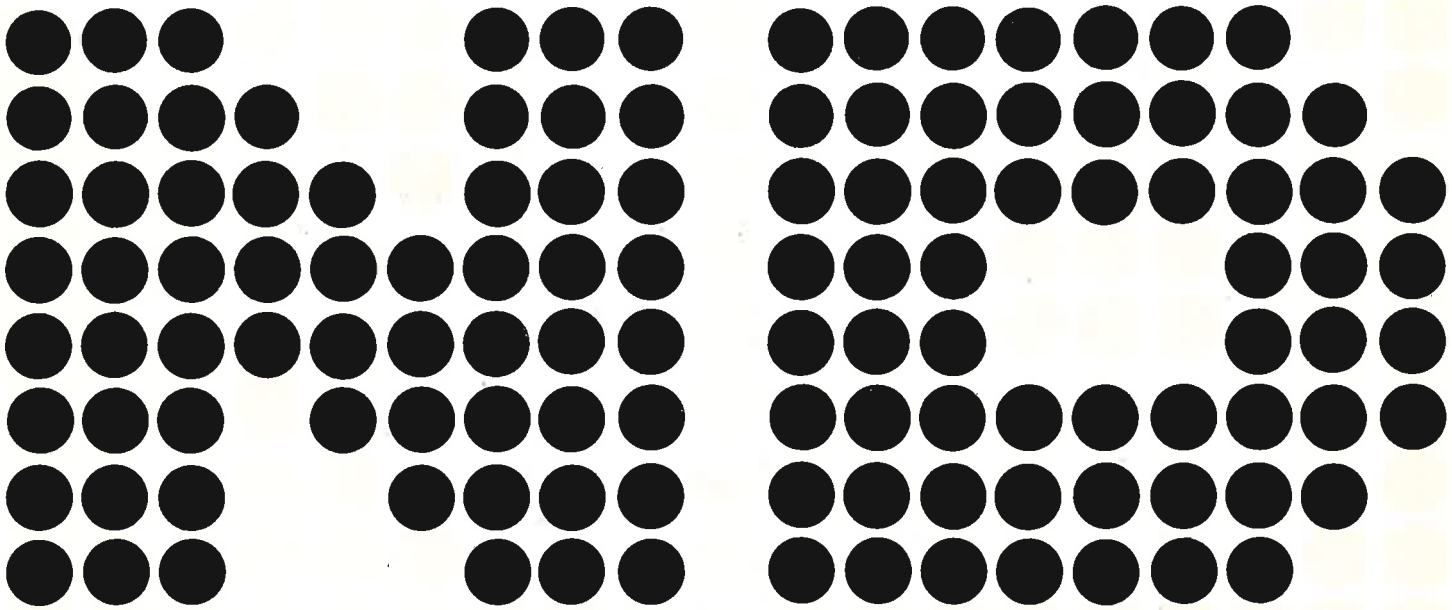
NORD-10 PASCAL

NORSK DATA A.S



NORD-10 PASCAL

REVISION RECORD

Revision	Notes
6/77	Original Printing
6/79	Version two, superseding previous version

Publ. No. ND–60.086.02
June 1979

NORSK DATA A.S.

Lørenveien 57, Postboks 163 Økern, Oslo 5, Norway

PREFACE

The Pascal language was designed in 1971 by Niklaus Wirth. The language design had two principal aims. The first was to make available a language suitable to teach programming as a systematic discipline, the second was to develop implementations of this language which are both reliable and efficient on presently available computers.

The success of this language design proves that Pascal is not "yet another language". Today Pascal has been implemented on almost all computers commonly in use, ranging from the very large computers to mini- and micro-computers. It is the first language which shows ability to bite into domains hitherto reserved for FORTRAN and BASIC. This ability is not only local, but is apparent on a world-wide scale.

This manual contains the information necessary to compile and execute Pascal programs on the NORD-10. It is assumed that the reader is familiar with the Pascal language. The uninitiated reader is referred to the Pascal Report or to an appropriate textbook.

TABLE OF CONTENTS

1 INTRODUCTION

The present chapter gives a general description of the NORD-10 Pascal system. The specific information necessary for the compilation and execution of Pascal programs is found mainly in chapters 2 to 4. Most of the chapters 5 and 6 describe features for the more advanced use of NORD-10 Pascal.

NORD-10 Pascal has been implemented according to the definition in "Niklaus Wirth: The Programming Language Pascal. Revised Report. (1973)". Hereafter this language definition will be referred to as Standard Pascal.

NORD-10 Pascal has one minor restriction and several extensions in relation to Standard Pascal. Especially, several extensions have been introduced to make it convenient to compile and run Pascal programs in a time-sharing environment. Explicit extensions of the Standard Pascal language will be noted as such in this manual. The extensions should be avoided if program exportation is planned or probable.

1.1 The Pascal compiler

The NORD-10 Pascal compiler was developed from the Pascal TRUNK compiler designed at ETH, Zurich. The compiler produces BRF code, which can be loaded by the Nord Relocatable Loader and then executed. A program may refer to separately compiled procedures and functions written in Pascal and FORTRAN.

1.2 The Main Machine Dependent Characteristics

A NORD-10 Pascal program may be run either as a 1-bank or a 2-bank program. As a 1-bank program, all program and data reside within 64K of memory. As a 2-bank program, the program may occupy up to 64K in the instruction bank, and the data occupy up to 64K in the data bank. 1- or 2-bank execution may be selected at compile-time with the B option.

The NORD-10 Pascal system has been constructed to run on NORD-10 computers with both 32-bit and 48-bit floating point arithmetic. Cross compilation is possible by using the compiler R option.

A variable of type set will by default occupy 8 words, i.e. a set can have up to 128 elements. The S option can be used to reduce the number of words occupied by set variables.

1.3 Restrictions

File variables may only be declared in the main program.

The standard procedure DISPOSE is not implemented. Instead, heap space may be deallocated by the help of the procedures MARK and RELEASE (see section 2.5.2).

