The Pascal Compiler

This compiler, like all Pascal compilers, is case insensitive. The examples in the example directory are in uppercase but that is not a requirement for using this compiler.

At the time this compiler was written, standard Pascal was described by two documents: Wirth, Niklaus. *The Programming Language Pascal*, Springer-Verlag, 1971 and Jensen, Kathleen, and Niklaus Wirth. *PASCAL User Manual and Report*, Springer-Verlag, 1974. The ISO standard ISO 7185:1990 was published 15 years after this compiler was written.

This compiler was designed for the IBM/360 which uses the EBCDIC character set. The original designers of this compiler included a number of character translation rules to accommodate the EBCDIC character set. These alternate representations are still enabled within the compiler along with the more natural characters available in the ASCII character set.

Alternate character representations used to accommodate the EBCDIC character set.

up arrow:	@
left square bracket:	(.
right square bracket:	.)
left curly bracket:	(*
right curly bracket:	*)

in addition, ~= can be used for <>; & for AND; | for OR; and ~ for NOT.
The ASCII characters ^ [] and {} are also supported.

Major differences between Stony Brook Pascal and the 1974 standard Pascal.

Identifiers can contain the underscore character('_').

The word "forward" is reserved.

Nonstandard files (i.e., all files other than input and output) are not supported.

A new predeclared function, 'clock':

```
function clock:integer;
```

which has no parameters, and returns elapsed CPU time in units of 0.01 seconds.

Of all the predeclared procedures and functions, only the arithmetic functions **sin, cos, exp, ln, sqrt** and **arctan** may be passed as formal parameters. A nonstandard syntax is used which requires programmers to fully specify the types of formal procedure and function parameters.

The write functions do not support field width.

```
v := 4096;
```

```
write(v:5);  /* This is an Error.  Write-parameters are not supported. */
```

Instead the compiler has implemented three predefined variables **intfieldsize**, **realfieldsize** and **boolfieldsize** to specify the minimum field width for when integer, real and boolean values respectively are written. For example:

```
v := 4096;
intfieldsize := 5;
write(v);        /* This will print 5 characters */
```

These variables are initialized to:

intfieldsize : 12
realfieldsize: 14
boolfieldsize: 6

As in Pascal 6000-3.4, the end of each line of output must be explicitly indicated by a call to **writeln**.

This compiler uses the EOLN function rather than an EOL character as originally defined by Wirth.

Comments

A comment may be defined in any of these ways:

```
(* This is standard comment *)
/* PL/I style comments are supported */
{ Curley bracket style comments are supported }
```

Comments may not be nested.

Compiler control toggles

A $ within a comment specifies that the next character is a control character. The upper case letters and the vertical bar (|) are used for control toggles. When $ <character> is encountered in a comment, the value of the corresponding toggle is complemented. Not all control toggles are used. In the current version, the useful toggles are:

$B - trace the execution of the LL1 parser (Note: This produces voluminous output.)
$D - print compilation statistics.
$E - print code (assembly format) emitted in pass 3.
$L - produce a formatted listing of the Pascal program.
$M - produce an unformatted listing of the source program.
$P - dump the parse trees produced by pass 1.
$S - dump the symbol table and the post-mortem tables.
$T - print the triples emitted during pass 2.
$Z - trace the opening of identifier scopes during pass 1, and dump the symbol table at the end of pass 1.
$| - set margin. The portion of succeeding cards starting from the column containing the | will be ignored.

The initial value of the uppercase alphabetic control toggles are set from the command line. Using -L on the command line is equivalent to using /* $L */ as the first line of the program. Unless enabled on the command line the default value of all control toggles is off.

Control toggles on the command line are only recognized by pass 1 of the compiler. Toggles for $P, $T and $E are sampled at the beginning of each function or block and passed to subsequent compiler passes to be acted on when that pass processes that block.

Compiler debug levels

The --debug=<number> option on the simulator may be used to set the debug level. Three levels are supported:

0 Disable all debug features.

1 Do range checking on array subscripts.
Preinitialize storage.
Initialize function return values.
Enable the Post Mortem Dump (PMD) program.

