Compared Anatomy of the Programming Languages Pascal and C

Vincent Hayward *

Abstract

The programming languages PASCAL and C belong to the class of algorithmic compiled languages, and feature comparable facilities. The syntax of C is more regular and enjoys less restrictions than that of PASCAL. Consequently, the syntax of C is significantly simpler than that of the PASCAL language. The design differences are particularly apparent as far as data types, operators, and system interface are concerned. PASCAL implements abstract data types (booleans, sets), whereas C defines data types that reflect the architecture of currently available computers. For PASCAL, the system interface is implemented via functions built in the language and is therefore not extensible by the user. Most of the practical implementations of the language require non standard extensions. In C, system calls and libraries of subroutines and macro instructions are provided to handle the system interface. The portability of applications then depends on the libraries and not on the compiler. A greater flexibility is thus achieved. The use of pointers and the rich set of operators of the C language make it more general than PASCAL, while retaining the principles of structured programming. Separate compilation is a standard feature of C, not of PASCAL. PASCAL compilers are often complex, while C compilers are simple and compact. C compilers include a macro processor, and the program verifier is a separate tool. The application programmers have more responsibility programming in C than in PASCAL, conversely, applications depend less on compilers in C than in PASCAL.

* Computer Vision and Robotics Laboratory, Department of Electrical Engineering, McGill University, Montréal, Québec, Canada, on leave from Laboratoire d'Informatique et de Mécanique pour les Sciences de l'Ingénieur, LIMSI-CNRS, BP 30, 91406 Orsay Cedex, France

1. Introduction

A detailed comparison of the programming languages PASCAL and C can be helpful when it comes to deciding of the implementation language of a large project because the two languages have similar capabilities. PASCAL [Wirth] and C [Kernighan] are algorithmic, procedural, strongly typed, and compiled languages both inspired from the ALGOL language [Backus]. Structured programming, data structures, and recursivity are easily used in the two languages.

An exhaustive comparison of the languages should take into account the operating systems under which the programs will be developed and run, the available compilers, and the target machines. A complete analysis taking all these parameters into account is a difficult task, given the large number of possible combinations. Dogmatic conclusions are therefore risky to establish. As much as possible, I shall try to mention the points where such parameters are to be considered. This report does not review the case of concurrent programming.

2. Surface Aspect

In the two languages, programs can be written in free format. Programs can be properly indented, spaced, and commented. In any case, the result depends on the good taste of the programmer.

C allows the user to write arbitrarily dense expressions (`'d%=*a++***+b-*&c[0]+1'` , for example). The use of such expressions is rarely justified. The practice shows that only a few commonly encountered idioms are used and rapidly mastered by the programmers (for example, the expression `'*p++'` that simultaneously accesses a value pointed by the pointer `p` and increments this pointer, is a very common construction, most of the time compiled into a single machine instruction). PASCAL is more restrictive and such constructions are not allowed. In PASCAL, expressions have a more familiar aspect to the beginner.

In both languages, identifiers can be arbitrarily long. Restrictions are rarely due to compilers, but rather to other system utilities such as link editors and debuggers. Generaly speaking, expressivity is comparable, but C programs tend to be more concise than their PASCAL equivalents. Economy of expression can be advantage, when it is properly used.

The syntax of the C language is described with 35 BNF rules and 26 key-words. The syntax of PASCAL requires more than a hundred rules and 35 key-words. Figures cannot be accurate in the case of PASCAL because its implementation often requires extensions. The difference in complexity can be explained by the fact that the syntax of C only handles the capabilities of the CPU of a general purpose computer and does not take into account "high-level" functions that depend on the operating system. In other terms, input-output, dynamic memory allocation, mathematical functions, etc... are handled by standard libraries of subroutines, of macro instructions, and system calls. On the other hand, PASCAL is a closed system that must rely on itself.

3. Constructions

3.1 Control Structures

3.1.1 Blocks

Both languages implement the block construct, in other terms, sequences of instructions can be gathered inside execution units. In C, local variables can be allocated within any block, and their scope is reduced to the surrounding block. Blocks are recursive in C, not in PASCAL.

3.1.2 Choice

The two languages have similar constructs: `'if (expression) block'`, with an optional `'else'` clause.

3.1.3 Iteration

The two languages have a similar `while` loop. The PASCAL language has a `'REPEAT block UNTIL (condition)'` loop style, the C language has a `'do block while (condition)'`. Strictly speaking, in C there is no loop with a control variable in the style of the PASCAL or FORTRAN `FOR` loop. Instead, C provides the following construct: `'for (s1; s2; s3) block'`, where `'s1'` is an initialization clause, `'s2'` is the loop control expression, and `'s3'` is a loop re-initialization clause. C has two supplementary constructs: the `break` statement breaks the execution of a loop at an arbitrary point, the `continue` instruction causes the program to abandon the execution of a loop body and forces the evaluation of the loop control expression.

