Pascal (programming language)

Pascal

Paradigm(s)	imperative, structured
Designed by	Niklaus Wirth
Appeared in	1970
Typing discipline	static, strong, safe
Major implementations	CDC 6000, ICL 1900, Pascal-P, PDP-11, PDP-10, IBM System/370, HP, Free Pascal, GNU Pascal
Dialects	Borland, Turbo Pascal, UCSD Pascal
Influenced by	ALGOL W
Influenced	Ada, Component Pascal, Java,[1][2] Modula, Modula-2, Modula-3, Oberon, Oberon-2, Object Pascal, Oxygene, Seed7

Pascal is an influential imperative and procedural programming language, designed in 1968–1969 and published in 1970 by Niklaus Wirth as a small and efficient language intended to encourage good programming practices using structured programming and data structuring.

A derivative known as Object Pascal designed for object-oriented programming was developed in 1985.

History

Pascal, named in honor of the French mathematician and philosopher Blaise Pascal, was developed by Niklaus Wirth.

Before his work on Pascal, Wirth had developed Euler and ALGOL W and later went on to develop the Pascal-like languages Modula-2 and Oberon.

Initially, Pascal was largely, but not exclusively, intended to teach students structured programming.[3] A generation of students used Pascal as an introductory language in undergraduate courses. Variants of Pascal have also frequently been used for everything from research projects to PC games and embedded systems. Newer Pascal compilers exist which are widely used.[4]

Pascal was the primary high-level language used for development in the Apple Lisa, and in the early years of the Macintosh. Parts of the original Macintosh operating system were hand-translated into Motorola 68000 assembly language from the Pascal sources.[5] The popular typesetting system TeX by Donald E. Knuth was written in WEB, the original literate programming system, based on DEC PDP-10 Pascal, while applications like Total Commander, Skype and Macromedia Captivate were written in Delphi (Object Pascal).

Object Pascal (Embarcadero Delphi) is still used for developing Windows applications but also has the ability to cross compile the same code to Mac and iOS. Another cross-platform version called Free Pascal, with the Lazarus IDE, is popular with Linux users since it also promises write once, compile anywhere, development. CodeTyphon is a variant of Lazarus with more preinstalled packages and cross compilers.

Brief description

Wirth's intention was to create an efficient language (regarding both compilation speed and generated code) based on so-called structured programming, a concept which had recently become popular. Pascal has its roots in the ALGOL 60 language, but also introduced concepts and mechanisms which (on top of ALGOL's scalars and arrays) enabled programmers to define their own complex (structured) datatypes, and also made it easier to build dynamic and recursive data structures such as *lists*, *trees* and *graphs*. Important features included for this were *records*, *enumerations*, *subranges*, *dynamically* allocated variables with associated *pointers*, and *sets*. To make this possible and meaningful, Pascal has a strong typing on all objects, which means that one type of data cannot be converted or interpreted as another without *explicit* conversions. Similar mechanisms are standard in many programming languages today. Other languages that influenced Pascal's development were COBOL, Simula 67, and Wirth's own ALGOL W.

Pascal, like many programming languages of today (but unlike most languages in the C family), allows nested procedure definitions to any level of depth, and also allows most kinds of definitions and declarations inside procedures and functions. This enables a very simple and coherent syntax where a complete **program** is syntactically nearly identical to a single **procedure** or **function** (except for the keyword itself, of course.)

Implementations

Early Pascal compilers

The first Pascal compiler was designed in Zürich for the CDC 6000 series mainframe computer family. Niklaus Wirth reports that a first attempt to implement it in Fortran in 1969 was unsuccessful due to Fortran's inadequacy to express complex data structures. The second attempt was formulated in the Pascal language itself and was operational by mid-1970. Many Pascal compilers since have been similarly self-hosting, that is, the compiler is itself written in Pascal, and the compiler is usually capable of recompiling itself when new features are added to the language, or when the compiler is to be ported to a new environment. The GNU Pascal compiler is one notable exception, being written in C.

The first successful port of the CDC Pascal compiler to another mainframe was completed by Welsh and Quinn at the Queen's University of Belfast (QUB) in 1972. The target was the ICL 1900 series. This compiler in turn was the parent of the Pascal compiler for the ICS Multum minicomputer. The Multum port was developed – with a view to using Pascal as a systems programming language – by Findlay, Cupples, Cavouras and Davis, working at the Department of Computing Science in Glasgow University. It is thought that Multum Pascal, which was completed in the summer of 1973, may have been the first 16-bit implementation.

A completely new compiler was completed by Welsh et al. at QUB in 1977. It offered a source-language diagnostic feature (incorporating profiling, tracing and type-aware formatted postmortem dumps) that was implemented by Findlay and Watt at Glasgow University. This implementation was ported in 1980 to the ICL 2900 series by a team based at Southampton University and Glasgow University. The Standard Pascal Model Implementation was also based on this compiler, having been adapted, by Welsh and Hay at Manchester University in 1984, to check rigorously for conformity to the BSI 6192/ISO 7185 Standard and to generate code for a portable abstract machine.

The first Pascal compiler written in North America was constructed at the University of Illinois under Donald B. Gillies for the PDP-11 and generated native machine code.

The Pascal-P system

To propagate the language rapidly, a compiler "porting kit" was created in Zurich that included a compiler that generated code for a "virtual" stack machine, *i.e.*, code that lends itself to reasonably efficient interpretation, along with an interpreter for that code – the *Pascal-P* system. The P-system compilers were termed Pascal-P1, Pascal-P2, Pascal-P3, and Pascal-P4. Pascal-P1 was the first version, and Pascal-P4 was the last to come from Zurich.