1.4 The Main Extensions

Variables in the main program can be initialized. There is a convenient syntax for array initialization.

The procedures OPEN and CLOSE enable a program to associate a Pascal file variable with an external file at run-time. OPEN has been implemented such that the actual name of the external file easily can be entered from the terminal running the program.

Random access I/O can be performed with the routines GETRAND and PUTRAND.

2 THE SOURCE PROGRAM

A Pascal source file must contain either

1) A full Pascal program, or
2) One or more procedures and functions.

The source language must be Standard Pascal, with the restrictions and possible extensions described in this manual.

A full Pascal program will compile into an executable object program, while procedures and functions will compile into code that may be loaded together with a full program. A file of procedures and functions to be compiled separately, must be terminated with the character "." (period).

The source file character set must be ASCII, where the lines are separated by the Carriage Return character, and optionally, the Line Feed character. Files produced by QED are acceptable as input to the compiler.

A source input line must not exceed 100 characters. The Pascal compiler will indicate a longer line as an error.

2.1 Identifiers

An identifier may be of any length, but only the first 8 characters are significant. Within an identifier, lower and upper case letters will be treated as distinct, unless the U option is on (see section 2.4.3).

2.2 Key-words

The following are Pascal key-words, and can not be used as identifiers:

Standard Pascal key-words:

and	array	begin	case
const	div	do	downto
else	end	file	for
function	goto	if	in
label	mod	not	of
or	packed	procedure	program
record	repeat	set	then
to	type	until	var
while	with		

Extra key-words in NORD-10 Pascal:

value

A key-word may be written with lower and/or upper case characters. However, within a key-word all lower case characters will be converted to upper case. Thus,

end END End

are all representations of the key-word end.

2.3 Standard Identifiers

Following is a list of the standard identifiers in NORD-10 Pascal. A standard identifier may be thought of as having been defined in a block enclosing the program, and as such, may be redefined. Normally, such redefinition should be avoided, since it easily may lead to confusion.

Standard Pascal standard identifiers:

ABS	ARCTAN	BOOLEAN	CHAR
CHR	COS	DISPOSE*	EOLN
EOF	EXP	FALSE	GET
INPUT	INTEGER	LN	MAXINT
NEW	NIL	ODD	ORD
OUTPUT	PACK	PAGE	PRED
PUT	READ	READLN	REAL
RESET	REWRITE	ROUND	SIN
SQR	SQRT	SUCC	TEXT
TRUE	TRUNC	UNPACK	WRITE
WRITELN			

*DISPOSE is not implemented in NORD-10 Pascal.

Extra standard identifiers in NORD-10 Pascal:

CLOSE	COSH	GETRAND	HALT
MARK	MAXREAL	OPEN	POWER
PUTRAND	RELEASE	SINH	WRITEEOF

All standard identifiers are written with upper case letters.

2.4 Compiler Commands

The source program text may contain commands to the compiler. A command is signalled by the character "$" in position one in a source line. The rest of such a line is

treated as a command to the compiler, and no part of it will be included in the proper program text.

The available compiler commands are

$SET
$RESET
$IFTRUE
$IFFALSE
$ENDIF
$OPTIONS
$INCLUDE

A compiler command may be abbreviated to its shortest unambiguous form.

2.4.1 Conditional compilation

The NORD-10 Pascal compiler may be instructed to skip specified parts of the source text. This may be useful in order to generate different versions of a program from the same source file.

The skipping of source text is steered by flags, which are Boolean variables. The flag identifiers are distinct from the program identifiers, therefore no name conflicts between flag and program identifiers can occur. A flag identifier can have up to 8 characters. No distinction is made between upper and lower case characters.

A flag is given the value TRUE by the command

$SET <flag>

A flag is given the value FALSE by the command

$RESET <flag>

The skipping of source text is effected by the commands

$IFTRUE, $IFFALSE, and $ENDIF

The command

$IFTRUE <flag>

has the effect:
If <flag> has the value TRUE: No effect.
If <flag> has the value FALSE: Skip all source text up to an $ENDIF <flag> with the same flag name.

The command

$IFFALSE <flag>

has the effect:
If <flag> has the value TRUE: Skip all source text up to an $ENDIF <flag> with the same flag name.
If <flag> has the value FALSE: No effect.

If an $IFTRUE or $IFFALSE command has a flag parameter that was not previously defined, it will become defined and given the value FALSE.

Note that when source text is skipped, compiler commands (such as $SET, $IFTRUE etc.) will also be skipped.

2.4.2 Multiple source files

The $INCLUDE-command facilitates insertion in a program of source text from an alternate file. This is useful when a set of programs (within the same project, say) use a common set of type, variable, and procedure definitions. Also, "standard" data structures and procedures for handling problems within a specific problem area, can easily be incorporated in a program with the $INCLUDE-command.

The command

$INCLUDE <filename>

has the effect of switching the input stream from the present input file to <filename>. When end of file on <filename> is reached, the input stream will be switched back to the previous input file. The effect is to insert the text in <filename> at the place where the $INCLUDE-command occurs.

$INCLUDE-commands may be nested to a maximum depth of 4.

2.4.3 Options

There is a set of options that affect the output produced by the Pascal compiler. Each option has a one-letter name. Some of the options are associated with counters. A counter value greater than zero means that the option is on, a value equal to or less than zero means that the option is off. The remaining options are associated with specific values.

A counter option is increased or decreased by one by writing the option name followed by "+" or "-" respectively.

The available options are (counter options are indicated by the character "*"):

Bn Specify n-bank execution of program (n=1 or 2). Default value is n=1.

L* Generate listing. Default value is 1 (on).

M* List generated object code (MAC). Default value is 0 (off).

Rn Specify n-word real (n=2 or 3). Default value is 2 on NORDs with 32-bit floating point arithmetic, and 3 on NORDs with 48-bit arithmetic. A program that is to be cross-compiled, must not contain real constants.

