Pascal implementation

The P4 Compiler

Scott A. Moore

[1 Overview of Pascal-P5 7](#_Toc397026154)

[1.1 Introduction 7](#_Toc397026155)

[1.2 Why a P5 compiler? 8](#_Toc397026156)

[1.3 The organization of the Pascal-P compiler 11](#_Toc397026157)

[1.4 P5 as a practical compiler 12](#_Toc397026158)

[1.5 P5 as a model compiler 12](#_Toc397026159)

[2 Using Pascal-P5 13](#_Toc397026160)

[2.1 Configuring P5 13](#_Toc397026161)

[2.2 Compiling and running Pascal programs with P5 13](#_Toc397026162)

[2.3 Compiler options 14](#_Toc397026163)

[2.4 Other operations 14](#_Toc397026164)

[2.5 Reliance on Unix commands in the P5 toolset 15](#_Toc397026165)

[2.6 The “flip” command and line endings 15](#_Toc397026166)

[3 Building the Pascal-P5 system 16](#_Toc397026167)

[3.1 Compiling and running P5 with an existing ISO 7185 compiler 16](#_Toc397026168)

[3.2 Evaluating an existing Pascal compiler using P5 17](#_Toc397026169)

[3.3 Notes on using existing compilers 18](#_Toc397026170)

[3.3.1 GPC 18](#_Toc397026171)

[3.3.1.1 GPC on Cygwin 19](#_Toc397026172)

[3.3.1.2 GPC for mingw 19](#_Toc397026173)

[4 Files in the P5 package 20](#_Toc397026174)

[4.1 Directory: gpc 23](#_Toc397026175)

[4.2 Directory: gpc/linux\_X86 23](#_Toc397026176)

[4.3 Directory: mac\_X86 23](#_Toc397026177)

[4.4 Directory: gpc/standard\_tests 23](#_Toc397026178)

[4.5 Directory: gpc/windows\_X86 24](#_Toc397026179)

[4.6 Directory: ip\_pascal 24](#_Toc397026180)

[4.7 Directory: ip\_pascal/standard\_tests 24](#_Toc397026181)

[4.8 Directory: ip\_pascal/windows\_X86 24](#_Toc397026182)

[4.9 Subdirectory: sample\_programs 24](#_Toc397026183)

[4.10 Directory: standard\_tests 25](#_Toc397026184)

[5 Differences between Pascal-P4 and Pascal-P5 25](#_Toc397026185)

[5.1 Viewing changes 25](#_Toc397026186)

[5.2 Notes about change descriptions 26](#_Toc397026187)

[5.3 Changes to the parser 26](#_Toc397026188)

[5.3.1 Thematic changes 26](#_Toc397026189)

[5.3.1.1 Variable strings 26](#_Toc397026190)

[5.3.1.2 Recycling based on dispose 26](#_Toc397026191)

[5.3.1.3 Files 27](#_Toc397026192)

[5.3.1.4 Byte oriented pseudo-machine 27](#_Toc397026193)

[5.3.2 Reading the source code 28](#_Toc397026194)

[5.3.2.1 Exit label 28](#_Toc397026195)

[5.3.2.2 The machine parameter block 28](#_Toc397026196)

[5.3.2.3 Other constants 28](#_Toc397026197)

[5.3.2.4 Types 29](#_Toc397026198)

[5.3.2.5 Variables 31](#_Toc397026199)

[5.3.2.6 Procedures and functions 33](#_Toc397026200)

[5.3.2.7 Recycling support routines 36](#_Toc397026201)

[5.3.2.7.1 Procedure getstr 36](#_Toc397026202)

[5.3.2.7.2 Procedure putstrs 36](#_Toc397026203)

[5.3.2.7.3 procedure getlab 36](#_Toc397026204)

[5.3.2.7.4 procedure putlab 36](#_Toc397026205)

[5.3.2.7.5 procedure pshcst 37](#_Toc397026206)

[5.3.2.7.6 procedure putcst 37](#_Toc397026207)

[5.3.2.7.7 procedure pshstc 37](#_Toc397026208)

[5.3.2.7.8 procedure putstc 37](#_Toc397026209)

[5.3.2.7.9 procedure ininam 37](#_Toc397026210)

[5.3.2.7.10 procedure putnam 37](#_Toc397026211)

