Pascal-S

Pascal Subset Compiler/interpreter

Scott A. Franco

1 Overview of Pascal-S

This section contains background material on Pascal-S. If you want to get started using Pascal-S now, skip to “Using Pascal-S”

1.1 Introduction

Pascal-S was a companion interpreter/compiler implementation for a subset of the Pascal language created in about 1975 by Niklaus Wirth in ETH, Zurich. It was intended as a lightweight implementation for students to use on a batch or timeshare terminal based computer system in an era when computers were expensive with limited access. It was created as part of the Pascal-P family.

The prime use of the Pascal-P and Pascal-S family today is to learn how compilers work. Pascal itself is a very regular language with straightforward rules and syntax. Further, most of the various versions of Pascal-P and Pascal-S still exist, and thus you can pick a subset and level of complexity for the implementation you want to study:

Implementation	Lines of code
Pascal-S	1930
Pascal-P2	4703
Pascal-P4	5218
Pascal-P5	9046

The Pascal-P family is different than Pascal-S. Pascal-S is a combined compiler/interpreter. Pascal-P generates intermediate code in text file form, and has a separate compiler and interpreter section. Although Pascal-S was created by the same group (Niklaus Wirth’s ETH students and himself) as Pascal-P, it is significantly different in form.

The original paper that discusses the Pascal-S system is here:

http://www.standardpascal.com/Wirth-PascalS.pdf

And also in the Pascal-S archive. That article is a very complete explaination of the implementation, and I won’t repeat it here. It includes a complete copy of the original source for Pascal-S.

1.2 Language of Pascal-S

Pascal-S omits many features of full Pascal, including:

- Subrange types are not implemented (type a = 1..10).
- Scalar types are not implemented (type a = (one, two, three)).
- Sets are not implemented.
- Dynamic variables (pointers) are not implemented.
- Variant records are not implemented.
- Procedure/function parameters.
- gotos.
- The predefined functions succ and pred only function on type char.
- Only files of type "text" can be used, and then only the ones that are predefined by Pascal-S, which are "input", "output".
- Curly bracket comments {} are not implemented.
- Packing, the "packed" keyword, and the "pack" and "unpack" procedures, are not implemented.
- "get", "put", and file buffer variable handling are not implemented.
- Strings are unimplemented, except for literals as parameters to write/writeln, and those cannot have field lengths applied to them.
- The "forward" specifier, and forwarded procedures and functions, are not implemented.
- The predeclared identifiers maxint, text, round, page, new, dispose, and the functions they represent, are not present.
- The procedures reset, rewrite, pack and unpack are not implemented (they are recognized as valid predefined procedures, but give an 'unimplemented' error on use).

2 Using Pascal-S

2.1 Configuring Pascal-S

Pascal-S has a simple configuration script to set up the binary, script files and compiler in use for the system, that uses the proper defaults for your system:

[Windows]

```
> setpath
> configure
> make
```

[Linux/Mac]

```
$ ./setpath
$ ./configure
$ make
```

You can avoid “setpath” by placing the ./bin directory on your path.

The configure script attempts to automatically determine the environment you are running under, choose the correct compiler, bit width of your computer, etc. You can override this by using the options for configure:

Option	Meaning
--gpc	Selects the GPC compiler.
--ip_pascal	Selects the IP Pascal compiler.
--fpc	Selects the FPC Pascal compiler.
--help	Prints a help menu.

The configure script will take the preconfigured versions of the Pascal-S binaries, the script files and other files and install them for the specified compiler. The Pascal-S system is configured by default for GPC Pascal running on windows, and can be left as such if desired.

Note that if you have more than one acceptable compiler resident, the configure script will choose the first one found in the order IP Pascal, GPC Pascal, and then FPC Pascal.

2.2 Compiling and running Pascal programs with Pascal-S

To simply compile a run a program, use the ps batch file:

```
C:\> ps hello
```

When a pascal program is run this way, it gets it's input from the terminal (you), and prints its results there. The ps script accommodates the compiler that was used to build the system, and therefore you don’t need to know the exact command format of the executable.

All files in Pascal-S are anonymous (as in “no filename”), and only last the length of the program run. If you need to read from a file or write to a file use redirection:

```
C:\> ps test < myinputfile > myoutputfile
```

You will find you can get a lot of tasks done this way.