3.1.4 Selection

The `switch` construct of C is equivalent to the `CASE` of PASCAL. The tags must be integer constants, but in C, by default, the flow of control "falls through" the different cases if it is not stopped by a `break` statement. This is more general because one can implement clauses like `'case ...or case ...or case ... : action'`, but it is questionable as far as program clarity.

3.1.5 Goto

In C, as well as in PASCAL, the `goto` can be used unwisely, arbitrary jumps are possible within a function. Labels are literal in C, they are numeric in PASCAL.

3.2 Data Types and Data Structures

3.2.1 Simple Types

The differences between the two designs appear more clearly in this section. The PASCAL language provides a set of abstract data types whose implementation depends on the decisions of author of the compiler. Implementation details are, as far as possible,

hidden from the user. In C, the user is made more conscious of the realities of a machine and he or she has more freedom for choosing and using adequate data types. The C language provides several real and integer types that correspond to the types that one might expect to find on a modern computer. However, experience has shown the good portability of C programs, provided that the program authors have shown a minimum of competence for encapsulating, in their programs, the parts that can be sources of problems, while retaining the efficiency inherent in the language. The macro processor and the `typedef` construct, that can rename types within a whole project, are effective tools. Badly designed PASCAL programs that depend on the compiler will leave the application implementor without systematic resort.

Boolean Type

PASCAL implements the boolean type as an abstract type, whereas C, as stated before, ignores it because the boolean type is not a machine type. If there little to say about the PASCAL boolean type, the C language provide a few features. The result of any integer expression can be used in a logical expression (to be null or not), furthermore, the result of any logical expression (0 or 1) can be used in an integer expression, somewhat in the way of the APL language (and LISP as well). In C, pointers and characters can be converted or be considered as integer types, therefore, pointers and characters can be operands of logical operators.

Character Type

In PASCAL, the character is also an abstract type. The internal functions `ORD` and `CHR` are therefore made necessary for low level manipulations and ordering. In C, the type `char` is rather the memory allocation unit in which a character can be stored. In the great majority of cases, it an eight bit byte, the smallest addressable unit, and the "seed" for the measure of the size of objects. In C, the type `char` behaves like an integer. Programs are made portable for different character codes through the use of macro instructions (isalpha, isdigit, isupper, ...) included in an easily extensible standard library. Characters, considered as integers, can index arrays and serve the purposes of very short integers.

Integer Type

PASCAL has only one integer type, C implements four of them: `char`, `short`, `int`, and `long` (this corresponds on most machines to 8, 16, 32, and 32 bits or 8, 16, 16, and 32). The `int` type can be qualified `unsigned` to specify if the sign must be taken into account for integer operations and shifts. For finely tuned applications, the `typedef` construct permits the programmer to adjust his programs to the best of a machine, otherwise, the `int` type will do for ordinary applications. This feature protects the C user from a compiler deficient for certain operations. The PASCAL user is obliged to comply with the decisions of the author of the compiler.

Real Type

In the same vein, PASCAL has one `real` type, whereas C distinguishes the `float`

(single precision) from the `double` (double precision). Again, the `typedef` construct or type renamer can be used to switch from simple to double precision within a whole project.

Enumeration Type

Since 1978, the C language implements the type `enum` quite comparable to the `SCALAR` type of PASCAL. Nevertheless, C has no equivalent for the `RANGE` and `SUBRANGE` concepts.

Set Type

The C language has no type `SET` like PASCAL, because set operations can be simply and efficiently implemented with the bitwise operators of the language (complement, and, inclusive and exclusive or). The implementation of sets can have important consequences for the efficiency and the portability of PASCAL programs, and is not, in C, buried within the compiler. Finally, in C, the user is free to create set operators, such that the cardinal of a set, that PASCAL would not provide.

3.2.2 Aggregates

C and PASCAL provide comparable facilities for the aggregate data types. Aggregate objects gather under the same identifier a collection of objects non necessarily identical. The PASCAL language does as much as possible to hide the implementation of those types, and leaves the user little choice to determine an adequate solution to his problem. The flexibility of the C language will become even more apparent in the section "pointers", because pointers are closely related to aggregates.

