM Writeout;
VAR
A, B, C : INTEGER;
OUTFILE : FILE OF CHAR;
BEGIN
READLN(A);
READLN(B);
C := A + B;
REWRITE(OUTFILE, 'OUTDATA');
WRITELN(C);
CLOSE(OUTFILE)
END.
```

OUTFILE is declared as FILE OF CHAR. A and B are read from the terminal. REWRITE creates a PRIMOS file named OUTDATA in your directory. The value of C is written out to the new file, and the file is closed with CLOSE. (The CLOSE procedure is discussed later in this chapter.)

The second parameter 'filename' can also be a pathname. For example:

REWRITE (OUTFILE, 'PAUL>HOMEWORK>OUTDATA');

An output file called OUTDATA will be created in the subdirectory HOMEWORK within the directory PAUL.

<u>Note</u>

Be sure to find out what your directory access rights are at your installation.

The use of EOF, as well as RESET, GET, REWRITE, and PUT is illustrated in the following example:

```
VAR
INFILE, OUTFILE : TEXT;
BEGIN
RESET(INFILE, 'INDATA');
REWRITE(OUTFILE, 'OUTDATA');
WHILE NOT EOF(INFILE) DO
BEGIN
OUTFILE^ := INFILE^;
PUT(OUTFILE);
GET(INFILE);
END;
CLOSE(INFILE); {The CLOSE procedure is discussed at the end}
CLOSE(OUTFILE) {of this chapter.}
END.
```

The EOLN Function: The function EOLN tests for an end-of-line condition in a textfile. It has the form:

EOLN(file)

This function is true if the buffer variable file corresponds to the position of a line separator marking the end of the current line. The line separator is the ASCII character LF (Line feed), which is a carriage return. FOLN is applied to the standard textfile INPUT, if the parameter <u>file</u> is omitted, whether INPUT is a data file or the terminal.

Auxiliary Procedures

There are two auxiliary procedures that manipulate I/O in Prime Pascal - PAGE and CLOSE. The CLOSE procedure is a Prime extension.

The PAGE Procedure: The form of the PAGE procedure is:

PAGE(file)

The PAGE procedure generates a skip to the top of a new page before the next line of the output textfile <u>file</u> is written. If the single parameter <u>file</u> is omitted, then this procedure is applied to data that is written out to the standard textfile OUTPUT by default, whether OUTPUT is a data file or the terminal.

\$

UPD4303-192

- -----

For example:

WRITELN('Page Test');
WRITELN('Page 1');
PAGE;
WRITELN('Page 2');

The CLOSE Procedure: All input and output data files must be explicitly closed using the CLOSE procedure. Otherwise they will remain open after the program terminates.

The form of the CLOSE procedure is:

CLOSE (file);

The CLOSE procedure is a Prime extension to standard Pascal.

For example:

```
VAR

Fyle: TEXT;

BEGIN

REWRITE(Fyle, 'FYLE');

WRITELN(Fyle, 'ABC');

WRITELN(Fyle, 'DEF');

CLOSE(Fyle)

END.
```

11 Standard Functions

A standard function, denoted by a standard identifier, is a built-in function supplied by the Pascal language. There are four types of standard functions — arithmetic, transfer, ordinal, and BOOLEAN.

ARITHMETIC FUNCTIONS

- ABS(X) Computes the absolute value of X. The type of X must be INTEGER, LONGINTEGER, REAL, or LONGREAL. The type 19.1 of the result is the same as that of X.
- SQR(X) Computes the square of X. X and the result will be of the same data type: INTEGER, LONGINTEGER, REAL, or LONGREAL. [19.1]

<u>Note</u>

For the following <u>arithmetic</u> functions, the type of X must be INTEGER, LONGINTEGER, REAL, or LONGREAL. The type of result is always REAL or LONGREAL.

- SIN(X) Computes the sine of X.
- COS(X) Computes the cosine of X.

19.1

DOC4303-191

- EXP(X) Computes the value of the base of natural logarithms raised to the power X. This is exponential function (e^{x}) .
- LN(X) Computes the natural logarithm of X. X must be greater than zero.
- SQRT(X) Computes the non-negative square root of X. X must be non-negative.
- ARCIAN(X) Computes the value, in radians, of the arctangent of X.

TRANSFER FUNCTIONS

19.1 TRUNC(X) Truncates a real number into an integer. X must be of type REAL or LONGREAL. The result is of type INTEGER or LONGINTEGER. If X is positive then the result is the greatest integer less than or equal to X; otherwise it is the least integer greater than or equal to X. Examples:

TRUNC(3.7) yields 3

TRUNC(-3.7) yields -3

19.1 ROUND(X) Rounds a real number to the nearest integer. X must be of type REAL or LONGREAL. The result, which is of type INTEGER or LONGINTEGER, is the value X rounded. That is, if X is positive, ROUND(X) is equivalent to TRUNC(X + 0.5); otherwise ROUND(X) is equivalent to TRUNC(X - 0.5). Examples:

ROUND(3.7) yields 4

ROUND(-3.7) yields -4

ROUND(3.2) yields 3

ROUND(-3.2) yields -3

<u>Note</u>

Be careful when the result of your TRUNC or ROUND function is of an INTEGER type. You can assign an INTEGER value to a LONGINTEGER variable without any possible errors, but when you attempt to assign a LONGINTEGER value to an INTEGER variable an error is generated. This also applies to REAL and LONGREAL. (See Chapter 6 for more information on LONGINTEGER and LONGREAL.)