2.3 Other operations

Within the Pascal-S toolset, you will find a series of scripts to perform common operations using Pascal-S. This includes building the compiler and interpreter using an existing ISO 7185 compatible compiler, and also testing Pascal-S.

The scripts used in Pascal-S are designed to be independent of what operating system you are running on. The Pascal-S system as been successfully run on the following systems:

- Windows
- Ubuntu linux
- Mac OS X

To enable this to work, there are two kinds of scripts available, one for DOS/Windows command shells, and another for Unix/Bash. These two script files live side by side, because the DOS/Windows scripts use a .bat extension, and Bash scripts use no extentions. Thus, when a script command is specified here, the particular type of script file is selected automatically.

The only exception to this rule is that Unix users commonly do not place the current directory in the path. This means to execute a script file in the current directory, you need to specify the current directory in front of the script. For example:

~/Pascals$ ./ps hello

2.4 Reliance on Unix commands in the Pascal-S toolset

Most of the scripts in this package, even the DOS/Windows scripts, rely on Unix commands like cp, sed, diff, chmod and others. I needed a reasonable set of support tools that were command line callable, and these are all both standard and reasonable.

For Windows, the Cygwin toolset is available:

http://www.cygwin.com

Note that to run the cygwin tools, you will need the environment variable:

CYGWIN=nodosfilewarning

This prevents cygwin utilities from complaining about dos mode file specifications.

An alternative to Cygwin is the Mingw toolkit. Mingw uses GNU programs that are compiled as native Windows .exe files without special .dll files. It typically has better integration with Windows than Cygwin, since it does not try to emulate Unix on Windows. MinGW is available at:

http://www.mingw.org/

Where possible, I have tried to use DOS/Windows commands. The scripts are available in both DOS/Windows and bash versions. I could have just required theuse of bash, which is part of the cygwin toolkit, but my aim is not to force Windows users into a Unix environment.

2.5 The “flip” command and line endings

Every effort was made to make the Pascal-S compile and evaluate system idenpendent of what system it is running on, from Windows command shell, to Linux with Bash shell. One common thing I have found is that several utilities don’t appreciate seeing a line ending outside of their “native” line ending, such as CRLF for Windows, and LF for linux. Examples include “diff” (find file differences) and Bash.

Therefore many of the scripts try to remove the line ending conserations, either by ignoring such line endings, or by converting all of the required files to the particular line ending in use.

The key to this is the “flip” utility. After searching for several line ending converters, “flip” was found on the most number of systems, as well as being one of the most clear and reliable utilities (it translates in both directions, it tolerates any mode of line ending as input, will not corrupt binaries, etc.).

Unfortunately, even flip was not found on some systems. The simpliest way to fix this was to include the flip.c program with the distribution, then let you compile to form a binary on your system to replace the utility.

To make the flip utility, you run:

$ make_flip

Then flip will exist in the root directory.

3 Building the Pascal-S system

3.1 Compiling and running Pascal-S with an existing ISO 7185 compiler

You do not need to compile Pascal-S unless you are using an alternative compiler or installation. The current Pascal-S has been compiled and run with the following compilers and operating systems:

Compiler	Installations
IP Pascal	Windows
GPC	Windows, Ubuntu, Mac OSx

First, you must have a ISO 7185 Pascal compiler available. There are several such compilers, see:

http://www.standardpascal.org/compiler.html

You will probally need to compile pcom.pas and pint.pas with the ISO 7185 Pascal compatibility mode option on for your compiler. See your documentation for details.

If you are using a compiler or version of a compiler that is not tested to ISO 7185 standards, you will want to make sure that it is ISO 7185 compliant. There are tests for a full ISO 7185 compiler implementation included in the Pascal-P5 project.

To compile Pascals, the Pascal-S compiler/interpreter, use the make utility:

make Make pascals.

make pascals Make pascals.

make clean Remove all product files (usually to insure a clean remake).

To run the other programs and batch files, you should modify the following files to work with your compiler:

ps[.bat] The single program compile and run batch file.

The reason you need to change these files is because pascals uses the header file "srcfil" to read the source file. You need to find out how to connect this file in the program header to an external named file.

For example, in IP Pascal and FPC, header files that don't bear a standard system name (like "input" and "output") are simply assigned in order from the command line. Thus, ps.bat is simply:

```
ps %1.pas
```

Where %1 is the first parameter from the command line.