The Pascal-P4 compiler/interpreter can still be run and compiled on systems compatible with original Pascal. However, it only accepts a subset of the Pascal language.

Pascal-P5, created outside of the Zurich group, accepts the full Pascal language and includes ISO 7185 compatibility.

UCSD Pascal branched off Pascal-P2, where Kenneth Bowles utilized it to create the interpretive UCSD p-System. In the early 1980s, UCSD Pascal was ported to the Apple II and Apple III computers to provide a structured alternative to the BASIC interpreters that came with the machines, and the UCSD p-System was one of three operating systems available at the launch of the original IBM PC.[6]

A compiler based on the Pascal-P4 compiler, which created native binaries, was released for the IBM System/370 mainframe computer by the Australian Atomic Energy Commission; it was called the "AAEC Pascal Compiler" after the abbreviation of the name of the Commission.

In the early 1980s, Watcom Pascal was developed, also for the IBM System 370.

IP Pascal was an implementation of the Pascal programming language using Micropolis DOS, but was moved rapidly to CP/M running on the Z80. It was moved to the 80386 machine types in 1994, and exists today as Windows/XP and Linux implementations. In 2008, the system was brought up to a new level and the resulting language termed "Pascaline" (after Pascal's calculator). It includes objects, namespace controls, dynamic arrays, along with many other extensions, and generally features the same functionality and type protection as C#. It is the only such implementation that is also compatible with the original Pascal implementation, which is standardized as ISO 7185.

An example of educational use. Into the 1990s Pascal was still running on VAX terminals at GMU. Pascal books were sold and taught to fulfill the programming requirement.

Object Pascal, Turbo Pascal

Apple Computer created its own Lisa Pascal for the Lisa Workshop in 1982 and ported this compiler to the Apple Macintosh and MPW in 1985. In 1985 Larry Tesler, in consultation with Niklaus Wirth, defined Object Pascal and these extensions were incorporated in both the Lisa Pascal and Mac Pascal compilers.

In the 1980s Anders Hejlsberg wrote the Blue Label Pascal compiler for the Nascom-2. A reimplementation of this compiler for the IBM PC was marketed under the names Compas Pascal and PolyPascal before it was acquired by Borland. Renamed *Turbo Pascal*, it became hugely popular, thanks in part to an aggressive pricing strategy and in part to having one of the first full-screen Integrated development environments, and fast turnaround-time (just seconds to compile, link, and run.) Additionally, it was written and highly optimized entirely in assembly language, making it smaller and faster than much of the competition. In 1986 Anders ported Turbo Pascal to the Macintosh and incorporated Apple's Object Pascal extensions into Turbo Pascal. These extensions were then added back into the PC version of Turbo Pascal for version 5.5. At the same time Microsoft also implemented the Object Pascal compiler.[7][8] Turbo Pascal 5.5 had a large influence on the Pascal community, which began concentrating mainly on the IBM PC in the late 1980s. Many PC hobbyists in search of a structured replacement for BASIC used this product. It also began to be adopted by professional developers. Around the same time a number of concepts were imported from C to let Pascal programmers use the C-based API of Microsoft Windows directly. These extensions included null-terminated strings, pointer arithmetic, function pointers, an address-of operator and unsafe typecasts.

However, Borland later decided it wanted more elaborate object-oriented features, and started over in Delphi using the *Object Pascal* draft standard proposed by Apple as a basis. (This Apple draft is still not a formal standard.) The first versions of the Delphi language were accordingly named Object Pascal. The main additions compared to the older OOP extensions were a reference-based object model, virtual constructors and destructors, and properties. Several other compilers also implement this dialect.

Turbo Pascal, and other derivatives with units or module concepts are modular languages. However, it does not provide a nested module concept or qualified import and export of specific symbols.

Other variants

Super Pascal was a variant that added non-numeric labels, a return statement and expressions as names of types.

The universities of Wisconsin-Madison, Zurich, Karlsruhe and Wuppertal developed the Pascal-SC and Pascal-XSC[9][10][11] (*EXtension for Scientific Computing*) compilers, aimed at programming numerical computations. Pascal-SC originally targeted the Z80 processor, but was later rewritten for DOS (x86) and 68000. Pascal-XSC has at various times been ported to unix (Linux, SunOS, HP-UX, AIX) and Microsoft/IBM (MS-DOS with EMX, OS/2, Windows) operating systems. It operates by generating intermediate C source code which is then compiled to a native executable. Some of the Pascal-SC language extensions have been adopted by GNU Pascal.

Language constructs

Pascal, in its original form, is a purely procedural language and includes the traditional array of ALGOL-like control structures with reserved words such as **if**, **then**, **else**, **while**, **for**, and so on. However, Pascal also has many data structuring facilities and other abstractions which were not included in the original ALGOL 60, like type definitions, records, pointers, enumerations, and sets. Such constructs were in part inherited or inspired from Simula 67, ALGOL 68, Niklaus Wirth's own ALGOL W and suggestions by C. A. R. Hoare.

Hello world

Pascal programs start with the **program** keyword with a list of external file descriptors as parameters[12] (not required in Turbo Pascal etc.); then follows the main block bracketed by the **begin** and **end** keywords. Semicolons separate statements, and the full stop (i.e., a period) ends the whole program (or *unit*). Letter case is ignored in Pascal source.