Sn Specify n-word sets (n=1,2,...,8). All variables of type set will then occupy n words, and can have up to 16n elements. The option can only be used once in a program, and must appear before any reference to or use of set is made. n=1 will cause in-line code to be generated for most of the set operations. Default value is n=8 (up to 128 elements).

T* Generate code to check array indices, subrange assignments, pointer values and arithmetic overflow. Turning this option off will make the object program smaller and faster, but also unsafe. Default value is 1 (on).

Note that the NORD hardware does not facilitate checking of overflow on floating point arithmetic operations. Therefore, Pascal can only detect overflow on integer operations. As a special case, attempted floating division by zero is detected.

U* Convert lower case characters outside strings to upper case. Default is 1 (on).

Z* Initialize all variables to zero. Default value is 0 (off).

Options may be set within a comment in the source program. The first character within the comment must be "$". Thereafter, option settings separated by "," may follow. Options may also be set following the $OPTIONS compiler command.

Examples:

(*$M+,S3,T-*) means:

M+ List object code.
S3 Sets will occupy 3 words (up to 48 elements).
T- Do not generate testing instructions.

$OPT Z+,U- means:

Z+ Initialize all variables to zero.
U- Do not convert lower case characters to upper case.

2.4.4 Special symbols

Some of the special symbols in Standard Pascal have one or more alternate representations in NORD-10 Pascal:

Standard Pascal	NORD-10 Pascal
{	(*
}	*)
[	[or (.
]	] or .)
↑	↑ or @

2.5 Extensions in NORD-10 Pascal

This section will describe all extensions in NORD-10 Pascal not related to input/output. Refer to chapter 5 for I/O extensions.

2.5.1 Variable initialization

Variables in the main program may be initialized. Initialization is signalled by the key-word value, and must appear after the var-declarations and before the first procedure or function declaration, or main program begin.

The syntax for initialization is:

```
<variableinit>::=       value {<initialization>;}*
<initialization>::=     <variable> = <val>
<val>::=                <constant> | (<valuelist>)
<valuelist>::=          <aval> { , <aval>}*
<aval>::=               <constant> | <count> * <constant>
<count>::=              <integer constant>
```

Examples:

```
value
  X = 2.55;
  I = 19;
  TABLE = (1,3,2*7,-1,11*0);
  NAME = ('PASCAL  ');
```

Since a string has the type array of CHAR, a string constant must be enclosed in parentheses as shown in the last example.

2.5.2 Standard procedures and functions

SINH and COSH

These real functions calculate the arithmetic functions sinh and cosh respectively.

POWER

POWER is a real function with two parameters x and y which calculates the function x↑y. When y is an integer, x↑y is (in principle) calculated by repeated multiplication. When y is real, x↑y is calculated by the formula x↑y = e↑y·ln(x). Thus, POWER(-1.0,2.0) will give a runtime error, while POWER(-1.0,2) will give the correct result 1.0.

HALT

HALT is a procedure which takes a string parameter. HALT will write this string on the terminal and abort the program.

MARK and RELEASE

The Standard Pascal standard procedure DISPOSE is not implemented in NORD-10 Pascal. Instead, heap space may be deallocated by the help of MARK and RELEASE.

Both procedures take a pointer variable as a parameter. The call MARK(<ptr>) will assign the address of the current heap top to <ptr>. The call RELEASE(<ptr>) will release everything on the heap which is beyond the value of <ptr>.

2.5.3 External procedures and functions

The Pascal library contains a set of external procedures and functions. To use one of these, the procedure or function must be declared as external within the program.

An installation may choose to have a system file containing external declarations for these external procedures and functions. This file may then be included in a program with the $INCLUDE compiler command.

TUSED

External declaration:

```
function TUSED: REAL; extern;
```

TUSED gives the elapsed CPU time in seconds.

TIME and DATE

External declarations:

```
procedure TIME(var hour, min, sec: INTEGER); extern;
procedure DATE(var year, month, day: INTEGER); extern;
```

TIME and DATE give the current time and date respectively.

ECHOM

External declaration:

```
procedure ECHOM(echomode: INTEGER); extern;
```

Executes MON 3 with echomode in the A register. This will define the echo mode for the terminal as specified in the Sintran manual.

BRKM

External declaration:

```
procedure BRKM(breakmode: INTEGER); extern;
```

Executes MON 4 with breakmode in the A register. This will define the break mode for the terminal as specified in the Sintran manual.

ERMSG

External declaration:

```
procedure ERMSG(errorno: INTEGER); extern;
```

Executes MON 64 with errorno in the A register. This will write the Sintran error message corresponding to the given error number to the terminal.

HOLD

External declaration:

```
procedure HOLD(time: REAL); extern;
```

Will suspend execution of the program in <time> seconds. <time> is accurate to 20 milliseconds.

VERSN

External declaration:
```
procedure VERSN(var year, month, day: INTEGER); extern;
```

Will give the date when the executing program was compiled.

RANDOM

External declaration:
```
function RANDOM(var x: REAL): REAL; extern;
```

This function gives a uniformly distributed pseudo random number in the interval <0,1>. Each new value is calculated from the value of the parameter. This new value is also assigned to the parameter variable. Thus, successive calls on RANDOM with the same variable as a parameter, will produce a uniformly distributed pseudo random number stream.

2.5.4 External FORTRAN routines

Separately compiled FORTRAN subroutines may be called from a Pascal program. A FORTRAN routine must be declared in the Pascal program with a procedure or function heading, and a body consisting of the word "FORTRAN". Example:

```
procedure ROUTINE(var x, y: REAL); FORTRAN;
```

Parameters of any type and kind, except Pascal procedure or function names, may be transmitted to the FORTRAN routine; however, no check is made that the parameters are consistent with the formal parameters of the FORTRAN routine. Parameters which are specified as var, or which occupy more than 8 words, are transmitted by reference. Value parameters occupying 8 words or less are transmitted by value.