Structures

The type `RECORD` of PASCAL corresponds to the `struct` of C. However, there are some differences. C has no `VARIANT RECORD`, or in other terms a structure with a variable definition. Nor C has the directive `WITH` to shorten the specification of members of nested structures, but C pointers can efficiently fulfill this role. Structure declarations, in C, are effectively mapping directives for storing objects in memory, but PASCAL hides this fact. C has two other constructs: the `union`, and the `field`. The `union` construct is used to store objects of different types into the same memory space and guarantees a correct alignment. The `field` construct is used to manipulate machine words as structured collections of bits, in a portable manner.

Arrays

Arrays are handled in a similar way in both languages. Arrays are ordered and indexable collections of objects of the same type, simple type or aggregate type. In particular, arrays of dimensions higher than one are considered in both languages as arrays of arrays. In C, one cannot write conventionally `t[i, j]` instead of `t[i][j]` as in PASCAL and indexes always start at 0, whatever their type can be. The `PACKED` arrays of PASCAL have no purpose in C. The `PACKED` arrays of PASCAL are made necessary because of the

possible use of arrays of booleans that are sometimes preferably implemented as contiguous bit arrays. The user then has the choice between two possibilities provided by the compiler (when the relevant information is available). The C user must design a proper implementation and the choice is then very large. Another justification for packed arrays is found in the implementation of character strings, however general purpose machines unable to address one single byte are becoming rare.

3.2.3 Pointers

In both languages, the main purpose of pointers is the creation of dynamic, self-referential, data structures such as lists and trees. C makes several uses of pointers that are unknown in PASCAL.

C pointers are used for the declaration of function arguments. All the function arguments are passed by value. Passing an argument by address can be implemented by passing a pointer to an object. Therefore, the declaration of function arguments is general and in harmony with the syntax and the semantics of the rest of the language.

C can handle address arithmetic. Integers can be added to, or substrated from pointers, and pointers can be compared. In effect, any program does address arithmetic, but in C, it is explicit, portable, efficient, and easy to use. Pointers and arrays are identicals. An array is simply a constant pointer, a pointer to the first element of the array. A number of techniques such as variable array sizes, index-less array scanning, etc... are then possible. Conversely, in PASCAL, arrays have a fixed size, and this restriction often leads to inefficient programs, for example, processing character strings. However, arrays cannot be passed by value in C.

In C, pointers can point to functions, therefore, pointers to functions can be passed as arguments or stored into data structures, and what is pointed to evaluated. This is not possible in PASCAL.

3.2.4 Initialization

As opposed to PASCAL, C provides a mechanism for the initialization of variables. This mechanism is not quite general, but is very useful in practice.

Failing to find an appropriate section, we can mention here the C compiler directive `register`. This directive allows the user to advise the compiler to store pointer variables and simple type variables in the machine's fast registers. In practice, there are restrictions that reflect the underlying hardware. However, this feature allows the user to write functions that are nearly as efficient as if they were written in assembly code. In many cases, resulting programs are more compact and efficient.

3.3 Operators

Relational, arithmetic, and indexing operators are quite comparable in the two languages. As opposed to PASCAL, C implements an automatic type conversion mechanism, however, C does not convert data types "with the wild abandon of PL/I". Automatic type conversions concern only the arithmetic and the assignment operators acting on integers and real variables.

The C language offers a variety of original operators: bitwise operator acting on integer types (inclusive or, exclusive or, and, one complement, shifts), reference, pointer dereference, operational assignment (`'x *= y'` stands for `'x = x * y'`), conditional expressions (the `max` of (x, y) can be coded `'max = (a > b) ? a : b'`), pre- or post-incrementation and decrementation. C has the special operator `cast` to force a type conversion. A complete description would be lengthy. In C, the assignment is a true operator (one can code: `'a = b = c = 0;'`). There is a total of 45 operators, that can all be implemented by one or two machine instructions.

The definition of the PASCAL language does not make clear the order of evaluation of logical expressions. In C, the evaluation is guaranteed to stop when the result of an expression is determined. Lastly, in C, simple optimizations such as the pre-calculation of constant subexpressions and the implementation by shifts of multiplications and divisions by powers of two are guaranteed to be applied. In C, high level optimizations such as redundant expression elimination, loops unwrapping, and calculations re-arrangement, will never be applied. This is for two reasons. The compiler must stay simple and compact, and the code generated by the compiler must always be predictable. The user is therefore responsible for detecting and implementing the optimizations that he wishes. For completeness, let us mention the operator `sizeof` . At compile time, the value of `sizeof` is the size of an object given as argument.

The C language has been criticized for the choice of the precedence of its operators.