[5.3.2.7.11 procedure putnams 37](#_Toc397026212)

[5.3.2.7.12 procedure putdsp 37](#_Toc397026213)

[5.3.2.7.13 procedure putdsps 38](#_Toc397026214)

[5.3.2.7.14 procedure getfil 38](#_Toc397026215)

[5.3.3 procedure putfil 38](#_Toc397026216)

[5.3.3.1.1 procedure getcas 38](#_Toc397026217)

[5.3.3.1.2 procedure putcas 38](#_Toc397026218)

[5.3.3.2 Character and string quata routines 38](#_Toc397026219)

[5.3.3.2.1 function lcase 39](#_Toc397026220)

[5.3.3.2.2 procedure lcases 39](#_Toc397026221)

[5.3.3.2.3 function strequri 39](#_Toc397026222)

[5.3.3.2.4 procedure writev 39](#_Toc397026223)

[5.3.3.2.5 function lenpv 40](#_Toc397026224)

[5.3.3.2.6 procedure strassvf 40](#_Toc397026225)

[5.3.3.2.7 procedure strassvr 40](#_Toc397026226)

[5.3.3.2.8 procedure strassvd 40](#_Toc397026227)

[5.3.3.2.9 procedure strassvc 40](#_Toc397026228)

[5.3.3.2.10 function strequvv 40](#_Toc397026229)

[5.3.3.2.11 function strltnvv 40](#_Toc397026230)

[5.3.3.2.12 function strequvf 40](#_Toc397026231)

[5.3.3.2.13 function strltnvf 40](#_Toc397026232)

[5.3.3.2.14 Function strchr 40](#_Toc397026233)

[5.3.3.2.15 Procedure strchrass 41](#_Toc397026234)

[5.3.3.2.16 procedure prtdsp 41](#_Toc397026235)

[5.3.4 Modifications 41](#_Toc397026236)

[5.3.4.1 procedure endofline 41](#_Toc397026237)

[5.3.4.2 procedure errmsg 41](#_Toc397026238)

[5.3.4.3 procedure error 41](#_Toc397026239)

[5.3.4.4 procedure insymbol 42](#_Toc397026240)

[5.3.4.4.1 Letter: identifiers and reserved words 42](#_Toc397026241)

[5.3.4.4.2 Number: integers and reals 42](#_Toc397026242)

[5.3.4.4.3 Chstrquo: Strings 43](#_Toc397026243)

[5.3.4.4.4 Chperiod: ‘.’, ‘..’ and ‘.)’ tolkens 43](#_Toc397026244)

[5.3.4.4.5 Chlparen: ‘(‘and comment start 43](#_Toc397026245)

[5.3.4.4.6 Chlcmt: ISO 7185 comment start 43](#_Toc397026246)

[5.3.4.4.7 Lexical dump 43](#_Toc397026247)

[5.3.4.5 procedure enterid 43](#_Toc397026248)

[5.3.4.6 procedure searchsection 44](#_Toc397026249)

[5.3.4.7 procedure searchidne 44](#_Toc397026250)

[5.3.4.8 procedure searchid 44](#_Toc397026251)

[5.3.4.9 procedure printtables 44](#_Toc397026252)

[5.3.4.10 procedure followstp 44](#_Toc397026253)

[5.3.4.11 procedure followctp 44](#_Toc397026254)

[5.3.4.12 procedure searchlabel 44](#_Toc397026255)

[5.3.4.13 procedure newlabel 44](#_Toc397026256)

[5.3.4.14 procedure prtlabels 45](#_Toc397026257)

[5.3.4.15 procedure block 45](#_Toc397026258)

[5.3.4.16 procedure constant 45](#_Toc397026259)

[5.3.4.17 function string 46](#_Toc397026260)

[5.3.4.18 function comptypes 46](#_Toc397026261)

[5.3.4.19 function filecomponent 48](#_Toc397026262)

[5.3.4.20 function string 48](#_Toc397026263)

[5.3.4.21 procedure typ 48](#_Toc397026264)

[5.3.4.22 procedure simpletype 48](#_Toc397026265)

[5.3.4.23 procedure fieldlist 48](#_Toc397026266)

[5.3.4.24 procedure labeldeclaration 48](#_Toc397026267)

[5.3.4.25 procedure constdeclaration 48](#_Toc397026268)