FORTRAN routines may only be called from one-bank Pascal programs.

When loading modules for a mix of Pascal and FORTRAN programs, the following order must be observed:

1. Pascal main program
2. Pascal and FORTRAN external routines
3. FORTRAN library
4. Pascal library

2.5.5 Miscellaneous extensions

The compiler accepts octal constants. The syntax for an octal constant is

dd*B

where d is an octal digit.

MAXREAL is a standard real constant with a value equal to the largest possible floating point value (approximately 10↑4930 and 10↑76 for 48- and 32-bit floating point numbers, respectively).

2.6 Implementation dependent features

2.6.1 Structured types

Variables of structured types (records and arrays) may be assigned to and compared, provided the variable type is not packed or contain packed variables. Variables of type packed array [...] of CHAR are excepted from this restriction.

2.6.2 Packed structures

Record and array types may be specified as packed. Each single variable will then occupy a minimum number of bits, and several single variables may be packed into one computer word. No single variable will cross word boundaries. Also, a record or an array will always start at a new word boundary.

The use of packed structures will save data space, but may increase execution time significantly.

A variable within a packed structure can not be used as a var parameter to a procedure.

See chapter 5 for information on packed files.

2.6.3 Strings and character arrays

Variables of type array [...] of CHAR will be packed whether packed was specified or not.

In Standard Pascal, a string constant with n characters is automatically given the type packed array [1..n] of CHAR. This inhibits assignment of, or parameter substitution with, a string to a variable or formal of type array [...] of CHAR where the lower bound is different from 1. In NORD-10 Pascal

such assignment or substitution will be legal provided the length of the string is equal to the length of the array.

2.6.4 Formal procedures

A formal procedure may only have value parameters. On entry to a formal procedure, the actual parameters are checked only to see if they occupy the same number of words as the formal parameters. The user is warned that the use of formal procedures with pointer parameters is unsafe.

3 PROGRAM COMPILATION

The Pascal compiler is invoked by the command

```
@PASCAL
```

Initially, the compiler enters into a command processing mode, to enable the user to specify source, list and code files, options etc. The command processor prompts the user to give a new command with the character "$".

The available commands are:

```
HELP
COMPILE
CLEAR
OPTIONS
SET
RESET
EXIT
```

A command may be abbreviated to its shortest unambiguous form.

Note that the SET, RESET, and OPTIONS commands also are available as compiler commands (cfr. section 2.4).

3.1 HELP

The HELP command lists the available commands on the user's terminal (or batch output file). The list includes both the command processor commands and the compiler commands.

3.2 COMPILE

The COMPILE command orders Pascal to compile the specified source file. The present setting of flags and options will be used during the compilation.

The syntax of the COMPILE command is

```
COMPILE <source file>, <list file>, <code file>
```

The parameter list may be omitted, in which case the command processor will ask the user to specify the files one by one.

<source file> contains the program to be compiled.

<list file> is the file on which the listing of the compiled program will be written. The <list file> parameter may be omitted, in which case no listing will be generated.

The listing contains:

in column 1: Source line number (decimal).
in column 2: Relative program and variable addresses (octal).
in column 3: A numbering of the begin-end, repeat-until, and case-end pairs in the program, to indicate the nesting structure of the program. Also, the declaration level for each procedure and function is indicated.
in column 4: The source program.

The listing is divided into pages with a heading on each page containing: version of compiler, date and time of compilation, and page number.

The listing will indicate a language syntax error at the exact spot where it was discovered, together with an error number. If a part of the source text was skipped as a result of the error, the part that was skipped will be indicated by a line containing the text **SKIP* at the left, and hyphens under the skipped text. Lines containing syntax errors will in addition be written on the terminal.

At the end of the listing a list of the error numbers and an explanatory text for each error will appear.

A list of all compiler error messages can be found in appendix A.

<code file> is the file on which the BRF output will be written. The <code file> parameter may be omitted, in which case no object code will be generated.

In a second or following COMPILE command, only <source file> need be specified. The previous <list file> and <code file> will be used if they were specified in a previous COMPILE command. If a new <list file> or <code file> is specified, the previous file will be closed, and the new file opened.

Be aware that option and flag values may be affected by a compilation, and thus may influence the result of a succeeding compilation. Use the CLEAR command to bring the processor back to its initial state.

3.3 CLEAR

The CLEAR command brings the command processor back to its initial state. The following actions are taken by CLEAR:

Set all options to their default values.
Delete all flags.
Close <list file> and <code file>.

3.4 OPTIONS

The OPTIONS command is used to set compiler options. The command and the options are described in section 2.4.3.

3.5 SET and RESET

The SET and RESET commands set a flag to TRUE and FALSE, respectively. These commands, and the use and effect of flags are described in section 2.4.1.

3.6 EXIT

The EXIT command closes all files and returns control to the operating system.

3.7 Program Compilation Example

Following is an example showing how a compilation of a program is performed. Computer-generated output is underlined.

Terminal input/output	Comments
@PASCAL	Call Pascal compiler
NORD-10 PASCAL. Version 78-11-01	Identifying text
$OPT B2,T-	Compile for 2-bank execution and suppress generation of test instructions.
$SET PARIS	Generate "PARIS" version of program. (Assumes source file contains $IFTRUE and $IFFALSE tests on flag with name PARIS.)
$COM	Compile
Source file=MYPROG	Source is MYPROG
List file=L-P	Listing to line printer
Code file=MYPROGCODE	BRF code goes to MYPROGCODE
NO ERRORS	Message from compiler
xx.xx SECONDS COMPILATION TIME	
$EX	Exit
@	Back to SINTRAN

4 PROGRAM LOADING AND EXECUTION

4.1 Program Loading

A compiled NORD-10 Pascal program must be loaded by the NRL loader before it can be executed. The reader should consult the NRL manual for details concerning the loader and the loading process. Here we will just give an example of how a Pascal program is loaded and executed:

```
Terminal input/output      Comments

@NRL                       Call loader
RELOCATIONG LOADER LDR-xxxxx   Identifying text
*L MYPROGCODE P-LIB        Load code file and Pascal library
FREE:xxxxxx-xxxxxx         Free memory area
*RUN                       Execute program
@                          Execution finished
```

BRF files containing the main program and external procedures may be loaded in any order. The Pascal library must always be loaded last.