3.4 Program Structure

3.4.1 Procedures and Functions

Procedures and functions are quite similar in the two languages, although C does not distinguish functions from procedures, a procedure is simply a function that does not return a result. As opposed to PASCAL, C does not allow the user to build a function locally within a function. In C, functions can return at an arbitrary point with the `return(`*expression*`)` construct.

3.4.2 Variables Allocation

The C language provides two distinct variable allocation modes. In static mode, memory space is allocated at program load time and is possibly initialized. In dynamic mode, variables are allocated in the stack at entry time of a function, the main function is considered as an ordinary function. The PASCAL language does not make any distinction and the variables allocated in the main procedure serve the purpose of static variables. This causes some problems for the separate compilation mechanisms. As opposed to PASCAL, C can allocate static variables, whose scope is limited to a function, that can play the role of private remnant variables.

3.4.3 Compilation Units

The standard definition of PASCAL does not provide any separate compilation mechanism. Given the importance of this feature, many academic and industrial laboratories

have added extensions to the language. Unfortunately, the structure of the language is not well adapted to separate compilation, and it led to the creation of a variety of "modular PASCAL's."

Since its inception, the C language has been designed for handling separate compilation. A compilation unit is a collection of global variables and functions that the programmer can decide to be accessible or not from outside the unit. The C preprocessor allows the user to store the declarations that are common to several compilation units in header files, and to build software packages including on one hand, the compiled code, and on the other, the type declarations and associated external references. The so-called modular PASCAL's provide similar mechanisms, sometimes more powerful, but their portability is never guaranteed.

3.5 Inter-language Communications

Inter-language communication is much dependent on the underlying architecture of the host machine. With very few exceptions, because of its generality, the C language does not require any particular construct to implement the interface to another language. Type conversions, in the great majority of cases, can be handled by C types and operators at application level. In PASCAL, inter-language communications must be considered in the design of the compiler and, when it is available, leads to the creation of yet new key-words and syntactical constructs.

4. System Interface

At this level, the design options are different enough to require separate discussions, one for each language.

4.1 Pascal

Dynamic memory allocation is implemented by the language build-in functions `NEW` and `DISPOSE`. Parameter passing to the main function is unknown to the standard, nevertheless, given the importance of this mechanism, it is implemented in various manners according to the installations. The PASCAL standard definition only provides for sequential input-output, and the number of logical units is determined at compile time. This can be a significant problem for many applications. Text files are handled in a special way, and the standard must provide for six supplementary internal functions (`READ`, `WRITE`, `READLN`, `WRITELN`, `EOL`, `PAGE`), besides the five minimum input-output functions (`PUT`, `GET`, `REWRITE`, `REWIND`, `EOF`). The data formatting facilities are not easy to use and depend on the various installations. The standard does not make any reference to random access files, files simultaneously opened for input and output, etc... Given the lack of generality of the input-output system, the various incarnations of the language exhibit various kinds of extensions.

4.2 C

At first, it is difficult to consider the development of the C language independently from that of the UNIX * system. The UNIX system, which is widely in use in use in universities as well as in industry, gives access to a general and complete set of system calls designed in such a way that their number is kept small. C can use directly any of these system calls, and the system interface is reduced to nothing. In order to facilitate the work of the programmer, the compiler is provided with standard libraries of subroutines and macro-instructions. The standard libraries form a general purpose package (string processing, buffered input-output, dynamic memory allocation, data formating, sorting, searching, mathematical functions, pattern matching, ..., not mentioning the large number of various libraries found on most of the systems, most of which are written in C) that, in turn, only use the system calls. If the C language is to be used under another operating system, the quality of the emulation of the UNIX system calls by the host system, and the completeness of the standard library, must be examined with the greatest care. Let us mention that some excellent installations of the C language have been performed under several operating systems. However, none of them can achieve the harmony of the installation of C under UNIX, itself written in C.

5. Programming Environment

Some PASCAL systems provide remarkable programming environments including syntax oriented editors, separate compilation facilities, and debuggers. However, given the great diversity of these systems, it is difficult to draw a definite conclusion. The C compilers always include a macro processor that provide for the use of header files, macro instructions, and conditional compilation. C compilers go along with a utility program called `lint` whose function is to help to the verification of the correctness of programs. Programs that pass the `lint` test without complaints are claimed to be as valid as ALGOL 68 programs. The verifications of `lint` concern type errors, wasteful code, dangling variables, variables used before being assigned, function that might not return a result, some infinite loops, non-portable constructs, expressions that might depend on the order of evaluation, in short, what a simple compiler would allow. Many PASCAL compilers provide these verifications, leading to a greater complexity of the compiler. C compilers are always fast and compact, and the `lint` program is used occasionally.