BOOLEAN FUNCTIONS

- ODD(X) X must be of type INTEGER or LONGINTEGER. The result 19.1 is TRUE if X is odd and FALSE otherwise.
- EOF(F) F is the file variable of an input file. This function returns the value TRUE if an end-of-file condition exists for F and FALSE otherwise. It applies to the standard textfile INPUT if the argument F is omitted.
- EOLN(F) F is the file variable of an input textfile. This function returns the value TRUE if the end of the current line is reached and FALSE otherwise. It applies to the standard textfile INPUT if F is omitted.

\$ and _ in identifiers (Chapter 4)

The & and ! integer operators (Chapter 7)

The OIHERWISE keyword (Chapter 8)

The EXTERN attribute (Chapter 9)

The {\$E} compiler switch (Chapters 2 and 9)

The {\$A} compiler switch (Chapter 2)

The {\$L} compiler switch (Chapter 2)

The {\$P} compiler switch (Chapter 2)

Dollar signs and underscores are allowed in identifiers in Prime Pascal. However, the underscore cannot be the first character.

Prime's integer operators & and ! perform Boolean AND and OR operations respectively on decimal integer and longinteger numbers.

Prime's OTHERWISE keyword can be used at the bottom of a CASE statement to execute an alternative statement, or group of statements, if no statement in the list of CASE statements has been selected.

When an external, separately compiled subprogram is declared in Prime Pascal, it must be declared with the word EXTERN at the end of the declaration heading.

External Pascal subprograms can be separately compiled by including the {\$E+} at the beginning of the subprogram file. This switch can also be used in the calling program's variable declarations so that the variables can be referenced by the external subprograms.

The {\$A} switch controls the generation of code used to perform array bounds checking at runtime.

The {\$L} switch controls the printing of source lines to the listing file at compile time, if -LISTING was specified.

The {\$P} switch controls page breaks or page "ejects" in the listing file.

18.3

The second parameter 'filename' in RESET and REWRITE procedures (Chapter 10)

The CLOSE procedure (Chapter 10)

The standard data files INPUT and OUTPUT (Chapters 6 and 10) When input or output data files are used, your RESET and REWRITE procedures, which open the files, should have as their second parameter the name of the PRIMOS file that has to be opened for reading or writing. This filename must be enclosed in single quotes. The first parameter is a variable declared as a FILE type, which is associated with the second parameter, 'filename'.

The CLOSE procedure must be used to close an input or output data file after it has been opened with RESET or REWRITE.

The standard data files INPUT and OUTPUT, when used in a RESET or REWRITE procedure without the second 'filename' parameter will automatically default to I/O to and from the terminal. If a file is not specified in a READ or READLN statement, the standard textfile INPUT is assumed, whether the standard textfile INPUT is a file or the terminal. This also applies to WRITE, WRITELN, and the standard textfile CUTPUT.

B Data Formats

OVERVIEW

This appendix illustrates how values of Prime Pascal data types are represented in storage. For more information on all of the data types, see Chapter 6. In Prime Pascal, a word consists of 16 bits.

Prime Pascal supports the following data types:

Scalar Data Types

INTEGER LONGINTEGER (Prime extension) Subrange	19.1
REAL	I
LONGREAL (Prime extension)	19.1
CHAR	
BOOLEAN	
Enumerated	

Structured Data Types

ARRAY RECORD SET FILE Pointer Data Type

Pointer

INTEGER TYPE DATA

Integers are 16-bit (one word) twos-complement, fixed-point whole binary numbers. Integers can hold values within the range -32768 to +32767. Bit 1 is the sign bit, which indicates whether the integer is positive or negative. Bits 2-16 are the integer itself.



LONGINTEGER TYPE DATA

Longintegers are 32-bit (two-word) twos-complement, fixed-point whole binary numbers that hold values within the range -2147483648 to +2147483647. Bit 1 is the sign bit, which indicates whether the longinteger is positive or negative, and bits 2-32 are the longinteger itself. The LONGINTEGER type is a Prime extension.







DOC4303-191

POINTER TYPE DATA

Each value of a pointer type variable is the actual address of the data to which each variable is pointing. Therefore the storage area for each pointer variable contains an address.

A pointer is represented in storage by 48 bits (three words). Specifically:

- Bit 1 is the fault code, which determines if the desired data is found or not found.
- Bits 2 and 3 contain the ring number of the data that is being pointed to.
- Bit 4 is the extension bit, which indicates whether the pointer contains a bit offset (three-word pointer) or doesn't contain a bit offset (two-word pointer).
- Bits 5-16 contain the segment number of the data.
- Bits 17-32 contain the word number of the data within that segment.
- Bits 33-36 are the bit offset, which allows the pointer to point to any bit in memory.
- Bits 37-48 are reserved for future storage.



D Interfacing Pascal to Other Languages

OVERVIEW

This appendix offers guidelines for interfacing Pascal data types with compatible data types of other Prime languages.

The key to interfacing compatible data types is storage representation. For example, a Pascal INTEGER value and a PL/I Subset G Fixed Bin(15) value are both stored as 16-bit (one-word) whole binary numbers. Therefore, an INTEGER value can be passed to a Fixed Bin(15) value and vice versa. In order to interface Pascal to another language successfully, you should be familiar with how Prime Pascal data types are represented in storage. (See Appendix B.) You should also be familiar with the other Prime language and how data types of that language are represented in storage.

Table D-1 matches the compatibility of Prime Pascal data types with the data types of Prime's PL/I Subset G, FORTRAN 77, FORTRAN IV, COBOL, and BASIC/VM. The leftmost column lists the generic storage unit, which is measured in bits, bytes, or words for each data type. Each storage unit matches the data types listed to the right on the same row. Following Table D-1, this appendix briefly discusses data type compatibility and includes several program examples.