Note that for a one-bank program, more space will be allocated for the stack and heap if the loading process is done via an image file.

Further information on how a running Pascal program utilizes memory, and how to make an absolute program, can be found in chapter 6.

4.2 Run-time Errors

If a program attempts to do an illegal operation, the program will abort with an appropriate error message. If the error was an illegal I/O operation, the name of the file variable involved will be part of the message. A list of all run-time error messages can be found in appendix B.

The error message will indicate at which absolute address (octal) the error occurred, and, if the T option was on during compilation, which line number in the source program this address corresponds to.

Be aware of the following pit-falls regarding the source program line number:

1 If the T option was turned off and on one or more times during the compilation, the source line number may be wrong.

2 If the program calls separately compiled procedures, the source line number may be that of an external procedure, if that procedure was compiled with the T option on.
3 If an error occurs within an external FORTRAN subroutine or function, the Pascal system will not be able to give any information about the error.

If there is any doubt regarding the source line number given in cases 1 and 2 above, you should correlate the octal address in the error message with the octal progam addresses in the listing by the help of a loader map. The loader map can be acquired by the NRL *ENTRIES-DEFINED command.

If the program aborts with the error message STACK-HEAP OVERFLOW, then your program needs more space for data. If the program ran with the B1 option, you can recompile the program with the B2 option and rerun the program.

5 INPUT/OUTPUT

Input/output is that part of a programming language which is most operating system dependent. Several design and implementation decisions therefore have to be taken by any implementor of Pascal. The reader is warned that some of the features described in this chapter may not be implemented, or may work differently, in other Pascal implementations.

5.1 File Variables

File types may be used as any other type in a Pascal program, with the following limitations:

1 A file variable may only be declared in the main program (i.e. file declarations within procedures and functions are not allowed).
2 file of . . . file of . . . is not allowed.
3 File variables, or structures containing file variables may not be generated with the NEW constructor.

5.1.1 The type TEXT

There is a standard file type TEXT. A file of type TEXT is assumed to contain a sequential text, subdivided into lines of maximum 136 characters each.

Note: In NORD-10 Pascal, the type TEXT is not equivalent to the type packed file of CHAR. The latter type will be interpreted as a sequence of characters where no line subdivision is visible.

The following procedures and functions may only be used on files of type TEXT:

EOLN READ READLN WRITE WRITELN

On input, the CR character (value 15_8) will be taken as a line separator. An LF character (value 12_8) following CR will be ignored. According to Standard Pascal, EOLN(<file>) will become TRUE when a READ(<file>,c) reads the last character before the CR. When EOLN(<file>) is TRUE, the next READ(<file>,c) will deliver the space character (value 40_8).

On output, WRITELN will write the two characters CR and LF.

The editing specifications in READ and WRITE are extended to enable I/O of the octal representation of integers. In READ, an integer parameter may be followed by a :n specification, while in WRITE, an integer parameter may have a :n

specification after the :<field width> specification. In both cases, if n has the value 8, the octal representation of the integer will be read or written. If n is not equal to 8, decimal conversion will be performed.

On output, a :<field width> specification with zero value is legal. In this case, the number will be written left justified, using the minimum space.

5.1.2 Standard files

There are two standard files, INPUT and OUTPUT, both of type TEXT. These files may therefore be used without declaration.

5.1.3 Packed files

In a GET or PUT-operation on a non-packed file, a whole number of 16-bit words will always be transferred.

In the declaration

packed file of T,

the key-word packed will have an effect only if values of type T occupy 8 bits or less. In these cases, PUT and GET will operate as follows:

If the values of type T occupy 6,7 or 8 bits:
Transfer one value.

If the values of type T occupy 1, 2, 3, 4 or 5 bits:
Pack (unpack) the maximum number of values in one 16-bit word. Transfer a word when it is full (PUT) or empty (GET).

Be aware that on reading a file of this kind, it may be the case that EOF is found too late, if the last word was not completely filled when the file was written.

5.2 OPEN and CLOSE

The procedures OPEN and CLOSE have been implemented in NORD-10 Pascal to enable run-time association between a file variable and an external file.

5.2.1 OPEN

The OPEN procedure can have up to 3 parameters:

OPEN(<file>,<filename>,<status>)

<file> is the variable name of the file.

<filename> is a string or an array of CHAR containing the external name of the file, the file type, and the SINTRAN file access code (R, W, WX, RX, RW, WA, WC, or RC). If no file type is given, the type :SYMB is assumed. The file name and the access code should be separated by a blank or a comma. In case no access code is given, the access mode is taken to be R (read sequential).

<status> is an integer variable where status for the open operation will be left.

Example:

```
var F: file of . . .;
    STS: INTEGER;
  . . .
  OPEN(F,'MYDATA:DATA RW',STS);
```

The action of OPEN is to open <filename>, and associate that file with <file>. Status for the operation is left in <status>. If the open operation was errorfree, <status> will be zero; if an error occurred, <status> will contain the SINTRAN error number.

One or both of the parameters <filename> and <status> may be omitted:

If the <filename> parameter is omitted, the OPEN procedure will

1 Write the name of the file name variable followed by an equal sign to the terminal.
2 Read the file name and access code entered at the terminal.
3 Open this file.
4 If the <status> parameter is present, leave status in <status>. If the <status> variable is not present, and the open operation resulted in an error condition, write the error message and
   a) repeat from 1 if an interactive job,
   b) abort the program if a batch job.

If the <filename> parameter is present, and the <status> parameter is omitted, the program will be aborted in case of an open error.