Here is an example of the source code in use for a very simple "Hello world" program:

```
Program HelloWorld;
Begin
  WriteLn('Hello world!')
  {no ";" is required after the last statement of a block -
   adding one adds a "null statement" to the program}
End.
```

Data types

A type in Pascal, and in several other popular programming languages, defines a variable in such a way that it defines a range of values which the variable is capable of storing, and it also defines a set of operations that are permissible to be performed on variables of that type. The predefined types are:

Data type	Type of values which the variable is capable of storing
integer	integer (whole) numbers
real	floating-point numbers
boolean	the value True or False
char	a single character from an ordered character set
string	a group or "string" of characters

The range of values allowed for each (except boolean) is implementation defined. Functions are provided for some data conversions. For conversion of `real` to `integer`, the following functions are available: `round` (which rounds to integer using banker's rounding) and `trunc` (rounds towards zero).

The programmer has the freedom to define other commonly used data types (e.g. byte, string, etc.) in terms of the predefined types using Pascal's type declaration facility, for example

```
type
  byte        = 0..255;
  signed_byte = -128..127;
  string      = packed array[1..255] of char;
```

(Often-used types like byte and string are already defined in many implementations.)

Subrange types

Subranges of any ordinal data type (any simple type except real) can also be made:

```
var
  x : 1..10;
  y : 'a'..'z';
```

Set types

In contrast with other programming languages from its time, Pascal supports a set type:

```
var
  Set1 : set of 1..10;
  Set2 : set of 'a'..'z';
```

A set is a fundamental concept for modern mathematics, and they may be used in many algorithms. Such a feature is useful and may be faster than an equivalent construct in a language that does not support sets. For example, for many Pascal compilers:

```
if i in [5..10] then ...
```

executes faster Wikipedia:Citation needed than:

```
if (i > 4) and (i < 11) then ...
```

Sets of non-contiguous values can be particularly useful, in terms of both performance and readability:

```
if i in [0..3, 7, 9, 12..15] then ...
```

For these examples, which involve sets over small domains, the improved performance is usually achieved by the compiler representing set variables as bitmasks. The set operators can then be implemented efficiently as bitwise machine code operations.

Type declarations

Types can be defined from other types using type declarations:

```
type
  x = integer;
  y = x;
...
```

Further, complex types can be constructed from simple types:

```
type
  a = array[1..10] of integer;
  b = record
        x : integer;
        y : char
      end;
  c = file of a;
```

File type

As shown in the example above, Pascal files are sequences of components. Every file has a buffer variable which is denoted by *f^*. The procedures *get* (for reading) and *put* (for writing) move the buffer variable to the next element. Read is introduced such that *read(f, x)* is the same as *x := f^; get(f);*. Write is introduced such that *write(f, x)* is the same as *f^ := x; put(f);* The type text is predefined as file of char. While the buffer variable could be used for inspecting the next character to be used (check for a digit before reading an integer), this leads to serious problems with interactive programs in early implementations, but was solved later with the "lazy I/O" concept.

In Jensen & Wirth Pascal, strings are represented as packed arrays of chars; they therefore have fixed length and are usually space-padded. Some dialects have a custom string type; e.g. Delphi 2 has a new string type whose length is stored in a 32-bit value instead of a single byte.

Pointer types

Pascal supports the use of pointers:

```
type
  pNode = ^Node;
  Node  = record
          a : integer;
          b : char;
          c : pNode  {extra semicolon not strictly required}
          end;
var
  NodePtr : pNode;
  IntPtr  : ^integer;
```

Here the variable *NodePtr* is a pointer to the data type *Node*, a record. Pointers can be used before they are declared. This is a forward declaration, an exception to the rule that things must be declared before they are used.

To create a new record and assign the value *10* and character *A* to the fields *a* and *b* in the record, and to initialise the pointer *c* to NIL, the commands would be:

```
New(NodePtr);
...
NodePtr^.a := 10;
NodePtr^.b := 'A';
NodePtr^.c := NIL;
...
```

This could also be done using the **with** statement, as follows:

```
New(NodePtr);
...
with NodePtr^ do
begin
```

```
  a := 10;
  b := 'A';
  c := NIL
end;
...
```

Inside of the scope of the **with** statement, a and b refer to the subfields of the record pointer **NodePtr** and not to the record Node or the pointer type pNode.

Linked lists, stacks and queues can be created by including a pointer type field (c) in the record (see also NIL).

Unlike many languages that feature pointers, Pascal only allows pointers to reference dynamically created variables that are anonymous, and does not allow them to reference standard static or local variables. Pointers also must have an associated type, and a pointer to one type is not compatible with a pointer to another type (e.g. a pointer to a char is not compatible with a pointer to an integer). This helps eliminate the type security issues inherent with other pointer implementations, particularly those used for PL/I or C. It also removes some risks caused by dangling pointers, but the ability to dynamically deallocate referenced space by using the *dispose* function (which has the same effect as the *free* library function found in C) means that the risk of dangling pointers has not been entirely eliminated[13] as it has in languages such as Java and C#, which provide automatic garbage collection (but which do not entirely eliminate the related problem of memory leaks).

Some of these restrictions can be lifted in newer dialects.

Control structures

Pascal is a structured programming language, meaning that the flow of control is structured into standard statements, usually without 'goto' commands.

```
while a <> b do  WriteLn('Waiting');

if a > b then WriteLn('Condition met')    {no semicolon allowed!}
           else WriteLn('Condition not met');

for i := 1 to 10 do  {no semicolon for single statements allowed!}
  WriteLn('Iteration: ', i);

repeat
  a := a + 1
until a = 10;

case i of
  0 : Write('zero');
  1 : Write('one');
  2 : Write('two');
  3,4,5,6,7,8,9,10: Write('?')
end;
```

Procedures and functions

Pascal structures programs into procedures and functions.

```
program Mine(output);

var i : integer;

procedure Print(var j : integer);
begin
...
end;

begin
...
Print(i);
end.
```

Procedures and functions can nest to any depth, and the 'program' construct is the logical outermost block.

