Pascal implementation

The P4 Compiler

Scott A. Moore

1 Overview of Pascal-P4

This section contains background material on Pascal-P4. If you want to get started using Pascal-P4 now, skip to 2 “Using Pascal-P4”

1.1 Introduction

The Pascal-P series compilers were the original proving compilers for the language Pascal. Created in 1973, Pascal-P was part of a “porting kit” designed to enable the quick implementation of a Pascal language compiler on new machines. It was released by Niklaus Wirth’s students at ETH in Zurich.

The implementation and description of the language Pascal in terms of itself and in terms of a “pseudo machine” were important factors in the propagation of the language Pascal. From the early version of Pascal-P came the CDC 6000 full compiler at Zurich, several independent compilers including an IBM-360 compiler and a PDP-11 compiler, and the UCSD “byte code” interpreter.

The original article for the Pascal-P compiler is at:

http://www.standardpascal.org/The_Pascal_P_Compiler_implementation_notes.pdf

In the name “Pascal-P” the “P” stood for “portable”, and this was what Pascal-P was designed to do. It also stood for an example and reference implementation of Pascal, although Wirth later issued a paper, together with Tony Hoare for the “Axiomatic definition of Pascal”, which was also aimed at exactly specifying the semantics of Pascal.

As the importance of Pascal-P grew, the authors adopted a version number system and working methodology for the system. A new, cleaner and more portable version of the system was created in 1974 with the name Pascal-P2, and left the multiple early versions of the system as termed Pascal-P1.

From the Pascal-P2 revision of the compiler comes many of the original Pascal compilers, including UCSD. In 1976, Wirth’s group made one last series of improvements and termed the results Pascal-P3 and Pascal-P4. Pascal-P3 was a redesigned compiler, but used the same pseudo machine instruction set as P2, and thus could be bootstrapped from an existing P2 implementation. P4 featured a new pseudo instruction set, and thus was a fully redesigned compiler.

The Zurich implementations of Pascal-P were:

- Pascal-P1 1973
- Pascal-P2 1974
- Pascal-P3 1976
- Pascal-P4 1976

Pascal-P was always an imcomplete implementation of the Pascal language (a subset), and was designed to be so. After it was created, the ISO 7185 standard for Pascal was issued, and today Pascal-P4 exists and is still usable with minor changes to bring it into ISO 7185 compliance.

However, Pascal-P4 has it’s legacy problem of being a subset compiler of the full language. Further, it is only usable for programs that avoid its weaknesses, such as string storage. Keep in mind that Pascal-P was never designed to be a general purpose system, but rather to compile itself on a new machine – and then rapidly be improved to become a full compiler.

You will find standards related to Pascal here:

http://www.standardpascal.com/standards.html

The Pascal-P compiler series (and it's companion Pascal-S) have been extensively documented in the literature. However, the book "Pascal implementation: The P4 compiler" by Steve Pemberton and Martin Daniels stands out as a running code commentary on the level of "lion's commentary on Unix", it is just that good. It is also available free on line at:

http://homepages.cwi.nl/~steven/pascal/book/.

1.2 Differences between Pascal-P4 and full Pascal

P4 is not a full Pascal at all, but rather a subsetted version of the language with several features removed. The omissions and changes were (from the P4 web page at: http://www.standardpascal.com/p4.html):

- Procedure/function parameters.
- Interprocedural gotos (goto must terminate in the same procedure/function).
- Only files of type "text" can be used, and then only the ones that are predefined by P4, which are "input", "output", and two special files defined so that P4 can compile itself.
- "mark" and "release" instead of "dispose".
- Curly bracket comments {} are not implemented.
- The predeclared identifiers maxint, text, round, page, 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).
- Undiscriminated variant records.
- Output of boolean types.
- Output of reals in "fixed" format.
- Set constructors using subranges ('0'..'9').

However, there are several other issues with the P4 compiler beyond simply the language it implemented. P4 made no attempt to economize on its storage of strings. This means that each string constant, no matter how long it was, is stored in a fixed length in the pseudo-machine code. Although the internal string length in the interpreter was a settable constant, and implementor using P4 is always be operating between the mutually exclusive goals of having enough string characters to represent usable strings and having the total string storage use too much space.

Also, every variable in the P4 interpreter was afforded the same space. A character or a boolean used the same space as a floating point value, and an array of characters would be as costly as an array of floating point numbers.

P4 itself gets around these limitations by using strings and string constants sparingly. This again goes to the idea that P4 (and the Pascal-P series) was primarily designed to compile itself, and was never designed as a real, working compiler.