Remember that RESET or REWRITE must be called before I/O on the file can be performed.

5.2.2 CLOSE

The CLOSE procedure has one parameter:

CLOSE(<file>)

The external file will be closed and disassociated from the <file> variable. A later OPEN may associate <file> with another external file.

5.2.3 Program heading parameters

The program heading may have file variable names as parameters. For each of these file variables, the compiler will automatically generate an OPEN(<file variable>) call, preceding the first statement of the main program. In addition, for the file INPUT a RESET(INPUT) will be generated, and for the file OUTPUT, a REWRITE(OUTPUT) generated. For all file names in the program heading, except INPUT and OUTPUT, the call on RESET or REWRITE must be programmed.

Placing a file variable as a parameter in the PROGRAM statement, therefore has the effect that the user at the terminal is inquired to specify the actual external file name and access mode (cfr. section 5.2.1).

Since OPEN and CLOSE are not part of Standard Pascal, file variables in programs that are to be ported should appear in the program heading, instead of being explicitly opened by calls on OPEN.

5.3 Terminal I/O

When the actual external file is the terminal running the program, certain special actions are taken by the I/O system.

On input, a RESET will not read the first character into the file window, as specified in Standard Pascal. Instead, RESET will put the space character into the window, and set EOLN to TRUE. Thus, in the input from the terminal, an extra initial space will appear. The reason for this modification is to permit output to the terminal prior to the first input without program hang-up.

In a READ operation from the terminal, a number syntax error will not result in a program abortion. Instead, the message

ILLEGAL NUMBER SYNTAX

will be written to the terminal, and the READ performed anew, such that the correct number can be retyped.

An input TEXT file associated with the terminal will be given logical unit number zero. This enables editing of the terminal input with CTRL A and CTRL Q.

5.4 Random Access I/O

A file variable may be associated with an external random access file. Random access I/O may be done on that file with the procedures PUTRAND and GETRAND. Each of these procedures has two parameters:

<file> and <block number>

PUTRAND writes the current content of the file window to the given <block number> on the file. GETRAND reads the block in <block number> on the file into the file window.

The block size is equal to the number of words occupied by the file component type. This block size is determined when the file is opened by a call on OPEN.

RESET and REWRITE have no effect on random access files.

A random access file can not be packed.

5.5 WRITEEOF

WRITEEOF takes a file variable as a parameter. The procedure will write an end-of-file mark on the file, provided this operation is meaningful for the kind of medium on which the file resides.

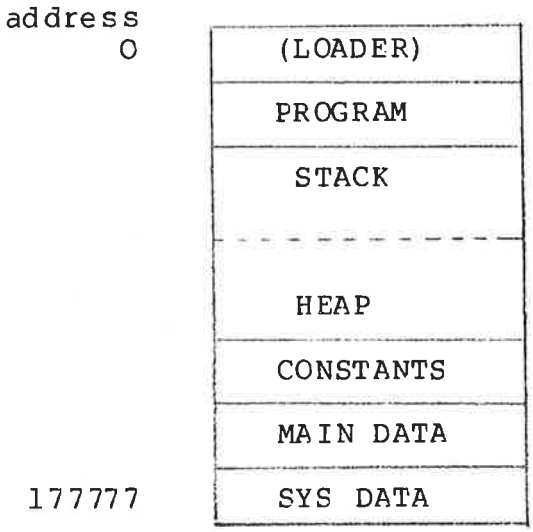
6 IMPLEMENTATION DESCRIPTION

This chapter will give some information on how the NORD-10 Pascal system works internally, to enable more advanced use of the system. Be aware that most of the features described in this chapter are very NORD-10 and SINTRAN dependent. Therefore, the reader should not assume that other Pascal implementations work in the same or a similar manner. Also, the reader is warned that implementation details may change in future versions of NORD-10 Pascal.

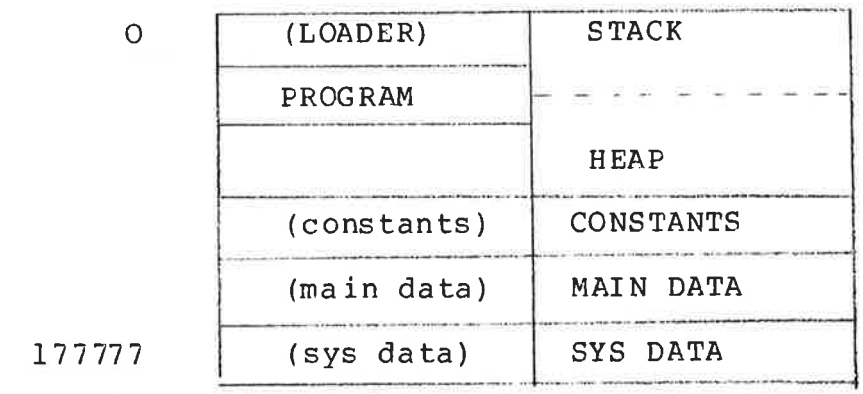
6.1 Memory Layout

The following figures show how memory is utilized by a running Pascal program (including the Pascal compiler itself).

One-bank program



Two-bank program



PROGRAM	The Pascal program together with the necessary library routines.
STACK	The memory used by procedures and functions that the program calls. The stack grows from low towards high addresses.
HEAP	The memory used by data allocated with the NEW constructor. The heap grows from high towards low addresses.
CONSTANTS	The constants of length 4 words or more referred to by procedures. For each procedure, a common block containing such data is allocated within the CONSTANTS area.
MAIN DATA	All variables declared in the main program. This area is a common block named C.MAIN.
SYS DATA	The variables and constants used by the Pascal library routines.

The object program and Pascal library are identical in the one- and two-bank versions. When running, the system detects the actual execution mode by sensing bit zero in the STATUS register.