Each procedure or function can have its own declarations of goto labels, constants, types, variables, and other procedures and functions, which must all be in that order. This ordering requirement was originally intended to allow efficient single-pass compilation. However, in some dialects (such as Embarcadero Delphi) the strict ordering requirement of declaration sections has been relaxed.

Semicolons as statement separators

Pascal adopted many language syntax features from the ALGOL language, including the use of a semicolon as a statement separator. This is in contrast to other languages, such as PL/I, C etc. which use the semicolon as a statement terminator. As illustrated in the above examples, no semicolon is needed before the **`end`** keyword of a record type declaration, a block, or a *case* statement; before the **`until`** keyword of a repeat statement; and before the **`else`** keyword of an *if* statement.

The presence of an extra semicolon was not permitted in early versions of Pascal. However, the addition of ALGOL-like empty statements in the 1973 *Revised Report* and later changes to the language in ISO 7185:1983 now allow for optional semicolons in most of these cases. A semicolon is still not permitted immediately before the *`else`* keyword in an *if* statement, because the `else` follows a single statement, not a statement sequence. In the case of nested ifs, a semicolon cannot be used to avoid the dangling else problem (where the inner if does not have an else, but the outer if does) by putatively terminating the nested if with a semicolon – this instead terminates both if clauses. Instead, an explicit `begin...end` block must be used.[14]

Programmers usually include these extra semicolons out of habit, and to avoid changing the last line of a statement sequence when new code is appended.

Resources

Compilers and interpreters

Several Pascal compilers and interpreters are available for general use:

- Delphi is Embarcadero's (formerly Borland/CodeGear) flagship rapid application development (RAD) product. It uses the Object Pascal language (termed 'Delphi' by Borland), descended from Pascal, to create applications for the windows platform. The .NET support that existed from D8 through D2005, D2006 and D2007 has been terminated, and replaced by a new language (Prism, which is rebranded Oxygene, see below) that is not fully backwards compatible. In recent years Unicode support and generics were added (D2009, D2010, Delphi XE).
- Free Pascal is a multi-platform compiler written in Object Pascal (and is self-hosting). It is aimed at providing a convenient and powerful compiler, both able to compile legacy applications and to be the means of developing new ones. It is distributed under the GNU GPL, while packages and runtime library come under a modified GNU LGPL. Apart from compatibility modes for Turbo Pascal, Delphi and Mac Pascal, it also has its own procedural and object-oriented syntax modes with support for extended features such as operator overloading. It supports many platforms and operating systems.
- Turbo51 is a free Pascal compiler for the 8051 family of microcontrollers, with Turbo Pascal 7 syntax.
- Oxygene (formerly known as *Chrome*) is an Object Pascal compiler for the .NET and Mono platforms. It was created and is sold by RemObjects Software [15], and recently by Embarcadero as the backend compiler of Prism.
- Kylix was a descendant of Delphi, with support for the Linux operating system and an improved object library. It is no longer supported. Compiler and IDE are available now for non-commercial use.
- GNU Pascal Compiler (GPC) is the Pascal compiler of the GNU Compiler Collection (GCC). The compiler itself is written in C, the runtime library mostly in Pascal. Distributed under the GNU General Public License, it runs on many platforms and operating systems. It supports the ANSI/ISO standard languages and has partial Turbo Pascal dialect support. One of the more painful omissions is the absence of a 100% Turbo Pascal-compatible (short)string type. Support for Borland Delphi and other language variations is quite limited. There is some support for Mac-pascal however.
- DWScript [16] aka DelphiWebScript, is an interpreter created by Matthias Ackermann and Hannes Hernler in 2000. Current version runs a dialect of Object Pascal largely compatible with Delphi, but also supports language constructs elements introduced in Prism. DWScript code can be embedded into Delphi applications similar to PascalScript, compiled into standalone application using SimpleMobileStudio or compiled into JavaScript code and placed on a web page.[17]
- Dr. Pascal [18] is an interpreter that runs Standard Pascal. Notable are the "visible execution" mode that shows a running program and its variables, and the extensive runtime error checking. Runs programs but does not emit a separate executable binary. Runs on DOS, Windows in DOS window, and old Macintosh.
- Dr. Pascal's Extended Pascal Compiler [19] tested on DOS, Windows 3.1, 95, 98, NT.
- Virtual Pascal was created by Vitaly Miryanov in 1995 as a native OS/2 compiler compatible with Borland Pascal syntax. Then, it had been commercially developed by fPrint, adding Win32 support, and in 2000 it became freeware. Today it can compile for Win32, OS/2 and Linux, and is mostly compatible with Borland Pascal and Delphi. Development was canceled on April 4, 2005.
- P4 compiler [20], the basis for many subsequent Pascal-implemented-in-Pascal compilers. It implements a subset of full Pascal.
- P5 compiler [21], is an ISO 7185 (full Pascal) adaption of P4.
- Turbo Pascal was the dominant Pascal compiler for PCs during the 80s and early 90s, popular both because of its powerful extensions and extremely short compilation times. Turbo Pascal was compactly written and could compile, run, and debug all from memory without accessing disk. Slow floppy disk drives were common for programmers at the time, further magnifying Turbo Pascal's speed advantage. Currently, older versions of Turbo Pascal (up to 5.5) are available for free download from Borland's site.