If your compiler does nothing with header files at all, you will probally have to change the handling of the prd and prr files to get them connected to external files. To do this, search pcom and pint for "!!!" (three exclamation marks). This will appear in comments just before the declaration, reset and rewrite of these files.

3.2 Notes on using existing compilers

When using an existing ISO 7185 compatible compiler, the configure script will install all the files needed to work with that particular compiler. Thus these hosts will work without any modification.

3.2.1 GPC

GPC (GNU Pascal Compiler) is used in the following version:

```
GNU Pascal version 20070904, based on gcc-4.1.3 20080704 (prerelease)
(Ubuntu 2.1-4.1.2-27ubuntu2).
Copyright (C) 1987-2006 Free Software Foundation, Inc.
```

I have had several difficulties with other versions of GPC, which give errors on standard ISO 7185 source, or crash, or other difficulties. The GPC developers announced they were halting development on GPC in the gpc mailing list. Please see their web page:

http://www.gnu-pascal.de

For any further information.

The main difficulty with GPC vis-a-vie Pascal-S is that testing of the GPC compiler for ISO 7185 compatability was not regularly done on GPC releases. Thus, otherwise working GPC releases were not able to compile and run standard ISO 7185 source code.

Because of this, I can only recommend the above version of GPC be used, which compiles and runs Pascal-S error free.

In addition, please be aware that I have not run the GPC compiler, including the above version, through a current ISO 7185 compliance test such as appears here. My only concern is that GPC be able to complile and run Pascal-S, and that the resulting Pascal-S runs the compliance tests. I leave it for others to run full compliance for GPC itself.

3.2.1.1 GPC on Cygwin

The current Cygwin release as of 2012/03/26 does not work, since it uses GPC 2005, and is broken at that (it has the .dlls for GPC installed incorrectly).

A procedure to use GPC under the current Cygwin I have used is as follows:

1. Install the latest version of Cygwin (the one I tried is Cygwin/X, a very useful package).
2. Place c:\cygwin\bin on your path.
3. Go to the website:

   http://www.gnu-pascal.de/binary/cygwin/

   And download and install:

   gpc-20070904-with-gcc.i686-pc-cygwin.tar.gz

   (4.4mb, gpc-20070904, based on gcc-3.4.4, with gcc-3.4.4 support files)

4. After installing this package in an appropriate directory, say c:\gpc, modify your path to include c:\gpc\usr\bin ahead of the c:\cygwin\bin directory in the path.

5. Add cygwin=nodosfilewarning (as stated in section 2.4 “Reliance on Unix commands in the Pascal-S toolset”).

Now you will be able to follow the normal gpc instructions here to get Pascal-S running, using the standard Windows command shell. Note that this trick won’t work with the command shells Cygwin provides.

To reiterate the steps that follow:

$ configure --gpc	Configure for GPC compiler.
$ make	Build the Pascal-S binary.
$ regress	Run the regression suites to check the Pascal-S compiler.

3.2.1.2 GPC for mingw

Mingw (Minimal GNU for Windows) is a different port of the GNU catalog for windows that runs directly on windows. That is, each binary is statically linked with its support library, and it is designed to work with windows directly.

As Cygwin has become more and more a full emulation of the Unix environment (a good thing), it has become less usable in interaction with other Windows programs. Thus I have found the mingw package more cooperative for every day Windows work.

Mingw does not come natively with GPC installed (or much else). I recommend you also pick up the MSYS package for mingw, which is a series of GNU programs that are compiled to run in the windows environment using Mingw.

To get the mingw distribution of GPC, follow the steps:

1. Go to the website:

   http://www.gnu-pascal.de/binary/mingw32/

   And download and install:

   gpc-20070904-with-gcc.i386-pc-mingw32.tar.gz

   (4.4mb, gpc-20070904, based on gcc-3.4.5, with gcc-3.4.5 support files)

2. After installing this package in an appropriate directory, say c:\gpc, modify your path to include c:\gpc\usr\bin directory in the path.

Note that this is based on a slightly different version than Cygwin.

To reiterate the steps that follow:

$ configure --gpc	Configure for GPC compiler.
$ make	Build the Pascal-S binaries.
$ regress	Run the regression suites to check the Pascal-S compiler.

3.2.2 FPC

FPC is can be used in versions 3.0.4 or later:

Free Pascal Compiler version 3.0.4 [2017/10/06] for i386

Copyright (c) 1993-2017 by Florian Klaempfl and others

3.2.2.1 Obtaining FPC

FPC can be found at the web site:

https://www.freepascal.org/

Please make sure you download and run at least version 3.0.4.