For more information on interfacing Pascal to other languages, as well as calling Prime's standard subroutines, see the <u>Subroutines Reference</u> <u>Guide</u>.

Table D-1 Compatible Data Types

GENERIC UNIT/PMA	BASIC/VM	COBOL	FORTRAN IV	FORTRAN 77	PASCAL	PL/I SUBSET G
l bit	—	-				Bit Bit(1)
16 bits (one word)	INT	COMP	INTEGER INTEGER*2 LOGICAL	INTEGER*2 LOGICAL*2	INTEGER BOOLEAN ENUMERATED	Fixed Bin Fixed Bin(15)
32 bits (two words)	INT*4	_	INTEGER*4	INTEGER INTEGER*4 LOGICAL LOGICAL*4	LONGINTEGER	Fixed Bin(31)
64 bits (four words)		_	-	—		
32-bit Float single precision	REAL	1	REAL REAL*4	REAL REAL*4	REAL	Float Binary Float Bin(23)
64-bit Float double precision	REAL*8		REAL*8	REAL*8	LONGREAL	Float Bin(47)
Byte string (Max. 32767)	INT	DISPLAY(5) PIC A(n) PIC 9(n) PIC X(n)	INTEGER	CHARACTER *n	CHAR ARRAY [1n] OF CHAR	Char(n)
Varying character string		-		_	*	Char(n) Varying
48-bits (three words)					^ <type></type>	Pointer
256 bits					SET	Bit(256)

- Not available.

* See Subroutines Reference Guide

Index

\$ and ! integer operators (Prime extension) 7-7, A-3 \$ and _ in identifiers (Prime extension) 4-8, A-3 -32I compiler option 2-13 -64V compiler option 2-13 A compiler switch (Prime extension) 2-16, A-3 Abbreviations, compile option 2-13 to 2-15 ABS function 6-4, 11-1 Actual parameters 4-4, 9-1, 9-2 Allocating dynamic variables 6-32, 6-33 AND operator 7-6 ANSI standard 1-4, 2-12, 4-4, 6-8, C-1 to C-6

ARCTAN function 6-6, 11-2 Arithmetic operators 7-2, 7-3 ARRAY OF CHAR 6-17 to 6-19 ARRAY OF CHAR, (Prime extension) 6-18, 6-19 Array storage 6-16, B-5 ARRAY storage format B-5 ARRAY type 6-14 to 6-20, B-5 Arrays: external 6-16 multidimensional 6-19, 6-20, B-5 ASCII character set 4-4, 6-8 to 6-10, C-1 to C-6 Assignment compatibility 8-2, 8-3 Assignment statement 8-2, 8-3 Auxiliary procedures: CLOSE (Prime extension) 10-24, A-4

PAGE 10-23, 10-24 BEGIN and END keywords 4-7, 8-4, 8-5 -BIG and -NOBIG compiler options 2-8 Binary (object) file 2-1, 2-4, 2-5, 2-8, 3-1 to 3-7 -BINARY compiler option 2-8 Blanks 4-11, 4-12 Block: declaration part 5-4 to 5-9 definition 4-2description 5-3, 5-4 executable part 5-9 to 5-11 illustration 4-3 BOOLEAN operators 7-6 BOOLEAN storage format B-4 BOOLEAN type 6-8, B-4 Boundary-spanning object code 9-5 Call, recursive 9-19 to 9-22 Calling subprograms: external 9-16 functions 9-14 procedures 8-3, 8-4, 9-10, 9-11 Cardinality, data type 6-1 CASE and variant records 6-23 to 6-25 CASE statement 8-11 to 8-14 Changing compiler option defaults 2-6 Changing erase and kill characters 10-4

CHAR storage format B-4 CHAR type 6-8 to 6-10, B-4 Character set (See ASCII character set) Character string constants 4-11, 6-9 Character strings 4-11, 6-17 to 6-19, D-5, D-6 CHR function 6-8 to 6-10, 11-3CLOSE procedure (Prime 6-30, 10-7, 10-12, extension) 10-24, A-4 Closing data files 6-30, 10-7, 10-12, 10-24, A-4 Code, object: boundary-spanning 9-5 ordinary 9-5 Collating sequence 6-8, C-3 to C-6 Command files 3-8 Command level, PRIMOS 2-2, 2-3, 2-5, 3-2 to 3-8 Command line: options 2-1, 2-2, 2-6 to 2-15 Pascal compiler 2-2 Comments 4-11, 4-12 Compatibility with other languages 1-6, 9-1, 9-15 to 9-18, D-1 to D-8 Compile-time errors 2-1 to 2-4 Compiler switches: A switch 2-16, A-3 E switch 2-9, 2-17, 6-16, 6-22, 9-16 to 9-18, A-3 L switch 2-16, A-3 overview 2-16, 4-12 P switch 2-17, A-3

Compiler: error messages 1-4, 2-1 to 2 - 4filename conventions 2-4, 2-5, 3-2 to 3-8 invoking 2-2 option abbreviations 2-13 to 2-15 options 2-1, 2-2, 2-6 to 2-15 PASCAL command 2-2 switches 2-9, 2-16, 2-17, 4-12, 6-16, 6-22, 9-16 to 9-18, A-3 Compiling programs 2-1 to 2-17 Compound statement 8-4, 8-5 Conditional statements: CASE 8-11 to 8-14 IF 8-10, 8-11 CONST declaration 5-6 Constants: BOOLEAN 4-9, 6-8 CHAR 4-11, 6-9 character string 4-11, 6-9 declared 5-6 enumerated 6-11, 6-12 INTEGER and LONGINTEGER 4-10, 6-4, 6-5 MAXINT 4-9, 6-3 NIL 6-32, 6-33, 7-1 numeric 4-8, 4-10 REAL and LONGREAL 4-10, 6-6,6-7 standard 4 - 96-13, 6-14 subrange Control (nonprintable) characters 6-8, 11-3, 11-4, C-5, C-6 Control statements: 8-11 to 8-14 CASE 8-8, 8-9 FOR GOTO 8-14, 8-15 IF 8-10, 8-11 8-5, 8-7, 8-9 to 8-11 nested REPEAT 8-6 WHILE 8-7, 8-8