- IP Pascal [22] Implements the language "Pascaline" (named after Pascal's calculator), which is a highly extended Pascal compatible with original Pascal according to ISO 7185. It features modules with namespace control, including parallel tasking modules with semaphores, objects, dynamic arrays of any dimensions that are allocated at runtime, overloads, overrides, and many other extensions. IP Pascal has a built-in portability library that is custom tailored to the Pascal language. For example, a standard text output application from 1970's original Pascal can be recompiled to work in a window and even have graphical constructs added.
- Pascal-XT [23] was created by Siemens for their mainframe operating systems BS2000 and SINIX.
- PocketStudio is a Pascal subset compiler and RAD tool for Palm OS and MC68xxx processors with some own extensions to assist interfacing with the Palm OS API. It resembles Delphi and Lazarus with a visual form designer, an object inspector and a source code editor.
- MIDletPascal – A Pascal compiler and IDE that generates small and fast Java bytecode specifically designed to create software for mobiles
- Vector Pascal [24] Vector Pascal is a language for SIMD instruction sets such as the MMX and the AMD 3d Now, supporting all Intel and AMD processors, and Sony's PlayStation 2 Emotion Engine.
- Morfik Pascal [25] allows the development of Web applications entirely written in Object Pascal (both server and browser side).
- WDSibyl [26] – Visual Development Environment and Pascal compiler for Win32 and OS/2
- PP Compiler, a compiler for Palm OS that runs directly on the handheld computer [27]
- CDC 6000 Pascal compiler [28] The source code for the first (CDC 6000) Pascal compiler.
- Pascal-S [29][30]
- AmigaPascal [31] – AmigaPascal, a free Pascal-Compiler for Amiga-Computer.

A very extensive list can be found on Pascaland [32]. The site is in French, but it is basically a list with URLs to compilers; there is little barrier for non-Francophones. The site, Pascal Central [33], a Mac centric Pascal info and advocacy site with a rich collection of article archives, plus links to many compilers and tutorials, may also be of interest.

IDEs

- Dev-Pascal is a Pascal IDE that was designed in Borland Delphi and which supports Free Pascal and GNU Pascal as backends.
- Lazarus is a Delphi-like visual cross-platform IDE for rapid application development (RAD). Based on Free Pascal, Lazarus is available for numerous platforms including Linux, FreeBSD, Mac OS X and Microsoft Windows.
- Code Typhon [34] is a Pascal IDE built as extended version of Lazarus with a lot of packages shipped and scripts to build FPC cross compilers provided. Code Typhon was designed in free pascal and supports Object Pascal, Turbo Pascal and Delphi too. Programmers could build any program for any device.

Libraries

WOL Library [35] for creating GUI applications with the Free Pascal Compiler.

Standards

In 1983, the language was standardized, in the international standard IEC/ISO 7185, and several local country specific standards, including the American ANSI/IEEE770X3.97-1983, and ISO 7185:1983. These two standards differed only in that the ISO standard included a "level 1" extension for conformant arrays, where ANSI did not allow for this extension to the original (Wirth version) language. In 1989, ISO 7185 was revised (ISO 7185:1990) to correct various errors and ambiguities found in the original document.

In 1990, an extended Pascal standard was created as ISO/IEC 10206. In 1993 the ANSI standard was replaced by the ANSI organization with a "pointer" to the ISO 7185:1990 standard, effectively ending its status as a different standard.

The ISO 7185 was stated to be a clarification of Wirth's 1974 language as detailed by the User Manual and Report [Jensen and Wirth], but was also notable for adding "Conformant Array Parameters" as a level 1 to the standard, level 0 being Pascal without conformant arrays. This addition was made at the request of C. A. R. Hoare, and with the approval of Niklaus Wirth. The precipitating cause was that Hoare wanted to create a Pascal version of the (NAG) Numerical Algorithms Library, which had originally been written in FORTRAN, and found that it was not possible to do so without an extension that would allow array parameters of varying size. Similar considerations motivated the inclusion in ISO 7185 of the facility to specify the parameter types of procedural and functional parameters.

Note that Niklaus Wirth himself referred to the 1974 language as "the Standard", for example, to differentiate it from the machine specific features of the CDC 6000 compiler. This language was documented in "The Pascal Report" [36], the second part of the "Pascal users manual and report".

On the large machines (mainframes and minicomputers) Pascal originated on, the standards were generally followed. On the IBM-PC, they were not. On IBM-PCs, the Borland standards Turbo Pascal and Delphi have the greatest number of users. Thus, it is typically important to understand whether a particular implementation corresponds to the original Pascal language, or a Borland dialect of it.

The IBM-PC versions of the language began to differ with the advent of UCSD Pascal, an interpreted implementation that featured several extensions to the language, along with several omissions and changes. Many UCSD language features survive today, including in Borland's dialect.

Variations

Niklaus Wirth's Zurich version of Pascal was issued outside of ETH in two basic forms, the CDC 6000 compiler source, and a porting kit called Pascal-P system. The Pascal-P compiler left out several features of the full language. For example, procedures and functions used as parameters, undiscriminated variant records, packing, dispose, interprocedural gotos and other features of the full compiler were omitted.

UCSD Pascal, under Professor Kenneth Bowles, was based on the Pascal-P2 kit, and consequently shared several of the Pascal-P language restrictions. UCSD Pascal was later adopted as Apple Pascal, and continued through several versions there. Although UCSD Pascal actually expanded the subset Pascal in the Pascal-P kit by adding back standard Pascal constructs, it was still not a complete standard installation of Pascal.

Borland's Turbo Pascal, written by Anders Hejlsberg, was written in assembly language independent of UCSD or the Zurich compilers. However, it adopted much of the same subset and extensions as the UCSD compiler. This is probably because the UCSD system was the most common Pascal system suitable for developing applications on the resource-limited microprocessor systems available at that time.