2 Do everything supported by level 1
Count the number of times each basic block is executed.
Display the source location where the error occurred.
(level 2 is default)

Files and I/O assignments

Pass 1

```
-i0 sourcefile.pas        # Pascal source file
-o3LF /tmp/tree.tmp       # Parse Tree
-o4LF /tmp/symb.tmp       # Symbol table
-f3w+ dgns.obj            # DGNS file.  Holds a compiled image of the source code.
```

Pass 2

```
-i2L /tmp/tree.tmp        # Parse tree
-i3L /tmp/symb.tmp        # Input Pass 1 Symbol table
-o3LF /tmp/symb2.tmp      # Output Symbol table
-o4LF /tmp/triple.tmp     # Triples
-o5LF data.obj            # Data section of the compiled program
-o7LF pmd.obj             # Post Mortem Dump tables
```

Pass 3

```
-i2L /tmp/symb2.tmp       # Symbol table from Pass 2
-i3L /tmp/triple.tmp      # Triples from pass 2
-o3LF code.obj            # Compiled code
-o4LF org.obj             # ORG section.  Used by the runtime
-o6LF line.obj            # Map line numbers to executable code.
```

The executable file is a combination of code.obj, org.obj and data.obj.
The Post Mortem Dump (PMD) program uses line.obj, pmd.obj and dgns.obj.

==========

Post Mortem Dump

The Post Mortem Dump program (PMD) analyzes the state of the program in the event that it aborts during execution. PMD will display the current value of variables and show usage counts for each line of code, perfect for doing code coverage.

In order to get PMD to execute only when the program aborts you can use the --pmd option on the command line. If your program has successfully run to completion the --pmd option will tell the simulator to exit. Otherwise the simulator continues processing until it reaches the PMD program. After running the PMD program the simulator will exit.

The PMD program uses the following files:

```
  -i5L line.obj       # Line number file generated by pass 3
  -i6L pmd.obj        # PMD file generated by pass 2
  -f3r dgns.obj       # DGNS file generated by pass 1
```

The PMD program uses files generated by the compiler. The scripts provided in the makefiles use the same filenames for every compile. If you are doing multiple compiles the files needed by PMD will reflect only the last program compiled. It might be easier to put each Pascal program in a separate directory.

==========

The XPL compiler

This version of the XPL compiler closely matches the compiler described in the book "A Compiler Generator" by McKeeman, Horning, and Wortman. The two biggest changes were an optimized scanner and the replacement of the MSP parser. The scanner was changed to use the IBM/360 TRT instruction.

Additional builtin functions

corehalfword(<index>) - Read or write 16 bits of memory at <index>. This is a 16-bit array declared at location zero that can access all of memory. *addr(corehalfword(2)) = addr(corebyte(4))*

op(<string>) - The character string is looked up in the IBM/360 opcode table and returned as an integer. e.g. OP('EX') returns "44". The opcodes must be in uppercase.

rewind(<direction>, <unit>) - Rewind the <unit>. I/O <unit> numbers are unique for INPUT and OUTPUT. INPUT(3) and OUTPUT(3) are different I/O streams. The <direction> parameter is used to select which <unit> is to be rewound. When <direction> is zero the <unit> is INPUT. When <direction> is one the

<unit> is OUTPUT.

Compiler control toggles

A $ within a comment specifies that the next character is a control character. The upper case letters and the vertical bar (|) are used for control toggles. When $ <character> is encountered in a comment, the value of the corresponding toggle is complemented. Not all control toggles are used. In the current version, the useful toggles are:

$B - Display emitted code in hexadecimal.
$D - Print compilation statistics.
$E - Print code (assembly format).
$L - Produce a formatted listing of the XPL program.
$M - Produce an unformatted listing of the source program.
$N - Issue a warning if not all the parameters are supplied.
$S - Dump the symbol table at the end of each procedure.
$T - Begin tracing the compiler.
$U - Stop tracing the compiler.
$V - Verbose debug information.
$Y - Use vertical bar (|) in the formatted display.
$X - Continue compiling even if there are more than 25 severe errors.
$Z - Allow the program to execute even if it has severe errors.
$| - Set margin. The portion of succeeding cards starting from the column containing the | will be ignored.