3.2.2.2 Configure and build the FPC version

The steps to configure and build with FPC after Pascal-S and FPC are installed are:

$ setpath

$ Configure –fpc

$ make

And I recommend a check of the Pascal-S compiler with:

$ regress

Which will run a test series on the resulting compiler.

4 Files in the Pascal-S package

Note: for script files, both a DOS/Windows (X.bat) and bash script (X) are provided. Their function is identical, one is for use with the DOS/Windows command shell, the other for bash shell.

configure configure.bat	Sets the current compiler to use to create Pascal-S binaries.
INSTALL	Installation instructions
LICENSE	License information for Pascal-S
Makefile	The make file to run builds on Pascal-S. This is customized for a particular host compiler.
NEWS	Contains various information about the current release.
README	Brief introduction to the project, it points to this document now.
setpath setpath.bat	Adds the “bin” directory to the current path. Used to quickly run procedures in Pascal-S directory.
TODO	Contain a list of "to do" items in Pascal-S.

4.1 Directory: bin

diffnole diffnole.bat	Runs a diff, but ignoring line endings (DOS/Windows vs. Unix). Also ignores version numbers in compiler output.
doseol doseol.bat	Fixes the line endings on text files to match the DOS/Windows convention, CRLF.
fixeol fixeol.bat	Arranges the line endings on bash scripts to be Unix, and those of the DOS/Windows scripts to be DOS/Windows line endings. This is required because the editors on the respective systems insert their own line endings according to system, and this can cause problems when they are run on a different system.
make_flip make_flip.bat	A script to compile deoln and ueoln and create a flip script for Unix. This is used to replace the "flip" program if required.
pascals pascals.exe	The compiled binary for Windows/Unix. See comments in 2.2 for how to use this. All of the supplied batch files are customized for this version.
ps ps.bat	A batch file that compiles and runs a single Pascal program. You will need to change this to fit your particular Pascal implementation. It uses input and output from the terminal, so it is a good way to run arbitrary programs. *** You will need to change this to fit your particular Pascal system *** It uses input and output from the terminal, so is a good way to run arbitrary programs.
regress regress.bat	The regression test simply runs all of the possible tests through Pascal-S. It is usually run after a new compile of Pascal-S, or any changes made to Pascals-S.
repo_ready repo_ready.bat	Get the current Pascal-S tree ready for a push to the git repo.
testprog testprog.bat	An automated testing batch file. Runs a given program with the input file, delivering an output file, then compares to a reference file. Testprog is used to test the program files for Pascal-S.
unixeol unixeol.bat	Fixes the line endings on text files to match the Unix convention, LF.

4.2 Directory: c_support

c_support/flip.c — C program to replace the local version of "flip", the Unix line ending fixup tool. It is provided in source form here because not all Unix installations have it (for example MAC OS X didn't have it). This allows you to compile it yourself for your target system.

4.3 Directory: doc

iso7185rules.html — A description of the ISO 7185 Pascal language.

Pascals.docx — This document, in Word 2010.

The_Programming_Language_Pascal_1973.pdf

Niklaus Wirth's description of the Pascal language, the last version to come from ETH. This is the equivalent of the "Report", from "Pascal user's manual and report [Jensen and Wirth].

Wirth-PascalS.pdf — The original Niklaus Wirth paper describing the Pascal-S system.

4.4 Directory: fpc

This directory contains scripts and other files specifically modified for FPC.

Makefile — The FPC specific version of the make input file for Pascal-S builds.

ps
ps.bat — The FPC specific version of the ps script.

4.5 Directory: Fpc/linux_X86

A placeholder for Linux specific files.

4.6 Directory: mac_X86

A placeholder for Mac OS X specific files.

4.7 Directory: fpc/windows_X86

A placeholder for Windows specific files.

4.8 Directory: gpc

This directory contains scripts specifically modified for GPC.

Makefile — The GPC specific version of the make input file for Pascal-S builds.

ps
ps.bat — The GPC specific version of the ps script.

4.9 Directory: gpc/linux_X86

A placeholder for Linux specific files.

4.10 Directory: mac_X86

A placeholder for Mac OS X specific files.

4.11 Directory: gpc/windows_X86

A placeholder for Windows specific files.