Very tellingly, when Kenneth Bowles received P2 and wanted to use it as an interpreter in and of itself, not as just a stepping stone to a native compiler, his team extensively reworked it to use a byte orientation, and implemented string storage efficientcy.

The exact reasons why P4 is as it is, of course, belong to it's original authors. However, it is fair to say that Pascal-P was designed to be a lightweight porting kit for Pascal. The two main concerns were:

- Limiting the required memory for the run of the self compilation.
- Limiting the complexity of the self compilation.

For the first, obviously the Zurich crew was not particularly limited by memory. The CDC 6000 series computers they had access to were state of the art for their day, and after Pascal-P was produced, they extended it to a full native compiler for the CDC 6000 (see http://www.standardpascal.com/CDC6000pascal.html). This implies that a full language version of P4 could have been completed. However, they may have wished to lessen the load on other implementers of the language outside of Zurich.

The second reason is far more concrete. Even a complex program such as a compiler may not use the entire language, simply because the need did not arise. It was, and is, standard practice to implement a subset of a full language for the first compiler version and improve it later.

Finally, it is important to understand that the designers of Pascal-P never intended it to be used as a implementation for it's own sake. A Pascal implementation that simulated, not executed, its output code was interesting to Wirth, but that resulted in the Pascal-S project, a one piece compiler/interpreter program that has also been said to have originated with the Pascal-P project (although you will find little in common between the source code for the two).

Thus, it would never have occurred to the original designers to make Pascal-P an efficient and full implementation of Pascal. It was simply a bridge to better things.

1.3 Differences between this Pascal-P4 and Pemberton's book

The code here is slightly different from that in the book, but the line numbers have been kept the same. The changes were to allow modern Pascal compilers to compile the source (there were some laxities in the original code). The following are the changes:

```
The type marktype is added for the parameters of the routines mark and
release:
    76c76
    <
    ---
    >       marktype= ^integer;

The type setty (which represents set types) is added for the new type
compatibility rules of ISO Pascal:
    95c95
    <
    ---
    >       setty = set of setlow..sethigh;
    100c100
    <                           pset: (pval: set of setlow..sethigh);
    ---
    >                           pset: (pval: setty);

Missing variant parts:
    123c123
    <                                   declared: (fconst: ctp));
    ---
```

```
    >                                   declared: (fconst: ctp); standard: ());
    145a146
    >                       types: ();
    149,150c150
    <                       proc,
    <                       func:  (case pfdeckind: declkind of
    ---
    >                       proc, func:  (case pfdeckind: declkind of
    154,155c154,155
    <                                           actual: (forwdecl, extern:
    <                                            boolean)))
    ---
    >                                           actual: (forwdecl, extern:
boolean);
    >                                           formal: ()))
```

Pcom has the files prr and prd as standard identifiers. You have to declare them for other compilers:

```
    193d192
    <
    194a194
    >     prr: text; (* comment this out when compiling with pcom *)
    299d298
    <
```

Other compilers don't have the routines mark and release. Their effective semantics are null; you just waste heap:

```
    300a300,301
    >   procedure mark(var p: marktype); begin end;
    >   procedure release(p: marktype); begin end;
    302d302
    <
```

Output the line number with error messages, so that if the listing option has been switched off, you still know which line is in error:

```
    307c307
    <       begin write(output,' ****  ':15);
    ---
    >       begin write(output,linecount:6,' ****  ':9);
```

Accept tabs as white-space as well:

```
    398c398
    <     repeat while (ch = ' ') and not eol do nextch;
    ---
    >     repeat while ((ch = ' ') or (ch = '   ')) and not eol do nextch;
```