6. Conclusion

The reader will have noticed the larger portion of text dedicated to the C language rather than to the PASCAL language. This is due to the greater generality of C, but this generality must not be confused with the generality of an assembly language because C programs are truly portable. In fact, more responsibilities are given to the application programmer, and it will be a benefit only for certain projects, research projects, for example.

---

* UNIX is a trademark of AT&T Bell Laboratories.

6. Conclusion

The PASCAL language is attractive for small educational projects and for the explanation of algorithms and programming techniques. It has been designed for this purpose. The study of the construction of the compilers would require more space for PASCAL than for C.

If the development of programs is to be done under UNIX, the choice of the C language is a must. The choice of any other language of equivalent properties would lead to reduced possibilities. If the choice of the operating system is arbitrary, whatever language is chosen, the portability of the programs can be impaired for the usual reasons: programs that depend on the range of fixed point representations, on the order of bytes in machine words, on the order of evaluation of expressions, procedure side effects, etc... From this point of view, the C language, more permissive than PASCAL, might encourage these examples of bad programming habits. For ordinary applications, programs written in the two languages can be considered as equally portable, C programs being generally more efficient.

The languages that could compare to C and PASCAL are PL/I [IBM], BLISS [DEC], and ADA [DoD]. The design options of the PL/I language are quite different from those of C. PL/I goes much further than PASCAL in the endeavor to hide the implementation details since the user is free to create freakish types and operators such as 33 bits integers and their complete set of operators. Due to the complexity of the design, PL/I has only been implemented on a restricted range of machines. The BLISS language shares a number of design ideas with the C language, the use of pointers and expressions, for example. The BLISS compiler includes a very sophisticated code optimizer. The compiler construction, entirely machine independent, is one of the greatest originality of BLISS. Some operating systems designed for the machines of the Digital Equipment Corporation are written in BLISS. However, the language has not met a wide success in universities because of the complexity of the compiler. The ADA language has been designed to be the general purpose language of the future and integrates many of the modern programming concepts such as operator overloading, data encapsulation, packages, and tasking. The ADA design effort began some ten years ago, and the language has evolved through many stages. ADA seems to become a very promising standard, although it has not gained yet a very wide acceptance. The C language gained very early the interest of practitioners because of its simplicity and generality. Only elementary notions of machine architecture are required to make a good use of it. Created in 1972, it has been revised only once in 1978, and minor adjustments have been made along the years. Because of the simplicity of the language, the need for an international standard has not been felt crucial. However, ANSI has recently proposed a standard for the C language. The popularity of the PASCAL language is probably due to progress made in teaching, and to the necessity to replace FORTRAN for this purpose. The fact that labels are numeric in PASCAL could be explained by the intention of discouraging the use of `goto`'s and not by an improbable reminiscence of FORTRAN.

As far as the evolution of machine architecture is concerned, the C language seems to be a good candidate for programming the new generation of general purpose machines, the so-called RISC machines (Reduced Instruction Set Computers), because of the rich set of C operators and types. The architectural regularity of these machines, practically makes

them C machines. The development of PASCAL oriented machines seems to have reached a dead end, and ADA machines are still at the experimental stage.

The final conclusion will be left to the authors of the languages:

> The development of the language PASCAL is based on two principal aims. The first is to make available a language suitable to teach programming as a systematic discipline based on certain fundamental concepts clearly and naturally reflected by the language. The second is to develop implementations of this language which are both reliable and efficient on presently available computers. — N. Wirth.

> C is a general purpose programming language which features economy of expression, modern control flow and data structures, and a rich set a operators. C is not a "very high level" language, nor a "big" one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages. — B. W. Kernighan, D. M. Richie.

7. References

[Backus] J. W. Backus et al., Peter Naur Editor, "Revised Report on the Algorithmic Language ALGOL 60," *Communication of the ACM.*

[Wirth] K. Jensen, N. Wirth, "PASCAL User Manual and Report," Springer-Verlag.

[Kernighan] B. W. Kernighan, D. M. Richie, "The C programming Language," Prentice Hall.

[DEC] "BLISS 10 Programmer's Reference Manual," Digital Equipment Corporation, Maynard, Massachusetts.

[IBM] "OS PL/I Optimizing Compiler: General Information," IBM.

[DoD] "The ADA Programming Language Reference Manual," ANSI/MILSTD 1815A, US Dept. of Defense, US Government Printing Office, 1983.