The initial value of the uppercase alphabetic control toggles are set from the command line. Using -L on the command line is equivalent to using /* $L */ as the first line of the program. Unless enabled on the command line the default value of all control toggles is off.

Files and I/O assignments

```
-i0 sourcefile.xpl        # XPL source file
-i2 xplib.xpl             # Compactify Source code
-f1w+ sourcefile.out      # Executable output file
-f2w+ /tmp/data.tmp       # Scratch file for compiled data
-f3w+ /tmp/string.tmp     # Scratch file for character strings
```

Carriage Control Characters for Line Printers

When this compiler was written the primary output device was the line printer. XPL has special facilities designed to handle the carriage control characters on the Line Printer. Text written to OUTPUT(0) would advance one line before printing and text written to OUTPUT(1) would use the first character of the text as a carriage control character. Typical values were:

Space (' ') to advance one line
Zero ('0') to double space
One ('1') page eject

These carriage control characters have little meaning in today's environment. Because of this the simulator ignores the first character written to OUTPUT(1) then the <unit> is changed from 1 to 0 and the output proceeds as usual. The following two OUTPUT statements so the same thing:

```
TEXT = '0Print this line';
OUTPUT(1) = TEXT;
OUTPUT(0) = SUBSTR(TEXT, 1);
```

==========

IBM/360 Simulator

The simulator, sim360, creates an environment where Pascal and XPL programs written for the IBM/360 can be run on any hardware. The simulator includes nearly all of the IBM/360 instruction set with the exception of privileged opcodes. All fixed point and floating point opcodes along with most of the decimal arithmetic opcodes are simulated. Both the Pascal and XPL runtime support code has been included in the simulator.

XPL I/O support

The I/O support modules include both sequential I/O as well as direct access I/O. Sequential I/O can be translated from ASCII to EBCDIC or EBCDIC to ASCII which allows programs compiled on the IBM/360 in the 70's to run on the simulator.

Pascal programs are limited to one input and one output device which normally points to stdin and stdout respectively. XPL programs may have up to 32 devices each for input, output and direct access. Each of these I/O streams are controlled by the command line of the simulator. The general form:

INPUT(<number>)	-i<number><flags> filename
OUTPUT(<number>)	-o<number><flags> filename
FILE(<number>)	-f<number><flags> filename

The <number> describes the I/O unit in the range of 0 through 31. The unit <number>s are unique for INPUT, OUTPUT and FILE so a program can have a maximum of 96 open I/O devices.

The <flags> field does two things:

1) Lowercase letters describe how the I/O stream is opened.
2) Uppercase letters describe how the I/O is handled.

All lower case and special characters in the <flags> field are passed to the *fopen*(3) call as the "mode" operand and the filename is used for the "path". Typical values for "mode" are r, w, a and +.

- r Open for reading. The stream is positioned at the beginning of the file. Fail if the file does not exist.
- w Open for writing. The stream is positioned at the beginning of the file. Create the file if it does not exist.

a Open for writing. The stream is positioned at the end of the file. Subsequent writes to the file will always end up at the then current end of file, irrespective of any intervening *fseek*(3) or similar. Create the file if it does not exist.

\+ When following r or w, the plus sign (+) will open the file for both reading and writing.

The "mode" characters follow the conventions of the host operating system and may contain additional letters or symbols.

Uppercase letters in the <flags> field tell the simulator how to treat the I/O after the file is opened. The following letters are supported for INPUT files:

A For INPUT files this tells the monitor that the data on the external media is ASCII. This is used to select the proper values for carriage return and linefeed. ASCII is the default.

E For INPUT files this tells the monitor that the data on the external media is EBCDIC. This is used to select the proper values for carriage return and linefeed.