Jumping from the then part of an if into the else part is not allowed; fix cases like 1..10 in another way:

```
    429c429
    <           if (ch = '.') or (ch = 'e') then
    ---
    >           if ((ch = '.') and (input^ <> '.')) or (ch = 'e') then
    434c434
    <                       nextch; if ch = '.' then begin ch := ':'; goto 3 end;
    ---
```

```
    >                       nextch; (*if ch = '.' then begin ch := ':'; goto 3
end;*)

Fix modern type mismatches:
    668c668
    <   procedure align(fsp: stp; var flc: integer);
    ---
    >   procedure align(fsp: stp; var flc: addrrange);

An identifier misspelled after the 8th character:
    872c872
    <           if sy = stringconstsy then
    ---
    >           if sy = stringconst then

Unused variables, and new type names:
    1529,1531c1529,1531
    <       var oldlev: 0..maxlevel; lsy: symbol; lcp,lcp1: ctp; lsp: stp;
    <           forw: boolean; oldtop: disprange; parcnt: integer;
    <           llc,lcm: addrrange; lbname: integer; markp: ^integer;
    ---
    >       var oldlev: 0..maxlevel; lcp,lcp1: ctp; lsp: stp;
    >           forw: boolean; oldtop: disprange;
    >           llc,lcm: addrrange; lbname: integer; markp: marktype;
    1535c1535
    <           llc: addrrange; count,lsize: integer;
    ---
    >           llc,lsize: addrrange; count: integer;
    1819c1819
    <           i, entname, segsize: integer;
    ---
    >           entname, segsize: integer;
    2087c2087
    <         var lattr: attr; lcp: ctp; lsize,lmin,lmax: integer;
    ---
    >         var lattr: attr; lcp: ctp; lsize: addrrange; lmin,lmax: integer;
    2248c2248
    <             var lcp:ctp; llev:levrange; laddr:addrrange;
    ---
    >             var llev:levrange; laddr:addrrange;
    2306c2306
    <                 lcp:ctp; llev:levrange; laddr,len:addrrange;
    ---
    >                 llev:levrange; laddr,len:addrrange;
    2456,2457c2456,2457
    <             var lsp,lsp1: stp; varts,lmin,lmax: integer;
    <                 lsize,lsz: addrrange; lval: valu;
    ---
    >             var lsp,lsp1: stp; varts: integer;
    >                 lsize: addrrange; lval: valu;
    2750c2750
    <                     cstpart: set of 0..47; lsp: stp;
    ---
    >                     cstpart: setty; lsp: stp;
```

```
Unix pc can't cope with this line:
    2926c2926
    <             (*/*)     rdiv: begin
    ---
    >             (* / *)   rdiv: begin

More unused variables:
    3318c3318
    <           var lattr: attr; lsp: stp;  lsy: symbol;
    ---
    >           var lattr: attr;  lsy: symbol;
    3642c3642
    <     var sp: stp;
    ---
    >

Produce code as default:
    3800c3800
    <     prtables := false; list := true; prcode := false; debug := true;
    ---
    >     prtables := false; list := true; prcode := true; debug := true;

Unused variable:
    3868c3868
    <       var i: integer; ch: char;
    ---
    >       var i: integer;

Other compilers need to rewrite prr before using it:
    3995,3996c3995,3996
    <   (*compile:*)
    <   (**********)
    ---
    >   (*compile:*) rewrite(prr); (*comment this out when compiling with pcom *)
    >   (**********)

And all variables called 'extern' have been renamed to 'externl'.

Differences in the interpreter are minimal:

a set type has been added:
    45a46
    >       settype     = set of 0..58;
    63c64
    <                                 sett       :(vs :set of 0..47);
    ---
    >                                 sett       :(vs :settype);
    225c226
    <       var name :alfa;  b :boolean;  r :real;  s :set of 0..58;
    ---
    >       var name :alfa;  b :boolean;  r :real;  s :settype;

the initial writeln has been removed:
    667c667
    <   writeln(output); (* for testing *)
```

```
    ---
    >   (* writeln(output); for testing *)

and the type alfa has been added:
    47c47
    <
    ---
    >       alfa      = packed array[1..10] of char;
```

1.4 Bug fixes and improvements to Pascal-P4

Over the years, I discovered several issues and made several improvements in Pascal-P4. These are all made as part of the Pascal-P5 project. Pascal-P5 both fixes issues with Pascal-P4 and fills out the implementation to full implementation of Pascal. I don't plan to introduce any bug fixes to Pascal-P4.

2 Using Pascal-P4

2.1 Configuring P4

P4 has a simple configuration script to set up the binary, script files and compiler in use for the system:

$ Configure [options]…

When configure is run, it attempts to determine automatically what it needs in the host environment, and outputs errors if it cannot find the correct support. Normally configure does not need options to configure it.

The following options are available:

--help	Outputs a help guide to available options.
--gpc	Selects GPC as the host compiler.
--ip_pascal	Selects IP Pascal as the host compiler
--32	Configures for 32 bit operation.
--64	Configures for 64 bit operation.

You should know what you are doing before overriding any options. For example, setting 64 bit mode with a 32 bit host system will cause Pascal-P4 to malfunction.

2.2 Compiling and running Pascal programs with P4

To simply compile a run a program, use the P4 batch file:

```
C:\> p4 hello
```

When a pascal program is run this way, it gets it's input from the terminal (you), and prints its results there. The p4 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.