4.12 Directory: ip_pascal

This directory contains scripts specifically modified for IP Pascal.

Makefile	The IP Pascal specific version of the make input file for Pascal-S builds.
ps ps.bat	The IP Pascal specific version of the Pascal-S script.

4.13 Directory: ip_pascal/windows_X86

A placeholder for Windows specific files.

4.14 Subdirectory: sample_programs

Note that each test program as a .pas, a .inp (batch input, which could be empty) and output compare file .cmp. See 5.1.1 for more information.

hello.inp hello.cmp hello.pas	The standard "hello, world" program.
prime.cmp prime.inp prime.pas	A Pascal benchmark program.
roman.inp roman.cmp roman.pas	Prints roman numerals. From Niklaus Wirth's "User Manual and Report".

4.15 Directory: source

source/pascals.pas	The compiler/interpreter source in Pascal.

5 Testing Pascal-S

5.1 Running tests

5.1.1 testprog

The main test script for testing is:

testprog <Pascal source file>

testprog.bat <Pascal source file>

testprog is a “one stop” test resource for most programs. It expects the following files to exist under the given primary filename:

program.pas The Pascal source file.

program.inp The input file for the running program.

program.cmp The reference file for the expected output.

testprog compiles and runs the target program, and checks its output against the reference file. Several files are produced during the process:

Program.lst Contains the output from the program when run. This is all output to the standard “ output” file.

Program.dif This is the output of the diff command between program.lst and program.cmp. It should be empy if the program produced the output expected.

Not all of the files for testprog need to have contents. For example, a program that does not do input does not need to have a program.inp file. An example of this would be the “hello” program. However, testprog expects all of the files to exist.

To determine if the test ran correctly, the output of first the program.err file should be checked for zero errors, then the resulting program.dif file is checked for zero length. All of these results are announced during the test:

```
C:\projects\PASCAL\pascals>testprog sample_programs\roman
Compile and run sample_programs\roman
```

For a program that has a compile error.

```
C:\projects\PASCAL\pascals>testprog sample_programs\roman
Compile and run sample_programs\roman
03/18/2012  10:44 AM                76 roman.dif
```

For a program that does not match it’s expected output.

Regression testing

The regression test is performed as:

```
C:\projects\PASCAL\pascals>regress
Compile and run sample_programs\hello
02/24/2018  11:36 PM                 0 hello.dif
Compile and run sample_programs\roman
02/24/2018  11:36 PM                 0 roman.dif
Compile and run sample_programs\prime
02/24/2018  11:36 PM                 0 prime.dif
```

The regression test simply runs all of the sample programs and checks output.

6 Licensing

Pascal-S is derived from the original sources of the Pascal-S compiler from ETH Zurich, as created by Niklaus Wirth and his students. It was and is public domain, as acknowledged by Professor Wirth, and I add my modifications to it to the public domain as well.

Public domain is a widely misunderstood concept. There is no "license" possible nor needed for public domain works. There are no restrictions on it's use, nor do it's authors have any rights to it. It can be used for any purpose, public or private, and distributed or modified for any use whatever, paid or not.

The following are typical answers to questions about public domain works in general, and this work in specific.

Q. The Berne convention states that copyright in europe, where Pascal-S originated, is automatic. Doesn't that make Pascal-S a copyrighted work?

A. The laws in all copyright countries dictate what must be done to qualify as a copyrighted work. Since there is no specific legal agreement concerning public domain work, public domain is shaped by what constitutes enforceable copyright. The most common features of public domain are, but not limited to:

1. The author has stated the work is public domain.
2. The work has been distributed freely and with knowledge of the author(s).

In the case of Pascal-S, both are true.

Q. Dosen't public domain mean that I may no longer be able to gain access to the source?

A. If every copy of the work were to be erased or burned, but that is virtually impossible. Nobody can order you to release your copy since, by definition, there are no "rights" to a public domain work.

Q. Can't someone just copyright or patent the work later?

A. Showing that a work is in the public domain is part of denying copyright or patent to a work. By definition, a legitimate public domain work cannot later be copyrighted or patented.

Q. Can't someone improve the work, then gain rights to that derived work and thus restrict it's use?

A. Anyone can improve a public domain work, but they only have rights to their improvements, not to the original work. If their improvements are trivial, then it would be trivial for others to add that functionality. If it is not trivial, then you might want to pay for it.