F Fill with blanks up to 80 characters. The choice of fill character is determined by the A or E flags. 32 for ASCII and 64 for EBCDIC. If the record is longer than 80 characters then this option has no effect.

T Translate. If the I/O stream is ASCII this flag will translate it to EBCDIC. If the I/O stream is EBCDIC this will translate it to ASCII.

L Limit the record to 80 characters. On input this will limit the input to 80 characters. Longer records will be broken into 80 character records and passed to the program 80 characters at a time. This option is mainly used for EBCDIC files. The EBCDIC source files in this release use fixed length records without carriage return linefeed delimiters.

N Swap caret(^) and tilde(~)

The following letters are supported for OUTPUT files:

A For OUTPUT files this tells the monitor that the program is generating ASCII characters. This tells the monitor which character should be used in the fill option (F). ASCII is the default.

E For OUTPUT files this tells the monitor that the program is generating EBCDIC characters. This tells the monitor which character is used in the fill option (F).

F Fill with blanks up to 80 characters. The choice of fill character is determined by the A or E flags. 32 for ASCII and 64 for EBCDIC. If the record is longer than 80 characters then this option has no effect.

T Translate. If the I/O stream is ASCII this flag will translate it to EBCDIC. If the I/O stream is EBCDIC this will translate it to ASCII.

L On output this will suppress the addition of carriage return and linefeed characters.

D DOS mode. Use both carriage return and linefeed as a record

delimiter on output.

Examples:
Assume you have the following XPL program:

```
declare text character;
text = input;
do while length(text) > 0;
  output = text;
  text = input;
end;
eof;
```

Assuming copy.out is the name of the compiled executable then the above program can read an EBCDIC file and print it to the console using this command:

```
sim360 -i0ETL ../ebcdic/FILE01 -o0A stdout copy.out
```

Since stdout is default for -o0 the following command will also work:

```
sim360 -i0ETL ../ebcdic/FILE01 copy.out
```

If the files you wish to copy are ASCII then the following may be used:

```
sim360 copy.out
```

The above command will also convert UNIX files to DOS format. The following will do the reverse conversion:

```
sim360 -o0D stdout copy.out
```

FILE record size option

--size=bytes sets the record size for the FILE() pseudo variable.
The default record size is taken from the header of the object module.
The compiler sets this size to 7200 bytes.

Options used to set MONITOR_LINK.

XPL has a four word array that is passed from one module to the next. This array is defined as MONITOR_LINK in the XPL program. The Pascal compiler uses this area to send information from one pass of the compiler to the next.

--time=SSS - Max run time in seconds. MONITOR_LINK(0)
When set on the simulator command line the --time option will abort the program when the allotted time is exceeded. Default is disabled.

--lines=LLLL - Number of lines. MONITOR_LINK(1) (default 1000)
The value of --lines is placed into MONITOR_LINK(1) but otherwise has no effect.

--debug=D - Debug level. MONITOR_LINK(2) (default 2)
Set the debug level for the Pascal compiler. This has no effect on XPL programs.

Other Options
--trace - Begin tracing execution immediately when the program starts.
  The option works for both XPL and Pascal programs. Default disabled.

--watch=HHHHHHHH - Watch a memory location and report when the value
  at that location changes. Default disabled.

--version - Print the version information and exit.

-c - Display Op-code instruction execution counts when the program
  terminates. Default disabled.

-m - Allow the code section to be modified. Default disabled.

-v - Verbose. Prints stuff you probably won't find interesting.
  Default disabled.

@filename - This option allows additional options to be placed in a file.
  The file may contain any number of options and use multiple lines.
  Text after a # is considered a comment, which extends to the end of the line.

Control toggles.
  Uppercase letters immediately after a single minus sign (-) will be passed to the compilers as the initial value of a compiler control toggle.
Examples:
  sim360 -L xpl.out # Same as using /* $L */ as the first line of the program.
  sim360 -LS xpl.out # Same as using /* $L $S */

Control toggles are cleared after each module executes.