The decision whether to run a program in one-bank or two-bank mode may be postponed till after the loading process has been completed. Use the NRL DEPOSIT command to change the BANKS variable to 0 (for one-bank execution) or 1 (for two-bank execution) before entering the RUN command. The BANKS variable is found at relative location 2 in the 5CCOM common area.

One-bank programs

Note that the space occupied by the loader is wasted. Doing the loading process via an image file so that the Pascal program starts at address 0, will give more space for the stack and heap. Thus, an absolute version of a program giving maximum space for the stack and heap will need a file of 64 pages.

Two-bank programs

A two-bank program is loaded exactly as a one-bank program. Before execution starts, the CONSTANTS, MAIN DATA, and SYS DATA areas are moved to the data bank. The data will be located at the same addresses as they had in the instruction bank.

To make a minimal absolute version of a two-bank program, use the NRL SET-LOAD-ADDRESS command to minimize the space betwee: the PROGRAM and CONSTANTS areas.

A two-bank program will usually be slower than a one-bank program due to the necessary ALTON and ALTOFF monitor calls within the Pascal library.

6.2 Loader Map

The compiler generates 7-letter entry point names. The names found in the loader map are constructed as follows:

Main entry point: The name given by the programmer in the PROGRAM statement.

Procedures and functions local to the program: These have the form nnnndd* where nnnn are the first four characters of the procedure or function name. dd are two invented characters, to make entry point names distinct.

Non-local labels: These have the form LABLdd+ where dd are invented characters.

External procedures and functions: The name given by the programmer.

Labelled common areas: These have the form nnnndd+ where nnnn are the first four characters of the procedure or function with which this common area is associated. dd are invented characters.

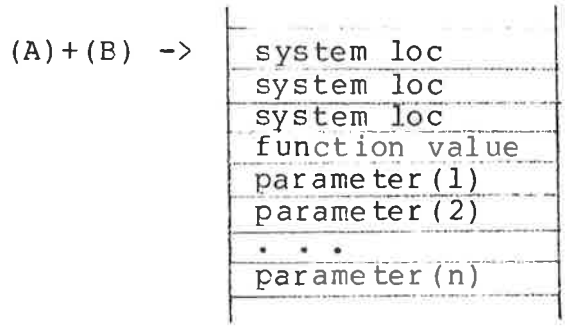
6.3 Procedure and Function Calls

The following information on how procedure and function calls are handled by Pascal should enable a user to write simple external routines in MAC or NPL.

For each procedure or function call, Pascal generates an object on top of the stack to hold system data, parameters, and data local to the routine. At the time of entry to the routine, the registers and stack contain the following data:

X Static Link
A Top of new procedure object relative to B
B Dynamic Link (calling procedure object)
L Return Address

Stack:



In a proper Pascal procedure, the three system locations are used to contain Static Link, Dynamic Link, and Return Address.

The function value occupies 0 words if the object is a procedure; 1, 2, or 3 words if the object is a function.

parameter(i) can have the following form:

when var parameter	reference to actual
when value parameter	k-word value if k<=8
	reference to actual if k>8

The routine may use 200 octal stack locations without causing stack-heap overflow.

On exit from a procedure or function, the following conditions must be satisfied:

The B-register must hold the same value as it had on entry.
For a function, the A-, AD-, or TAD-register must hold the function value.
The exit must be to Return Address (= contents of L-register on entry).

APPENDIX A

Compile-time Error Messages

```
  1:  Error in simple type
  2:  Identifier expected
  3:  'PROGRAM' expected
  4:  ')' expected
  5:  ':' expected
  6:  Illegal symbol
  7:  Error in parameter list
  8:  'OF' expected
  9:  '(' expected
 10:  Error in type
 11:  '[' expected
 12:  ']' exepcted
 13:  'END' expected
 14:  ';' expected
 15:  Integer expected
 16:  '=' expected
 17:  'BEGIN' expected
 18:  Error in declaration part
 19:  Error in field-list
 20:  ',' expected
 21:  '*' expected
 22:  Illegal character
 50:  Error in constant
 51:  ':=' expected
 52:  'THEN' expected
 53:  'UNTIL' expected
 54:  'DO' expected
 55:  'TO'/'DOWNTO' expected
 56:  'IF' expected
 57:  'FILE' expected
 58:  Error in factor
 59:  Error in variable
101:  Identifier declared twice
102:  Low bound exceeds high bound
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
109:  Type must not be real
110:  Tagfield type must be scalar or subrange
111:  Incompatible with tagfield type
112:  Index type must not be real
113:  Index type must be scalar or subrange
114:  Base type must not be real
115:  Base type must be scalar or subrange
```

117: Unsatisfied forward reference
118: Forward reference type identifier in variable declaration
119: Forward declared; repetition of parameter list not allowed
120: Function result type must be scalar, subrange or pointer
121: File value parameter not allowed
122: Forward declared function; repetition of result type not allowed
123: Misssing result type in function declaration
124: F-format for real only
125: Error in type of standard function parameter
126: Number of parameters does not agree with declaration
127: Illegal parameter substitution
128: Result type of parameter function does not agree with declaration
129: Type conflict of operands
130: Exepression is not of set type
131: Tests on equality allowed only
132: Strict inclusion not allowed
133: File comparison not allowed
134: Illegal type of operand(s)
135: Type of operand must be Boolean
136: Set element type must be scalar or subrange
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: Illegal parameter substitution
143: Illegal type of loop control variable
144: Illegal type of expression
145: Type conflict
146: Assignment of files not allowed
147: Label type incompatible with selecting expression
148: Subrange bounds must be scalar
149: Index type must not be integer
150: Assignment to standard function is not allowed
151: Assignment to formal function is not allowed
152: No such field in this record
153: Type error in read
154: Actual parameter must be a variable
155: Control variable must not be formal or global
156: Multidefined case label
157: Too many cases in case statement
158: Missing corresponding variant declaration
159: Real or string tagfields not allowed
160: Previous declaration was not forward
161: Again forward declared
162: Parameter size must be constant
163: Missing variant in declaration