Control-C end-of-file marker 10 - 22Conventions, filename (See Filename conventions) \cos function 6-6, 11-1 CPL files 3-8 Creating data files: input 10-6 to 10-11 output 10-11 to 10-14 Creating dynamic variables 6-32, 6-33 Data file I/O 6-30, 10-6 to 10 - 24Data files: closing 6-30, 10-7, 10-12, 10-24, A-4 10-6 to 10-14 creating 6-30, 10-6 to 10-14 opening Data format (See Storage format) Data type cardinality 6-1 Data types: ARRAY 6-14 to 6-10, B-5 BOOLEAN 6-8, B-4CHAR 6-8 to 6-10, B-4 enumerated 6-10 to 6-12, B-4 FILE 6-27 to 6-31, 10-6 to 10-24, B-6, B-7 illustration 6-2 6-3, 6-4, B-2 INTEGER interfacing with other languages D-1 to D-8 LONGINTEGER 6-4, 6-5, A-1,B-2 LONGREAL 6-7, A-1, B-3 6-1 overview pointer 6-31 to 6-33, B-8 REAL 6-6, B-3 RECORD 6-20 to 6-25, B-5 SET 6-25 to 6-27, B-5 standard scalar 6-2 to 6-10 storage formats B-1 to B-8 structured 6-14 to 6-31 subrange 6-12 to 6-14, B-3

TEXT 6-29, 10-10 user-defined scalar 6-10 to 6 - 14-DEBUG and -NODEBUG compiler options 2-8 Debugger utility 1-5, 2-6 to 2-8, 2-14 Decimal notation 4-10, 6-6, 6-7, 10-21 Declarations: CONST 5-6 description 5-3, 5-4 LABEL 5-4, 5-5 order of (Prime extension) 5-3, A-2 PROCEDURE and FUNCTION 5-9 TYPE 5-6, 5-7 5-7 to 5-9 VAR Default field widths 10 - 20Default options 2 - 6Delimiters, comment 4-12 Designator, function 9-14 Destroying dynamic variables 6-32, 6-33 Directives: 4-9, 9-15, A-3 EXTERN FORWARD 4-9, 9-15 DISPOSE procedure 6-32 DIV operator 6-4, 7-3 Documents related to Pascal 1-4, 1-5 Dollar signs and underscores in identifiers 4-8, A-3 Dynamic allocation procedures: DISPOSE 6-32 NEW 6-32

Dynamic storage 6-32, B-8 6-31 to 6-33 Dynamic variables E compiler switch (Prime extension) 2-9, 2-17, 6-16, 6-22, 9-16 to 9-18, A-3 EDITOR 1-4, 1-5, 10-6, 10-8 to 10 - 10Elements, Pascal language 4-1 to 4-12 EMACS editor 1-5, 10-6, 10-8 to 10-10 Empty record 6-25 Empty set 6-26 Empty statement 8-5 END and BEGIN keywords 4-7. 8-4, 8-5 End of File (EOF) condition 10-22, 10-23 End of Line (EOLN) condition 10 - 23Enumerated storage format B-4 Enumerated type 6-10 to 6-12, B-4 EOF function 10-22, 11-5 EOLN function 10-23, 11-5 Erase and kill characters: changing 10-4 overview 10 - 4using on terminal input 10 - 4to 10-6 with -INTERACTIVE switch 10 - 4to 10-6 Erasing terminal input 10-4 to 10-6

Error messages: compile time 2-1 to 2-4 for %INCLUDE files 2-3, 2-4 for -INTERACTIVE switch 10-5 for data types 6-5, 6-7, 6-12 to 6-14, 6-19, 6-27, 11-2 for external subprograms 9-18 for identifiers 4-8, A-5 for keyword PACKED 6-14, A-5 for labels 5-5 for non-ANSI standard 2-12 for PACK and UNPACK 9-12, A-5 for parameters 9-4 for standard functions 6-12, 11-2 format of 2 - 2in listing file 2-10 loading 3-3, 3-6 overview 1 - 4runtime 2-11, 3-3, 3-6, 6-12, 10 - 5severity codes 2-3 significance 2 - 3suppressing of 2-8, 2-12 -ERRITY and -NOERRITY compiler options 2-8 Executable (SEG) file 2-5, 3-2 to 3-8 Executable block part 5-9 to 5-11 Executable statements 5-9 to 5-11, 8-1 to 8-16 EXECUTE (load subprocessor) command 3 - 8Executing programs 3-7, 3-8 EXP function 6-6, 11-2-EXPLIST and -NOEXPLIST compiler options 2-9 Exponents 4-10, 6-6, 6-7, 10-20, B-3 Expressions 7-1 to 7-8