Pascal-P4 has an issue with input in that it tries to read from the console before the program runs (this issue was solved in Pascal-P5 with "lazy I/O"). For this reason, you often have to hit carriage return just to start the program.

If you were expecting P4 to look like UCSD Pascal or Borland Pascal, please note you took a wrong turn somewhere. P4 is the original Pascal language. The "Pascal" languages processed by UCSD and Borland were heavily modified, and very incompatible variants that that were brought out years after the original.

All files in P4 are anonymous, and only last the length of the program run. The exceptions to this are the "prd" and "prr" files, which are used by the P4 compiler to compile and run itself. You can use them, but you really have to know what you are doing. If you need to read from a file or write to a file use redirection:

```
C:\> p4 test < myinputfile > myoutputfile
```

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

Note:

P4 was designed to be a Pascal compiler porting tool and model implementation ***first***, and not really as a practical day to day compiler.

2.3 Compiler options

P4 uses a "compiler comment" to indicate options to the compiler, of the form:

(*$option+/-,...*)

This option can appear anywhere a normal comment can. The first character of the comment MUST be "$". This is followed by any number if option switches separated by ",". If the option end with "+", it means to turn it on. If the option ends with "-", it means turn it off.

Example:

(*$l-*)

Turns the listing of the source code OFF.

The following options are available:

Option	Meaning	Default
t+/-	Print/don't print internal tables after each routine is compiled.	OFF
l+/-	List/don't list the source program during compilation.	ON
d+/-	Add extra code to check array bounds, subranges, etc.	ON
c+/-	Output/don't output intermediate code.	ON

2.4 Other operations

Within the P5 toolset, you will find a series of scripts to perform common operations using P4. This includes building the compiler and interpreter using an existing ISO 7185 compatible compiler, and also testing P5.

The scripts used in P4 are designed to be independent of what operating system you are running on. The Pascal-P4 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:

~/p4$ ./p4 hello

2.5 Reliance on Unix commands in the P4 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.

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.

2.6 The “flip” command and line endings

Every effort was made to make the Pascal-P4 system 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 \bin directory.

Note that if you are using GIT to retrieve the P5 project, the file entries are given the "OS specific" line ending property. This means both that the line endings will be converted to the line endings particular to your OS, and also prevents line endings from causing GIT to think the file has changed.

3 Building the Pascal-P4 system

3.1 Compiling and running P4 with an existing ISO 7185 compiler

You do not need to compile P4 unless you are using an alternative compiler or installation. The current P4 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. See the Pascal-P5 project for details on doing this.

To compile pcom and pint, the components of the P4 compiler, use the command:

> make

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

P4.bat	The single program compile and run batch file.
compile.bat	To compile a file with all inputs and outputs specified.
run.bat	To run (interpret) the intermediate file with all inputs and outputs specified.

The reason you need to change these files is because pcom.pas uses the header file "prr" to output intermediate code, and pint.pas uses "prd" for input and "prd" for output. You need to find out how to connect these files in the program header to external named files.

For example, in IP Pascal, header files that don't bear a standard system name (like "input" and "output") are simply assigned in order from the command line. Thus, P4.bat is simply:

```
pcom %1.p5 < %1.pas
pint %1.p5 %1.out
```

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

P4.bat lets the input and output from the running program go to the user terminal. Compile.bat and run.bat both specify all of the input, output, prd and prr files. The reason the second files are needed is so that the advanced automated tests can be run using batch files that aren't dependent on what compiler you are using.

3.2 Notes on using existing compilers

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 P4 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 P4 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 P4, and that the resulting P4 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 2.5 "Reliance on Unix commands in the P4 toolset").

Now you will be able to follow the normal gpc instructions here to get p4 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 P4 binaries.
$ regress	Run the regression suites to check the P4 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:

$ make flip	Create flip.exe because msys does not have one. Note that you will need to move the resulting flip.exe to your bin directory.
$ configure --gpc	Configure for GPC compiler.
$ make	Build the P4 binaries.
$ regress	Run the regression suites to check the P4 compiler.

Note: The bash command shell does not work with programs generated by GPC. It gives and error when executing them. Thus it is necessary to use the standard command shell in conjuction with Mingw utilities.

4 Files in the P4 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.bat configure	Sets the current compiler to use to create P4 binaries.
INSTALL	How to install the package.
LICENSE	About the license terms for this package.
Makefile	The makefile for the project.
NEWS	Contains various information about the current release.
README	Brief introduction to the project, it points to this document now.
Setpath	
Setpath.bat	Sets the bin path to find executables.
TODO	List of action items on project.