[5.3.4.26 procedure typedeclaration 48](#_Toc397026269)

[5.3.4.27 procedure vardeclaration 48](#_Toc397026270)

[5.3.4.28 procedure procdeclaration 48](#_Toc397026271)

[5.3.4.29 procedure pushlvl 48](#_Toc397026272)

[5.3.4.30 procedure parameterlist 48](#_Toc397026273)

[5.3.4.31 procedure body 48](#_Toc397026274)

[5.3.4.32 procedure addlvl 48](#_Toc397026275)

[5.3.4.33 procedure sublvl 48](#_Toc397026276)

[5.3.4.34 procedure mes 48](#_Toc397026277)

[5.3.4.35 procedure putic 48](#_Toc397026278)

[5.3.4.36 procedure gen0 48](#_Toc397026279)

[5.3.4.37 procedure gen1 48](#_Toc397026280)

[5.3.4.38 procedure gen2 48](#_Toc397026281)

[5.3.4.39 procedure gentypindicator 48](#_Toc397026282)

[5.3.4.40 procedure gen0t 48](#_Toc397026283)

[5.3.4.41 procedure gen1t 48](#_Toc397026284)

[5.3.4.42 procedure gen2t 48](#_Toc397026285)

[5.3.4.43 procedure load 48](#_Toc397026286)

[5.3.4.44 procedure store 48](#_Toc397026287)

[5.3.4.45 procedure loadaddress 48](#_Toc397026288)

[5.3.4.46 procedure genfjp 48](#_Toc397026289)

[5.3.4.47 procedure genujpxjp 49](#_Toc397026290)

[5.3.4.48 procedure genipj 49](#_Toc397026291)

[5.3.4.49 procedure gencupent 49](#_Toc397026292)

[5.3.4.50 procedure genlpa 49](#_Toc397026293)

[5.3.4.51 procedure checkbnds 49](#_Toc397026294)

[5.3.4.52 procedure putlabel 49](#_Toc397026295)

[5.3.4.53 procedure statement 49](#_Toc397026296)

[5.3.4.54 procedure expression 49](#_Toc397026297)

[5.3.4.55 procedure selector 49](#_Toc397026298)

[5.3.4.56 procedure call 49](#_Toc397026299)

[5.3.4.57 procedure variable 49](#_Toc397026300)

[5.3.4.58 procedure getputresetrewriteprocedure 49](#_Toc397026301)

[5.3.4.59 procedure pageprocedure 49](#_Toc397026302)

[5.3.4.60 procedure readprocedure 49](#_Toc397026303)

[5.3.4.61 procedure writeprocedure 49](#_Toc397026304)

[5.3.4.62 procedure packprocedure 49](#_Toc397026305)

[5.3.4.63 procedure unpackprocedure 49](#_Toc397026306)

[5.3.4.64 procedure newdisposeprocedure 49](#_Toc397026307)

[5.3.4.65 procedure absfunction 49](#_Toc397026308)

[5.3.4.66 procedure sqrfunction 49](#_Toc397026309)

[5.3.4.67 procedure truncfunction 49](#_Toc397026310)

[5.3.4.68 procedure roundfunction 49](#_Toc397026311)

[5.3.4.69 procedure oddfunction 49](#_Toc397026312)

[5.3.4.70 procedure ordfunction 49](#_Toc397026313)

[5.3.4.71 procedure chrfunction 49](#_Toc397026314)

[5.3.4.72 procedure predsuccfunction 49](#_Toc397026315)

[5.3.4.73 procedure eofeolnfunction 49](#_Toc397026316)

[5.3.4.74 procedure callnonstandard 49](#_Toc397026317)

[5.3.4.75 procedure compparam 50](#_Toc397026318)

[5.3.4.76 procedure expression 50](#_Toc397026319)

[5.3.4.77 procedure simpleexpression 50](#_Toc397026320)

[5.3.4.78 procedure term 50](#_Toc397026321)

[5.3.4.79 procedure factor 50](#_Toc397026322)

[5.3.4.80 procedure assignment 50](#_Toc397026323)

[5.3.4.81 procedure gotostatement 50](#_Toc397026324)

[5.3.4.82 procedure compoundstatement 50](#_Toc397026325)

[5.3.4.83 procedure ifstatement 50](#_Toc397026326)