Extensions (See Prime extensions) EXTERN (Prime extension) directive 9-15, A-3 -EXTERNAL and -NOEXTERNAL compiler options 2-9, 9-17 External arrays 6-16 External procedures and functions (See External subprograms) External records 6-22 External subprograms: calling 9-16 declaring 9-15, 9-16 EXTERN (Prime extension) directive 9-15, 9-16, A-3 from libraries 9-19 overview 9-1, 9-15 written in other languages 9-18, D-1 to D-8 9-16 to written in Pascal 9 - 18External variables 9-17 Field widths 10-18 to 10-22 Fields, variant 6-23 to 6-25 File control block B-6, B-7 File I/O 6-30, 10-6 to 10-24 FILE OF CHAR 6-28 to 6-31, 10-8 to 10-12 FILE OF CHAR, reading and writing of 10-9, A-5 FILE OF INTEGER 6-28, 6-29, 10-8, 10-9 FILE OF REAL 6-28, 6-29, 10-8, 10-9 File storage B-6, B-7

FILE storage format B-6, B-7 FILE type 6-27 to 6-31, 10-6 to 10-24, B-5 File variables 6-27 to 6-31, 10-6 to 10-24 File-handling functions: 10-22, 11-5 EOF 10-23, 11-5 EOLN File-handling procedures: CLOSE (Prime extension) 6-30, 10-7, 10-12, A-4, 10-2 10-15 GET 10-23, 10-24 PAGE PUT 10-18 READ 10-15 to 10-17 READLN 10-17 10-6 to 10-10 RESET REWRITE 10-11 to 10-13 WRITE 10-18 to 10-21WRITELN 10-22 Filename conventions: overview 2-4 2-5, 3-4 to 3-7 prefix suffix 2-5, 3-2 to 3-4, 3-7 table 3-7 Filename in RESET and REWRITE (Prime extension) 10-6 to 10-14, A-4 Files, data (See Data files) Files: (See also Textfiles) 2-3, 2-4, 5-10, %INCLUDE 5-11, A-2 closing 6-30, 10-7, 10-12, 10-23, A-4 of CHAR 6-28 to 6-31, 10-8 to 10-12 of INTEGER 6-28, 6-29, 10-8, 10-9 of REAL 6-28, 6-29, 10-8, 10-9 opening 6-30, 10-6 to 10-14 PRIMOS input/output 6-27 to 6-31, 9-16 to 9-18, 10-1, 10-5 to 10-24standard INPUT 4-9, 6-31,

10-5, 10-10, 10-11, 10-16, 10-17, A-4, 10-22, standard OUTPUT 4-9, 6-31, 10-13, 10-14, 10-18, 10-22, 10-23, A-4 storage of data B-6, B-7 Fixed variant record part 6-23 FOR statement 8-8, 8-9 Formal parameters 4-4, 9-2, 9-3 Format, line 4-11 FORWARD directive 9-15 Forward procedures and functions 9-14, 9-15 -FRN and -NOFRN compiler options 2-9 FUNCTION declaration 5-9, 9-13, 9-15 Function designator 9 - 14Functions: (See also Subprograms) declarations 9-13 external 9-15 to 9-19 forward 9-14, 9-15 9-13, 9-15 heading I/O 10-14 to 10-24 invoking 9-14 overview 9-1, 9-12 recursive 9-19 to 9-22 standard 4-9, 6-4 to 6-12, 9-14, 10-1, 10-15, 10-22, 10-23, 11-1 to 11-5 GET procedure 10-15 Global: definition 4-2, 5-8, 9-10 external variables 9-17 illustration 4-3 GOIO statement 5-5, 8-14, 8-15

Second Edition

Graphic (printable) characters 6-8 to 6-10, 11-3, 11-4, C-5, C-6 Heading: external 9-15 function 9-13, 9-15 procedure 9-9, 9-15 program 5-1 to 5-3 I/O at terminal 6-29, 6-30, 10-2 to 10-6 I/O procedures and functions: CLOSE (Prime extension) 10 - 24EOF 10-22, 11-5 10-23, 11-5 EOLN 10-15 GET 10-1, 10-14 overview PAGE 10-23, 10-24 PUT 10-18 READ 10-15 to 10-17 10-17 READLN RESET 10-6 to 10-10 REWRITE 10-11 to 10-13 WRITE 10-18 to 10-21 WRITELN 10-22 Identifier length (Prime restriction) 4-7, A-5 Identifiers: dollar signs and underscores in (Prime extension) 4-8, A-3 standard 4-8, 4-9 user-defined 4-8 IF statement 8-10, 8-11 IN operator 6-27, 7-4 %INCLUDE files 2-3, 2-4, 5-10, 5-11, A-2 Index, array 6-14, 6-15 INPUT and OUTPUT, use of 6-31, 10-11, 10-13, A-4 Input and output: overview 10-1 procedures and functions 10-14 to 10-24 to and from data files 6-30,

10-6 to 10-24 to and from terminal 6-30, 10-2 to 10-6 -INPUT compiler option 2-9 INPUT, standard textfile 4-9, 6-31, 10-5, 10-10, 10-11, 10-16, 10-17, 10-22, 10-23, A-4 Integer (Prime extension) operators 7-7, A-3 INTEGER storage format B-2 INTEGER type 6-3, 6-4, B-2 -INTERACTIVE switch (Prime extension) 10-4 to 10-6, A-2, B--6 Interfacing Pascal to other languages: ARRAY OF CHAR interface D-5 BOOLEAN interface D-3 CHAR interface D-5 compatibility table D-2 enumerated interface D-3 INTEGER interface D-3 LONGINTEGER interface D-4 LONGREAL interface D-5 overview 1-6, 9-1, 9-15 to 9-18, D-1, D-2 pointer interface D--6 REAL interface D-4 RECORD interface D-7 to D-8 SET interface D-7 Internal representations (See Storage format) Invoking external subprograms 9-16 Invoking functions 9-14 Invoking procedures 8-3, 8-4, 9-10, 9-11 Invoking the compiler 2-2 Keywords 4-7