4.1 Directory: bin

compile compile.bat	Batch mode compile for P5. It takes all input and output from supplied files, and is used by all of the other testing scripts below. You will need to change this to fit your particular Pascal implementation. *** 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.
diffnole	

diffnole.bat	Runs a diff, but ignoring line endings (DOS/Windows vs. Unix).
doseol	
doseol.bat	Fixes the line endings on text files to match the DOS/Windows convention, CRLF.
flip.exe	changes text files between DOS and Unix mode line endings.
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.
flip.exe	Program to fix line endings in source files.
P4	
P4.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.
pcom	
pcom.exe	The IP Pascal compiled pcom binary for Windows/Unix. See comments in 2.2 “Compiling and running Pascal programs with P” for how to use this. All of the supplied batch files are customized for this version.
pint	
pint.exe	The IP Pascal compiled pint binary for Windows. See comments in 2.2 “Compiling and running Pascal programs with P” for how to use this. All of the supplied batch files are customized for this version.
regress	
regress.bat	The regression test simply runs all of the possible tests through P5. It is usually run after a new compile of P5, or any changes made to P5.
run	
run.bat	Batch mode run for P5. It takes all input and output from supplied files, and is used by all of the other testing scripts below. You will need to change this to fit your particular Pascal implementation. *** 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.
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 following program files for p5: hello, roman, match, startrek, basics and iso7185pat.
unixeol unixeol.bat	Fixes the line endings on text files to match the Unix convention, LF.

4.2 Directory: 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

the_p4_compiler.doc the_p4_compiler.docx the_p4_compiler.html the_p4_compiler.pdf	This document in various forms, word 2007, word 1997, PDF, and HTML.

4.4 Directory: gpc

This directory contains scripts specifically modified for GPC.

compile compile.bat	The GPC specific version of the compile script.
cpcom cpcom.bat	The GPC specific version of the compile compiler script.
cpint cpint.bat	The GPC specific version of the compile interpreter script.
P4 P4.bat	The GPC specific version of the p5 script.
run run.bat	The GPC specific version of the run script.

4.5 Directory: gpc/linux_X86

pcom pint	Contains binaries compiled by GPC for Linux/Ubuntu

4.6 Directory: gpc/standard_tests

standardp.pas	Contains the compare file for the standard test.
standardp.cmp	Contains the compare file for the standard test.
Standard.inp	Input file for the standard test.

4.7 Directory: gpc/windows_X86

pcom.exe
pint.exe — Contains binaries compiled by GPC for Windows.

4.8 Directory: ip_pascal

This directory contains scripts specifically modified for IP Pascal.

compile
compile.bat — The IP Pascal specific version of the compile script.

P4
P4.bat — The IP Pascal specific version of the p4 script.

run.bat
run — The IP Pascal specific version of the run script.

4.9 Directory: ip_pascal/standard_tests

iso7185pat.cmp — Contains the compare file for iso7185pat for IP Pascal.

iso7185pats.cmp — Contains the compare file for iso7185pats for IP Pascal.

4.10 Directory: ip_pascal/windows_X86

pcom.exe
pint.exe — Contains binaries compiled by IP Pascal for Windows

4.11 Subdirectory: sample_programs

hello.pas — One of several test programs used to prove the P4 system. This is the standard "hello, world" program.

hello.inp — Input to hello for automated testing.

hello.cmp — Hello compare file for automated testing.

qsort.pas — Demonstrate the quicksort algorithim.

qsort.inp — Input to qsort for automated testing.

qsort.cmp — qsort compare file for automated testing.

roman.pas — A slightly more complex test program, prints roman numerals. From Niklaus Wirth's "User Manual and Report".

roman.inp — Input file for roman automated testing.

roman.cmp — Compare file for roman automated testing.

4.12 Directory: source

pcom.pas — The compiler source in Pascal.

pcomOrg.pas — The original Steve Pemberton compiler source.

pint.pas	The interpreter source in Pascal.
pintOrg.pas	The original Steve Pemberton interpreter source.

4.13 Directory: standard_tests

standardp.cmp	Contains the output from the PAT file for pass/fail comparison.
standardp.inp	The input file for the Pascal acceptance test.
standardp.pas	The Pascal Acceptance Test. This is a single Pascal source that tests how well a given Pascal implementation obeys the P4 subset of Pascal.