[5.3.4.84 procedure casestatement 50](#_Toc397026327)

[5.3.4.85 procedure repeatstatement 50](#_Toc397026328)

[5.3.4.86 procedure whilestatement 50](#_Toc397026329)

[5.3.4.87 procedure forstatement 50](#_Toc397026330)

[5.3.4.88 procedure withstatement 50](#_Toc397026331)

[5.3.4.89 procedure programme 50](#_Toc397026332)

[5.3.4.90 procedure entstdnames 50](#_Toc397026333)

[5.3.4.91 procedure enterundecl 50](#_Toc397026334)

[5.3.4.92 procedure exitundecl 50](#_Toc397026335)

[5.3.4.93 procedure initscalars 50](#_Toc397026336)

[5.3.4.94 procedure initsets 50](#_Toc397026337)

[5.3.4.95 procedure inittables 50](#_Toc397026338)

[5.3.4.96 procedure reswords 50](#_Toc397026339)

[5.3.4.97 procedure symbols 50](#_Toc397026340)

[5.3.4.98 procedure rators 50](#_Toc397026341)

[5.3.4.99 procedure procmnemonics 50](#_Toc397026342)

[5.3.4.100 procedure instrmnemonics 50](#_Toc397026343)

[5.3.4.101 procedure chartypes 50](#_Toc397026344)

[5.3.4.102 procedure initdx 50](#_Toc397026345)

[6 Changes to the assembler/interpreter 51](#_Toc397026346)

[7 The intermediate language 51](#_Toc397026347)

[7.1 Format of intermediate 51](#_Toc397026348)

[7.1.1 Label 52](#_Toc397026349)

[7.1.2 Source line marker 52](#_Toc397026350)

[7.2 Intermediate instruction set 52](#_Toc397026351)

[7.3 System calls 66](#_Toc397026352)

[8 Testing P5 73](#_Toc397026353)

[8.1 Running tests 73](#_Toc397026354)

[8.1.1 testprog 73](#_Toc397026355)

[8.1.2 Other tests 75](#_Toc397026356)

[8.1.3 Regression test 75](#_Toc397026357)

[8.2 Test types 75](#_Toc397026358)

[8.3 The Pascal acceptance test 75](#_Toc397026359)

[8.4 The Pascal rejection test 75](#_Toc397026360)

[8.4.1 List of tests 76](#_Toc397026361)

[8.4.1.1 Class 1: Syntatic errors 76](#_Toc397026362)

[8.4.1.2 Class 2: Semantic errors 82](#_Toc397026363)

[8.4.2 Running the PRT and interpreting the results 85](#_Toc397026364)

[8.4.2.1 List of tests with no compile or runtime error. 86](#_Toc397026365)

[8.4.2.2 List of differences between compiler output and “gold” standard outputs. 86](#_Toc397026366)

[8.4.2.3 List of differences between runtime output and “gold” standard outputs. 86](#_Toc397026367)

[8.4.2.4 Collected compiler listings and runtime output of all tests. 86](#_Toc397026368)

[8.4.3 Overall interpretation of PRT results 87](#_Toc397026369)

[8.5 Sample program tests 87](#_Toc397026370)

[8.6 Previous Pascal-P versions test 87](#_Toc397026371)

[8.6.1 Compile and run Pascal-P2 87](#_Toc397026372)

[8.6.2 Compile and run Pascal-P4 88](#_Toc397026373)

[8.7 Self compile 88](#_Toc397026374)

[8.7.1 pcom 89](#_Toc397026375)

[8.7.1.1 Changes required 89](#_Toc397026376)

[8.7.2 pint 89](#_Toc397026377)

# 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-P5”

## 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/>.

## 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.

## 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.

# Using Pascal-P4

## Configuring P4

P5 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.

## 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 p5 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:\> p5 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.

## Compiler options

P5 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 |

## 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 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

## Reliance on Unix commands in the P5 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.

## The “flip” command and line endings

Every effort was made to make the Pascal-P5 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.

# Building the Pascal-P4 system

## 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 “Tesing P5” section 18 for details on how to test an existing compiler to ISO 7185 standards.

To compile pcom and pint, the components of the P5 compiler, use the command:

> make

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

p5.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, P5.bat is simply:

pcom %1.p5 < %1.pas

pint %1.p5 %1.out

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

P5.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.