and Kill character (See Erase kill characters) L compiler switch (Prime extension) 2-16, A-3 LABEL declaration 5-4, 5-5 4-1 to 4-12 Language elements Language interfaces 1-6, 9-1, 9-15 to 9-18, D-1 to D-8 Libraries: loading 3-1 to 3-6 Prime system 3-1 to 3-6, 9-19 LIBRARY (load subprocessor) command 3 - 2Library, Pascal 3-1 to 3-6 Line format 4-11 -LISTING compiler option 2 - 10Listing file (See Source listing file) LN function 6-6, 11-1 LOAD (load subprocessor) command 3 - 2-LOAD (SEG option) 3-2 Load subprocessor commands: EXECUTE 3-8 LIBRARY 3-2 3-2 LOAD QUIT 3-2 LOAD utility 1-5, 2-5, 3-1 to 3-8 Loading programs: overview 3-1, 3-2 3-4 to 3-6 with prefix method with suffix method 3-3, 3-4 Loading subprograms 3-1, 3-3 to 3-6, 9-18

Local: definition 4-4, 5-8, 9-10 external variables 9-17 9-19 recursive variables LONGINTEGER storage format (Prime extension) B-2 LONGINTEGER type (Prime extension) 6-4, 6-5, A-1, B-2 LONGREAL storage format (Prime extension) B-3 LONGREAL type (Prime extension) 6-7, A-1, B-2 -MAP and -NO_MAP compiler options 2-10 4-9, 6-3 MAXINT Messages: end-of-compilation 2-2, 2-3 error (See Error messages) MOD operator 6-4, 7-3Multidimensional arrays 6-19, 6-20, B-5 Nested statements: defined 8-5 FOR 8-9 IF 8-10, 8-11 WHILE 8--7 NEW procedure 6-32 6-32, 6-33, 7-1 NIL Non-ANSI standard errors 2 - 12Nonprintable (control) 6-8, 11-3, 11-4, characters C-5, C-6 NOT operator 7-6 Notation: 4-10, 6-6, 6-7, 10-21 decimal scientific 4-10, 6-6, 6-7, 10-20, B-3

Null program 5-4 Numeric constants 4-8, 4-10 Object (binary) file 2-1, 2-4, 2-5, 2-8, 3-1 to 3-7 ODD function 11-5 -OFFSET and -NOOFFSET compiler options 2-10 Opening data files: input 6-30, 10-6 to 10-11 output 6-30, 10-11 to 10-14 RESET 10-6 to 10-10 Operands 7-1, 7-2 Operator precedence 7-7 Operators, arithmetic: 6-4, 6-6, 7-3 * 6-4, 6-6, 7-3 6-4, 6-6, 7-3 + / 6-6, 7-3 DIV 6-4, 7-3 6-4, 7-3 MOD Operators, BOOLEAN: 7-6 AND 7-6 NOT 7-6 OR Operators, integer (Prime extension): 1 7-7 7-7 & Operators, relational: 6-4, 6-8, 6-10, 6-12, 7-4 < <= 6-4, 6-8, 6-10, 6-12, 6-27, 7-4 <> 6-4, 6-8, 6-10, 6-12, 6-27, 7-4 6-4, 6-8, 6-10, 6-12, 6-27, = 7-4 > 6-4, 6-8, 6-10, 6-12, 7-4 >= 6-4, 6-8, 6-10, 6-12, 6-27, 7-4 IN 6-27, 7-4

Operators, SET: 6-26, 7-5 * 6-26, 7-5 6-26, 7-5 + Operators: arithmetic 7-2, 7-3 BOOLEAN 7-6 defined 7-1 integer (Prime extension) 7-7 order of evaluation 7-8 precedence of 7-7 relational 7-3, 7-4 SET 6-26, 6-27, 7-4, 7-5 -OPT1 compiler option 2-11 -OPT3 compiler option 2-11 -OPTIMIZE and -NOOPTIMIZE compiler options 2-11 Optional program heading (Prime extension) 5-1, A-2 Options, compiler: abbreviations 2-13 to 2-15 -BIG and -NOBIG 2-8 -BINARY 2-8 commonly used 2-7 -DEBUG and -NODEBUG 2 - 8defaults 2-6 -ERRITY and -NOERRITY 2 - 82-9 -EXPLIST and -NOEXPLIST -EXTERNAL and -NOEXTERNAL 2**-**9, 9-17 -FRN and -NOFRN 2-9 -INPUT 2-9 -LISTING 2-10 -MAP and -NO_MAP 2 - 10not commonly used 2-7 -OFFSET and -NOOFFSET 2-10 -OPT1 2-11 -OPT3 2-11 -OPTIMIZE and -NOOPTIMIZE 2-11 -PRODUCTION and -NOPRODUCTION 2 - 11-RANGE and -NORANGE 2-11 -SILENT and -NOSILENT 2 - 12-SOURCE 2 - 12-STANDARD and -NOSTANDARD 2-12 -STATISTICS and -NOSTATISTICS

2-12 2-13 --UPCASE -XREF and -NOXREF 2 - 13OR operator 7-6 ORD function 6-10, 6-12, 11-3 Order of declarations (Prime extension) 5-3, A-2 Order of evaluation 7-8 Ordinal values 6-8 to 6-12, 11-3, 11-4 OTHERWISE (Prime extension) clause 8-11, 8-14, 8-15, A-3 OUTPUT, standard textfile 4-9, 6-31, 10-13, 10-14, 10-18, 10-22, 10-23, A-4 P compiler switch (Prime extension) 2-17, A-3 PACK and UNPACK procedures (Prime restrictions) 9-12, A-5 Packed arrays 6-14, 6-17, A-5 PACKED keyword (Prime restriction) 6-14, 6-17, A-5 Page breaks in listing file 2-17, A-3 PAGE procedure 10-23, 10-24 Parameters: 4-4, 9-2 actual array variable 9-5 formal 4-4, 9-2, 9-3 overview 4-4, 9-1, 9-2 procedures and functions passed 9-6 to 9-9 as record variable 9-5 value 9-3 variable 9-3, 9-4 PASCAL command 2-2