164: Substitution of standard proc/func not allowed
165: Multidefined label
166: Multideclared label
167: Undeclared label
168: Undefined label
169: Error in base set
170: Value parameter expected
171: Standard file was redeclared
172: Undeclared external file
173: Fortran procedure or function expected
174: Pascal procedure or function expected
175: Missing file 'INPUT' in program heading
176: Missing file 'OUTPUT' in program heading
177: Illegal assignment to control variable
178: Variable used as control variable in outer loop
179: Read into control variable not allowed
180: Source line too long
181: Value of tagfield out of range
182: Illegal assignment to function name
183: Forward declared procedure not defined
184: Illegal jump to label
185: Variant already defined
190: Type must be scalar, subrange or array
191: Value list too long
201: Error in real constant: digit expected
202: String constant must not exceed source line
203: Integer constant exceeds range
204: 8 or 9 in octal number
205: Real number overflow
206: Real number underflow
207: Too many decimals
208: String constant of zero length not allowed
250: Too many nested scopes of identifiers
251: Too many nested procedures and/or functions
252: Too many forward references of procedure entries
253: Procedure/function too long
254: Procedure/function has too many long constants
255: Too many errors on this source line
256: Too many external references
257: Too many externals
258: Too many local files
259: Expression too complicated
260: Procedure/function has too many local variables
300: Division by zero
301: No case provided for this value
302: Index expression out of bounds
303: Value to be assigned is out of bounds
304: Element expression out of range
320: Internal error (reference out of range)
322: Internal error (GETOPR)
331: Internal error (LOADAD - packed address)
332: Internal error (LOADAD - condition address)

333: Internal error (MAKEMREG)
340: Internal error (SELECTREG)
380: Illegal command
381: Unknown command
382: Ambiguous command
383: Too many flags
384: Too deep nesting of INCLUDE files
385: INCLUDE open error
386: Missing file name in INCLUDE
390: EOF encountered on source file
398: Implementation restriction
399: Variable dimension arrays not implemented
400: Internal error (MOAVATTR, RESETGATTRP)

APPENDIX B

Run-time Error Messages

ARGUMENT TO EXP TOO BIG
The argument to EXP will cause arithmetic overflow.

ARGUMENT TO LN WAS <= 0
The logarithm of a negative number is not defined.

ARGUMENT TO SIN/COS TOO BIG
Lost accuracy makes the function result meaningless.

ARGUMENT TO SINH/COSH TOO BIG
The argument will cause arithmetic overflow in the result.

ARGUMENT TO SQRT WAS < 0
The square root of a negative number is not defined.

ARITHMETIC OVERFLOW
Overflow caused by
a) integer arithmetic operations,
b) floating division by zero, or
c) conversion of real to integer.

BAD ARGUMENT TO ARCTAN
Lost accuracy makes the function result meaningless.

BLOCK DOES NOT EXIST
Program tried to read non-existing block on a random file.

EOF ON INPUT
Program tried to read past end-of-file on an input file.

FILE ALREADY OPEN
Program tried to OPEN an already opened file.

FILE NOT OPEN
Program tried to access a non-opened file.

FILE NOT RANDOM
Program tried random access to a sequential file.

FILE NOT SEQUENTIAL
Program tried sequential access to a random file.

ILLEGAL ARGUMENT(S) TO POWER
Either attempt to raise negative number to a real power, or the arguments will cause arithmetic overflow.

ILLEGAL CASE INDEX
The case label corresponding to the value of the case variable is not defined.

ILLEGAL FORTRAN CALL
A FORTRAN routine was called from a two-bank Pascal program.

ILLEGAL NUMBER SYNTAX
The number being read did not have the correct syntax.

ILLEGAL PARAMETERS TO FORMAL PROC/FUNC
The actual parameters to a formal procedure or function did not correspond in number or type to the formal parameters.

ILLEGAL SUBRANGE ASSIGNMENT
Attempted assignment of a value outside the subrange, or

the controlled variable in a for-loop was of a subrange type and lower or upper bound of the loop was outside the subrange.

INPUT RECORD TOO LONG
A TEXT file record must not exceed 135 characters.

INTERNAL PASCAL ERROR
Error within the Pascal system. Contact a systems expert.

NO RESET
Program tried to read from a file without a previous RESET.

NO REWRITE
Program tried to write to a file without a previous REWRITE.

OPEN ERROR
Failure in an attempt to OPEN a file. The SINTRAN error message will indicate the cause.

POINTER IS NIL
Attempted access to data via a pointer with the value NIL, or call on RELEASE with a NIL-valued pointer parameter.

POINTER IS OUTSIDE HEAP
Attempted access to data via a pointer which did not point to data within the heap, or call on RELEASE with a pointer parameter that did not point within the heap.

RESET ON OUTPUT FILE
RESET was attempted on a write only file.

REWRITE ON INPUT FILE
REWRITE was attempted on a read only file.

SET ELEMENT OUTSIDE RANGE
Program attempted to construct a set with an element value not within the set type.

SHORT FIELD ON OUTPUT
The number value being output did not fit in the given field size.

STACK-HEAP OVERFLOW
The program generated too much data by calling procedures recursively or with the NEW constructor. Running the program in two banks (see section 1.2) may solve the problem.

SUBSCRIPT OUT OF RANGE
The index(es) to an array are outside the array bounds.

UNKNOWN LUN
There is no file open on this logical unit.

WRONG I/O PARAMETER
Illegal specification of the formatting of a number.

INDEX

NORSK DATA A.S.
Jerikoveien 20, Box 4, Lindeberg gård
OSLO 10

COMMENT AND EVALUATION SHEET

ND-60.086.02 NORD-10 PASCAL

In order for this manual to develop to the point where it best suits your needs, we must have your comments, corrections, suggestions for additions, etc. Please write down your comments on this pre-addressed form and post it. Please be specific wherever possible.

FROM _____

_____

_____