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.

## Notes on using existing compilers

### 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 P5, and that the resulting P5 runs the compliance tests. I leave it for others to run full compliance for GPC itself.

#### 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](http://gnu-pascal.de/contrib/chief/win32/cygwin/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)

1. 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.
2. Add cygwin=nodosfilewarning (as stated in section 15 “Reliance on Unix commands in the P5 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.

#### 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](http://gnu-pascal.de/contrib/chief/win32/cygwin/gpc-20070904-with-gcc.i686-pc-cygwin.tar.gz)

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

1. 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 P5 binaries.

$ regress Run the regression suites to check the P5 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.

# 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.

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.

configure.bat

configure Sets the current compiler to use to create P5 binaries.

cpcoms

cpcoms.bat Self compile, run and check the pcom.pas file. This batch file compiles com.pas, then runs it on the interpreter and self compiles it, and checks the intermediate files match.

cpints

cpints.bat Self compile, run and check the pint.pas file. This batch file compiles pint.pas and iso7185pat.pas, then runs pint on itself and then runs iso7185pat.pas, and checks the result file.

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.

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.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.

iso7185.html

iso7185.pdf The full ISO 7185 Pascal standard, in html and pdf forms.

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

NEWS Contains various information about the current release.

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 12 “Compiling and running Pascal programs with P5” for how to use this. All of the supplied batch files are customized for this version.

pcom.pas The compiler source in Pascal.

pint.exe The IP Pascal compiled pint binary for Windows. See comments in 12 “Compiling and running Pascal programs with P5” for how to use this. All of the supplied batch files are customized for this version.

pint.pas The interpreter source in Pascal

README Brief introduction to the project, it points to this document now.

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.

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.

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].

TODO Contain a list of "to do" items in P5.

unixeol

unixeol.bat Fixes the line endings on text files to match the Unix convention, LF.

Zipp4.bat Creates a zipfile for the entire p5 project. This is used to create the releases available on the p5 web site.

## 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.

## Directory: gpc/linux\_X86

pcom

pint Contains binaries compiled by GPC for Linux/Ubuntu

## Directory: mac\_X86

A placeholder for Mac OS X binaries.

## 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.

## Directory: gpc/windows\_X86

pcom.exe

pint.exe Contains binaries compiled by GPC for Windows.

## 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 p5 script.

run.bat

run The IP Pascal specific version of the run script.

## 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.

## Directory: ip\_pascal/windows\_X86

pcom.exe

pint.exe Contains binaries compiled by IP Pascal for Windows

## Subdirectory: sample\_programs

basics.pas A tiny basic interpreter in Pascal.

basics.inp Input test file for basics. In fact, it is a basic verion of "match" above.

basics.cmp Compare file for basics.

hello.pas One of several test programs used to prove the P5 system. This is the standard "hello, world" program.

hello.inp Input to hello for automated testing.

hello.cmp Hello compare file for automated testing.

match.bas The basic version of the match game.

match.pas A game, place "match" a number game.

match.cmp Compare file for match automated testing.

match.inp Input file for match automated testing.

pascals.pas Niklaus Wirth’s Pascal-s subset interpreter..

pascals.cmp Compare file for pascals automated testing.

pascals.inp Input file for pascals 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.

startrek.pas The startrek game.

startrek.inp Startrek input file.

startrek.cmp Startrek compare file.

## Directory: standard\_tests

iso7185pat.cmp Contains the output from the PAT file with the IP Pascal compiled P5 executables above.

iso7185pat.inp The input file for the Pascal acceptance test.

iso7185pat.pas The Pascal Acceptance Test. This is a single Pascal source that tests how well a given Pascal implementation obeys ISO 7185 Pascal. It can be used on P5 or any other Pascal implementation.

iso7185pats.cmp Contains the output from the PAT file resulting from the cpints run. This is slightly different than the normal run.

iso7185prt.bat Compiles the Pascal rejection tests.

iso7185prtXXXX.cmp Comparision file for pascal rejection test. XXXX is a four digit number. See the rejection tests text for information.

Iso7185prtXXXX.inp Contains the input file for the Pascal rejection test XXXX.

Iso7185prtXXXX.pas Contains the source file for the Pascal rejection test XXXX.

# Changes to the assembler/interpreter

[tbd]