Pascal: ANSI standard 1-4, 2-12, 4-4, 6-8, C-1 to C-6 arithmetic operators 7-2, 7-3 ASCII character set 4-4, 6-8 to 6-10, C-1 to C-6 4-11, 4-12 blanks 7-6 BOOLEAN operators 4-11, 6-17 character strings to 6-19, D-5, D-6 comments 4-11, 4-12 2-1 to 2-17 compiler data storage formats B-1 to B--8 6-1 to 6-34 data types expressions 7-1 to 7-8 identifiers 4-7 to 4-9 input and output 10-1, 6-27 to 6-31 instruction books 1-1 integer operators 7-7 4-7 keywords language elements 4-1 to 4-12 language interfaces D-1 to D-8, 1-6, 9-1, 9-15 to 9-18 library 3-1 to 3-6 line format 4-11 numeric constants 4-8 operands 7-1, 7-2 operator precedence 7-7, 7-8 operators 7-2 to 7-8 parameters 9-2 to 9-9 Prime extensions 1-2, A-1 to A-4 Prime Pascal 1-2 1-2, A-5 Prime restrictions procedures and functions 9-1 to 9-22 program structure 5-1 to 5-15 punctuation symbols 4-5, 4-6 related documents 1-4, 1-5 relational operators 7-3, 7-4 separators 4-11 set operators 7–5 standard functions 4-9, 6-4 to 6-12, 9-14, 10-1, 10-15, 10-22, 10-23, standard procedures 9-12, 10-1, 10-7, 10-11, 10-14 to 10 - 24statements 8-1 to 8-16 storage requirements 6-17, 6-22, 6-32, B-1 to B-8, D-1, D-2, 6-16

Pass-by-reference parameters 9-3, 9-4 Pass-by-value parameters 9-3 PMA (See Prime Macro Assembler) Pointer data type 6-31 to 6-33, B-8 Pointer storage format B-8 Precedence of operators 7-7 PRED function 6-10, 6-12, 11-4 Prefix: executing file 3-7 filename conventions 2-5, 3-4 to 3-7 loading procedure 3-4 to 3-7Prime extensions to standard Pascal: \$ and _ in identifiers 4-8, A-3 %INCLUDE files 2-3, 2-4, 5-10, 5-11, A-2 A compiler switch 2-16, A-3 ARRAY OF CHAR enhancement 6-18, 6-19, A-2 CLOSE procedure 6-30, 10-7, 10-12, 10-23, comment delimiters /* */ 4-11, A-2 E compiler switch 2-9, 2-17, 6-16, 6-22, EXTERN directive 9-15, A-3 Filename in RESET and REWRITE 10-6 to, 10-24, A-4 -INTERACTIVE switch 10-4 to 10-6, A-2, B-6 L compiler switch 2-16, A-3 LONGINTEGER type 6-4, 6-5,A-1 LONGREAL type 6-7, A-1, B-3 optional program heading 5-1, A-2 order of declarations 5-3, A-2 OTHERWISE clause 8-11, 8-14, 8-15, A-3 P compiler switch 2-17, A-3 -TTY switch 10-5, 10-11,

10-14, A-2 & and ! integer operators 7-7, A-3 Prime Macro Assembler 1-6, 2-9, 9-18 Prime Pascal: ASCII character set 6-8 to 6-10, C-1 to C-6 compiler 2-1 to 2-17 defined 1-2 extensions 1-2, A-1 to A-4 3-1 to 3-6, 9-19library related documents 1-4, 1-5 restrictions 1-2, A-5 Prime restrictions to standard Pascal: FILE OF CHAR, reading/writing of 10-9, A-5 identifier length 4-7, A-5 PACK procedure 9-12, A-5 PACKED keyword 6-14, 6-17, A-4 UNPACK procedure 9-12, A-5 Prime: debugging utility 1-5, 2-6 to 2-8, 2-14 documents related to Pascal 1-4, 1-5 filename conventions 2-4, 2-5, 3-2 to 3-8high-level languages 1-4, 1-6, 9-1, 9-15 to 9-18, D-1 to D-8 input and output 10-1 to 10-24 libraries 3-1 to 3-6, 9-19 SEG loading utility 1-5, 3-1 to 3-8, 2-5 subroutines 1-5, 9-19 text editors 1-4, 1-5, 10-6, 10-8 to 10-10 PRIMOS: command level 2-2, 2-3, 2-5, 3-2 to 3-8 data files 6-27 to 6-31, 10-1, 10-5 to 10-24, 9-16 to 9-18 erase and kill characters 10-4 to 10-6

file variables 10-7, 10-8, 10-13 2-2, 9-17 PASCAL command 3-2 SEG command 1-5, 9-19 subroutines user file directories 6-27, 10-6 to 10-8, 10-12 Printable (graphic) characters 6-8 to 6-10, 11-3, 11-4, C-5, C-6 PROCEDURE declarations 5-9, 9-9, 9-10, 9-15 Procedure statement 8-3, 8-4, 9-10, 9-11 Procedures: (See also Subprograms) declarations 5-9, 9-9, 9-10 dynamic allocation 6-32 external 9-15 to 9-19 9-14, 9-15 forward 9-9, 9-15 heading I/0 10-14 to 10-24 8-3, 8-4, 9-10, 9-11 invoking overview 9-1, 9-9 recursive 9-19 to 9-22 standard 4-9, 6-32, 9-12, 10-1, 10-7, 10-11, 10-14 to 10 - 24-PRODUCTION and -NOPRODUCTION compiler options 2-11 Program definition 4-2 Program heading: definition 4-2 description 5-1 to 5-3 Program structure: 5-3 to 5-9 declaration part 5-9 to 5-15 executable part heading 5-1 to 5-3 overview 5-1 Program unit definition 4-2Program, null 5-4