In the early 90s, Alan Burns and Geoff Davies developed Pascal-FC, an extension to Pascal-0 (from the Niklaus' book 'Algorithms+Data Structures=Programs'). Several constructs were added to use Pascal-FC as a teaching tool for Concurrent Programming (such as semaphores, monitors, channels, remote-invocation and resources). To be able to demonstrate concurrency, the compiler output (a kind of P-code) could then be executed on a virtual machine. This virtual machine not only simulated a normal - fair - environment, but could also simulate extreme conditions (unfair mode).

List of related standards

- ISO 8651-2:1988 *Information processing systems – Computer graphics – Graphical Kernel System (GKS) language bindings – Part 2: Pascal*

Reception

Pascal generated a wide variety of responses in the computing community, both critical and complimentary.

Criticism

While very popular in the 1980s and early 1990s, implementations of Pascal that closely followed Wirth's initial definition of the language were widely criticized for being unsuitable for use outside of teaching. Brian Kernighan, who popularized the C language, outlined his most notable criticisms of Pascal as early as 1981, in his paper *Why Pascal Is Not My Favorite Programming Language*.[37] The most serious problem described in his article was that array sizes and string lengths were part of the type, so it was not possible to write a function that would accept variable length arrays or even strings as parameters. This made it unfeasible to write, for example, a sorting library. The author also criticized the unpredictable order of evaluation of boolean expressions, poor library support, and lack of static variables, and raised a number of smaller issues. Also, he stated that the language did not provide any simple constructs to "escape" (knowingly and forcibly ignore) restrictions and limitations. (However, there is a feature of "record variants" that does allow such an "escape," though it is decidedly cumbersome.) More general complaints from other sources[38][39] noted that the scope of declarations was not clearly defined in the original language definition, which sometimes had serious consequences when using forward declarations to define pointer types, or when record declarations led to mutual recursion, or when an identifier may or may not have been used in an enumeration list. Another difficulty was that, like ALGOL 60, the language did not allow procedures or functions passed as parameters to predefine the expected type of their parameters.

On the other hand, many major development efforts in the 1980s, such as for the Apple Lisa and Macintosh, heavily depended on Pascal (to the point where the C interface for the Macintosh Toolbox had to use Pascal data types).

Reactions

Pascal continued to evolve, and most of Kernighan's points do not apply to versions of the language which were enhanced to be suitable for commercial product development, such as Borland's Turbo Pascal. As Kernighan predicted in his article, most of the extensions to fix these issues were incompatible from compiler to compiler. Since the early 1990s, however, the varieties seem to have condensed into two categories, ISO and Borland-like, a better eventual outcome than Kernighan foresaw.Wikipedia:No original research

Although Kernighan decried Pascal's lack of type escapes ("there is no escape" from "Why Pascal is not my Favorite Programming language"), pointers and type escapes lead to the sorts of problems that were addressed by the development of languages such as Java and C#.

Based on his experience with Pascal (and earlier with ALGOL) Niklaus Wirth developed several more programming languages: Modula, Modula-2, Oberon and Oberon-2. These languages address some criticisms of Pascal and are intended for different user populations. However none has matched the commercial success or widespread impact on computer science that Pascal had.Wikipedia:Citation needed

The Corvus Systems Constellation (an innovative 1980's computer) centered around interpretative Pascal software which the user could edit at runtime. The manual had printed much of the Pascal code that also came on disk.

Further reading

- Niklaus Wirth: *The Programming Language Pascal.* 35–63, Acta Informatica, Volume 1, 1971.
- C A R Hoare: *Notes on data structuring.* In O-J Dahl, E W Dijkstra and C A R Hoare, editors, Structured Programming, pages 83–174. Academic Press, 1972.
- C. A. R. Hoare, Niklaus Wirth: *An Axiomatic Definition of the Programming Language Pascal.* 335–355, Acta Informatica, Volume 2, 1973.
- Kathleen Jensen and Niklaus Wirth: *PASCAL – User Manual and Report* [40]. Springer-Verlag, 1974, 1985, 1991, ISBN 0-387-97649-3 and ISBN 3-540-97649-3.
- Niklaus Wirth: *Algorithms + Data Structures = Programs*. Prentice-Hall, 1975, ISBN 0-13-022418-9.
- Niklaus Wirth: *An assessment of the programming language PASCAL.* 23–30 ACM SIGPLAN Notices Volume 10, Issue 6, June 1975.
- N. Wirth, and A. I. Wasserman, ed: *Programming Language Design.* IEEE Computer Society Press, 1980
- D. W. Barron (Ed.): *Pascal – The Language and its Implementation.* John Wiley 1981, ISBN 0-471-27835-1
- Peter Grogono: *Programming in Pascal*, Revised Edition, Addison-Wesley, 1980
- Richard S. Forsyth: *Pascal in Work and Play*, Chapman and Hall, 1982
- N. Wirth, M. Broy, ed, and E. Denert, ed: *Pascal and its Successors* [41] in *Software Pioneers: Contributions to Software Engineering.* Springer-Verlag, 2002, ISBN 3-540-43081-4
- N. Wirth: *Recollections about the Development of Pascal.* [42] ACM SIGPLAN Notices, Volume 28, No 3, March 1993.

References

[1] «We looked very carefully at Delphi Object Pascal and built a working prototype of bound method references in order to understand their interaction with the Java programming language and its APIs.»
«Our conclusion was that bound method references are unnecessary and detrimental to the language. This decision was made in consultation with Borland International, who had previous experience with bound method references in Delphi Object Pascal.» White Paper.About Microsoft's "Delegates" (http://java.sun.com/docs/white/delegates.html), java.sun.com

[2] A Conversation with James Gosling (http://queue.acm.org/detail.cfm?id=1017013)

[3] Essential Pascal (http://www.marcocantu.com/epascal/English/ch01hist.htm) by Marco Cantù

[4] TIOBE Programming Community Index for January 2011 (http://www.tiobe.com/index.php/content/paperinfo/tpci/)

[5] Hertzfeld, Andy. " Hungarian (http://www.folklore.org/StoryView.py?project=Macintosh&story=Hungarian.txt&topic=Software Design&sortOrder=Sort by Date&detail=medium)." Folklore.org: Macintosh Stories. Retrieved 2012-03-06.

[6] "An Interview with JOHN BRACKETT AND DOUG ROSS" (http://www.cbi.umn.edu/oh/pdf/oh392jb.pdf), p15, Charles Babbage Institute, 2004

[7] Jon Udell, Crash of the Object-Oriented Pascals, BYTE, July, 1989.

[8] M.I.Trofimov, The End of Pascal?, BYTE, March, 1990, p.36.

[9] PASCAL-XSC: PASCAL for Extended Scientific Computing (http://www.rz.uni-karlsruhe.de/~iam/html/language/pxsc.html)

[10] XSC Languages (C-XSC, PASCAL-XSC) (http://www.xsc.de/)

[11] Pascal-XSC Download (http://www2.math.uni-wuppertal.de/wrswt/xsc/pxsc_download.html)

[12] Pascal ISO 7185:1990 (http://www.moorecad.com/standardpascal/iso7185.pdf) 6.10

[13] J. Welsh, W. J. Sneeringer, and C. A. R. Hoare, "Ambiguities and Insecurities in Pascal", *Software Practice and Experience 7*, pp. 685–696 (1977)

[14] *Pascal,* Nell Dale and Chip Weems, "Dangling Else", p. 160–161 (http://books.google.com/books?id=5x2k4vWwn1wC&pg=PA160)

[15] http://www.remobjects.com/

[16] http://code.google.com/p/dwscript/

[17] "Flock" DWScript / JavaScript CodeGen demo (https://code.google.com/p/dwscript/downloads/detail?name=Flock-JSCodeGenDemo.7z&can=2&q=)

[18] http://www.visible-software.com/prod-dp.html

[19] http://www.visible-software.com/prod-ep.html

[20] http://homepages.cwi.nl/~steven/pascal/

[21] http://www.standardpascal.com/p5.html

[22] http://www.moorecad.com/ippas/

[23] http://ts.fujitsu.com/products/bs2000/software/compiler/pascalxt.html

[24] http://www.dcs.gla.ac.uk/~wpc/reports/compilers/compilerindex/Doc2.html

[25] http://www.morfik.com
[26] http://www.wdsibyl.org
[27] http://www.ppcompiler.org
[28] http://www.standardpascal.org/CDC6000pascal.html
[29] http://www.moorecad.com/standardpascal/pascals.html
[30] "Pascal-S: A Subset and Its Implementation", N. Wirth in Pascal – The Language and Its Implementation, by D.W. Barron, Wiley 1979.
[31] http://aminet.net/package/dev/lang/AmigaPascal
[32] http://pascaland.org
[33] http://pascal-central.com
[34] http://www.pilotlogic.com
[35] http://wol.sourceforge.net
[36] http://www.standardpascal.com/The_Programming_Language_Pascal_1973.pdf
[37] Brian W. Kernighan (1981). Why Pascal is Not My Favorite Programming Language (http://www.lysator.liu.se/c/bwk-on-pascal.html)
[38] O. Lecarme, P. Desjardins, "More Comments on the Programming Language Pascal," *Acta Informatica 4*, pp. 231–243 (1975)
[39] J. Welsh, W. J. Sneeringer, C. A. R. Hoare, "Ambiguities and Insecurities in Pascal," *Software Practice and Experience 7*, pp. 685–696 (1977)
[40] http://www.cs.inf.ethz.ch/~wirth/books/Pascal/
[41] http://www.swissdelphicenter.ch/en/niklauswirth.php
[42] http://portal.acm.org/citation.cfm?id=155378

External links

- The Pascal Programming Language (http://pascal-central.com/ppl/index.html)
- Standard Pascal (http://www.standardpascal.org) – Resources and history of original, standard Pascal.
- Free Pascal SciTech portal (http://wiki.lazarus.freepascal.org/Portal:SciTech) with applications of Lazarus and Free Pascal for Science, medicine and technology.