Punctuation symbols 4-5, 4-6 PUT procedure 10-18 QUIT (load subprocessor) command 3 - 2-RANGE and -NORANGE compiler options 2-11 10-15 to 10-17 READ procedure Reading arrays 6-14 to 6-19 READLN procedure 10-17 REAL storage format B-3 REAL type 6-6, B-3 Record storage 6-22, B-5 RECORD storage format B-5 RECORD type 6-20 to 6-25, B-5 Records: 6-25 empty external 6-22 using WITH 6-22, 6-23 variant 6-23 to 6-25Recursive procedures and functions 9-19 to 9-22 Relational operators 7-3, 7-4 REPEAT statement 8-6 Repetitive statements: 8-8, 8-9 FOR REPEAT 8-6 WHILE 8-7, 8-8 RESET procedure 10-6 to 10-10 Restrictions (See Prime restrictions) ROUND function 6-4, 11-2 RUNOFF utility 1-4, 1-5

Runtime errors 2-11, 3-3, 3-6, 6-12, 10-5 Scalar data types: standard 6-2 to 6-10 user-defined 6-10 to 6-14 Scientific notation 4-10, 6-6, 6-7, 10-20, B-3 Scope, definition 4-4 SEG command 3-2 SEG loading utility 1-5, 2-5, 3-1 to 3-8 Separators 4-11 SET operators 6-26, 6-27, 7-4, 7-5 SET storage format B-5 SET type 6-25 to 6-27, B-5 Set, empty 6-26 Severity codes 2-3 -SILENT and -NOSILENT compiler options 2-12 SIN function 6-6, 11-1 -SOURCE compiler option 2-12 Source listing file 2-1, 2-2, 2-4, 2-5, 2-10, 3-4 to 3-7 Source program file 2-1 to 2-6, 3-2, 3-4, 3-6, 3-7 SQR function 6-4, 11-1 SQRT function 6-6, 11-2 -STANDARD and -NOSTANDARD compiler options 2-12 Standard constants 4-9

Standard functions (See Functions) Standard identifiers 4-8, 4-9 Standard procedures (See Procedures) Standard scalar data types 6-2 to 6-10 Standard textfiles (See INPUT and OUTPUT) Statements, declaration (See Declarations) Statements, executable: assignment 8-2, 8-3 compound 8-4, 8-5 control 8-5 to 8-15 empty 8-5 function designator 9-14 overview 8-1 procedure 8-3, 8-4 WITH 8-16 Statements, nested (See Nested statements) Static variables 6-31 -STATISTICS and -NOSTATISTICS compiler options 2-12 Storage format: B-5 ARRAY CHAR B-4 enumerated B-4 FILE B-6, B-7 file control block B-6, B-7 B-2 INTEGER LONGINTEGER (Prime extension) B-2 LONGREAL (Prime extension) B-3 pointer B-8 B-3 REAL RECORD B-5 SET B--5

subrange B-3

Storage: array capacity 6-16 compatibility D-1 to D-8 data formats B-1 to B-8 6-32 dynamic illustrations B-1 to B-8 in other languages D-1 record capacity 6-22 Strings (See Character strings) 6-14 to Structured data types 6-31 Subprograms, external (See External subprograms) Subprograms: (See also Procedures and functions) defined 4-2, 9-1 external 9-15 to 9-19 forward 9-14, 9-15 from libraries 9-19 recursive 9-19 to 9-22 written in other languages 9-18, D-1 to D-8 Subrange storage format B--3 Subrange type 6-12 to 6-14, B-3 Subroutines 1-5, 9-19 SUCC function 6-10 to 6-12, 11-3, 11-4 Suffix: executing file 3-7 filename conventions 2-5, 3-2 to 3-4, 3-7 loading procedure 3-2 to 3-4 Suppressing error messages 2-8, 2 - 12Switches (Prime extension): -INTERACTIVE 10-4 to 10-6, A-2, B-6 -TTY 10-5, 10-11, 10-14, A-2

Switches, compiler (See Compiler switches) 6-29, 6-30, 10-2 Terminal I/O to 10-6 Text editors, Prime 1-4, 1-5, 10-6, 10-8 to 10-10 TEXT type 6-29, 10-10 Textbooks, Pascal instruction 1-1 Textfiles: 6-30, 10-7, 10-12, closing 10-23, A-4 defined 6-29, 10-8 6-30, 10-6 to 10-14 opening standard INPUT 4-9, 6-31, 10-5, 10-10, 10-11, 10-16, 10-17, 10-22, standard OUTPUT 4-9, 6-31, 10-13, 10-14, 10-18, 10-22, 10-23, A-4 TRUNC function 6-4, 11-2 -TTY switch (Prime extension) 10-5, 10-11, 10-14, A-2 TYPE declaration 5-6, 5-7 Types (See Data types) Unconditional GOTO statement 8-14, 8-15 Underscores and dollar signs in identifiers 4-8, A-3 UNPACK and PACK procedures (Prime restrictions) 9-12, A-5 -UPCASE compiler option 2 - 13User-defined identifiers 4-8 User-defined scalar data types 6-10 to 6-14