Article Sources and Contributors

Pascal (programming language) *Source*: http://en.wikipedia.org/w/index.php?oldid=609335423 *Contributors*: 16@r, 22JaKe22, A D Monroe III, A Man In Black, AbstractClass, Aekton, Ahoerstemeier, AlanUS, Alansohn, AlbertCahalan, Alexei Kopylov, Amniarix, AndersL, Andre Engels, Andreas Rejbrand, Andrejj, Andres, Andrewa, Angela, Anna Lincoln, Arifsaha, Arny, Arthena, Asendoh, Atanamir, Audriusa, Austriacus, BD2412, Baylor Rae, Bdshahab, Beetstra, Belovedfreak, Ben-Zin, Bento00, Betterusername, Bevo, Bige1977, Binarybits, Blahma, Blancomonk, Bluemask, BogdyBBA, Bonadea, Bongwarrior, Bowmanjj, BraneJ, BrianWren, Brighterorange, Brion VIBBER, Brothejr, Bubba73, BurntSky, Byrial, CRGreathouse, CWii, Canterbury Tail, Capricorn42, Captain Disdain, Cedars, Chairboy, Chaos5023, Chester Markel, Chinju, Chkno, Chosen1, Choster, Chris Burrows, ChrisGriswold, Cjthellama, Classicalecon, Conversion script, Corpx, Corti, Craig Stuntz, Curps, Cybercobra, D. Hirtle, DCEdwards1966, DEmerson3, DNewhall, DVdm, Dachshund, Damieng, Dan100, Danakil, Danallen46, Darth Newdar, Daustins, Davehi1, David Schaich, DavidCary, Davidheffernan, Dcoetzee, Deele, Dennis Bratland, Derek Ross, Derek farn, Dgpop, Dianelos, Diderot's dreams, Digital Brains, Diogenes2000, Discospinster, Disnevil, Dmacgr 22, Dmarshal, Docu, Dominus, Dori, Drnoitall.hello, Duskstalker, Dybdahl, Dysprosia, EWAdams, Earlypsychosis, Edcolins, Edward, Elitedev, Eloc Jcg, Elwikipedista, EmilJ, Enchanter, Epbr123, Ernst.schnell, Etxrge, EugeneZelenko, Evercat, Exercisephys, Fashionable.politics, Firedragonuk, Firefly's luciferase, Fleminra, FourBlades, Fubar Obfusco, Furrykef, GRAHAMUK, Gareth Griffith-Jones, Gary D Robson, Garydale, Georg Peter, Geregen2, Ghettoblaster, Giftlite, Gilliam, Glasreiniger, Glenn, Gnfgb2, Graham87, GregorB, Grstain, HCA, HMSSolent, Hackerz.bz, Haikupoet, Hannes Hirzel, Hans Adler, Hans Bauer, HappyCamper, Harveyz1, Hasek is the best, Hayabusa future, Headbomb, HenkeB, Henning Makholm, Heron, Hirzel, Homecoming 000, Hqb, HughesJohn, Hvn0413, Hydrargyrum, I am One of Many, Ian Pitchford, Icairns, Igorfuna, Iliealldaylong, Intrr, JC Chu, JLaTondre, JRM, Jaan513, JakobVoss, Jan Hidders, JanSuchy, Jarble, Jbolden1517, Jeffadams78, Jengelh, Jerryobject, Jim McKeeth, John, JohnBlackburne, Johncc330, Joly, Jonathan de Boyne Pollard, Jorge Stolfi, Joshbow, Jpbowen, Jpk, Jrogers@olorin.us, Jsimlo, Jurij Veresciaka, Jusdafax, Jwdietrich2, KP51499, Karl2620, Katieh5584, Kdau, Kesla, Killiondude, Kimiko, Kirananils, Kleg, Klilidiplomus, Knyf, Konsnos, Koyaanis Qatsi, Krischik, Krymson, Ks0stm, Ktx36, Kurt Shaped Box, Kuszi, Kuyabribri, Laurieboshell, Ldsandon, Leibniz, Leledumbo, Lemontea, Lfwlfw, Liberatus, LightningPower, Loadmaster, Luk, Lupinoid, M-le-mot-dit, MER-C, Macrakis, Mahir256, Marasmusine, Marcov, Marek69, Mark Foskey, Martarius, Matt.forestpath, Mendax666, Mernen, Mfwitten, MiCovran, Michael Hardy, Michael Zimmermann, Michaeltarpley, Microtony, Mike92591, Mindmatrix, Minesweeper, Mirror Vax, Mitch Ames, Mogism, Moonburntm, Mormegil, MrOllie, Mxn, Nanshu, Nasa-verve, Nbarth, NevilleDNZ, Newmanbe, Nunquam Dormio, Oldmanbiker, Oleg Alexandrov, OrenBochman, PGSONIC, Pascal.Tesson, PascalTheCat, Pascalman, Patrick, Paul Foxworthy, Pauli133, Pgn674, Pictureuploader, PierreAbbat, Pjvpjv, Pldtsmarter, Pmiller42au, Pne, Polluks, Poor Yorick, Pqrstuv, Pratyya Ghosh, Prodego, Professordad42, Prosfilaes, Pterre, Qfwfq, Qoou.Anonimu, Quest for Truth, Qwertyus, R'n'B, R.123, RTC, Raptor66, Rbonvall, RedWolf, Remember the dot, Retired username, Rfc1394, Rhobite, Rich Farmbrough, Rickjpelleg, Riddley, Rlee0001, Robbe, Robertgreer, Rocastelo, RodC, RogueMomen, Roundhouse0, Roxfan, Rsshelas, RucasHost, Ruud Koot, SMC, Samiam95124, Sarrazip, Sathiyamoorthysp, Sdfisher, Sehrgut, Sehugg, Sekelsenmat, SensiStarToaster, Sfitztrw, Shanes, Shervinafshar, Siva.eas, Smaug123, SniperDubey, Snori, Sonett72, Spindocter123, Splintax, SteinbDJ, Stevietheman, Stewartadcock, StuartBrady, SummerWithMorons, Svick, T-bonham, T0mt0m666, Tabletop, TakuyaMurata, Talandor, Tcncv, Tedickey, Template namespace initialisation script, Terryn3, TheEgolf, Thebdj, Thejoshwolfe, Theopolisme, ThreeDee912, Thumperward, Tide rolls, Tikiwont, Tim32, Tjeerdnet, Tobias Bergemann, Tom-, Tony Sidaway, Tyler, UnDeRTaKeR, Unconventional, Updatehelper, Uriyan, UtherSRG, Vikreykja, Vlad, Wagonkeys, Wavelength, Wbeek, Weel, Who, Wickorama, Widr, Wikifriend pt001, Will Beback, William Allen Simpson, Windymager, Wisgary, Wjl2, Woohookitty, Wrp103, Ww, Wws, Xlegiofalco, Yath, Zeimusu, Zero0w, Zoicon5, Zootm, Zudduz, Александър, Ὁ οἶστρος, 747 anonymous edits

License