**The Pascal Programming Language**
Bill Catambay, Pascal Developer

Updated: 9-5-01

# The Pascal Programming Language

by Bill Catambay

## Table of Contents



## I. Introduction

This paper is a review of the Pascal programming language. I will address the origin of the language, discuss the architecture, and talk about the language standards for unextended Pascal and Extended Pascal. I will confront the major criticisms of the language, explaining the origin and inaccuracy of the many myths about Pascal. Finally, I will address the Pascal implementations available today, comparing the

different compilers and the different platforms on which Pascal is currently available.

My experience with Pascal dates back to the PDP-11, the system used at Santa Clara University where I received my Bachelor of Science in Computer Science in 1984. During my college years, I learned and worked with several languages on campus, ranging from FORTRAN to Pascal, COBOL to Assembly. Off campus, I did extensive programming with BASIC (at home on my Radio Shack TRS-80 Model 1) and APL (at Lockheed where I worked part-time). Before tackling the subject of Pascal, I believe it's relevant to discuss my involvement with the language first.

## About the Author

While attending college, I worked part time for Lockheed Missiles & Space where I discovered APL (**A P**rogramming **L**anguage), a highly vectorized symbolic language that ran on an IBM mainframe computer. I originally dabbled in APL to write programs that checked my school work in vector algebra and other math courses, but as my knowledge of the language increased, so did my use of the language. I took over a project previously slated to be written in COBOL (**CO**mmon **B**usiness **O**riented **L**anguage), and wrote the entire system in APL. As this project grew, it wasn't long before I had my first painful experience of code maintenance. Unlike school assignments which could be discarded after turning in, programs written for business came with maintenance responsibilities. I would look at old code, and would wonder what the code was doing, sometimes killing hours of my time. It didn't matter how rigorously I commented my code, as there would always be some code that was not commented or where the comments were not too clear, and I would be clueless. Along with this new pain came a sense of doom associated with change requests. In fact, I often re-wrote a program rather than update an existing one in order to save time and sanity. In the beginning APL was fun, but it soon became obvious that it was not a good choice as a long term programming solution for an on-going project.

Around this same time, my focus in college shifted, as I changed my major from Math to Computer Science at Santa Clara University. One of my most memorable lessons was from the class, Theory of Algorithms. This class stressed that the art of programming was not the learning of a programming language, but the thought process of taking real-life problems, finding a solution, translating that solution to an algorithm, and finally converting the algorithm into working code. The professor of my Data Structures class furthered that idea by stating that learning the basics of a computer language could be accomplished in about two weeks, but the real work is learning to use that language to design smart solutions to complex problems.

After graduation, I entered the Graduate Engineer Program at Lockheed, and rotated through different organizations performing a variety of tasks. I wrote missile guidance FORTRAN (**FOR**mula **TRAN**slation) code on a VAX computer, voice I/O RATFOR (**RAT**ional **FOR**TRAN) code on a Data General computer, and I learned C on a Sun workstation in support of missile guidance analysis.

In 1986, I began to work full-time for Lockheed's missile production shops. Tasked with my first project, a dynamic capacity planning reporting system, I abandoned APL in favor of NOMAD, a high-level database language. NOMAD is a great language

for reporting needs, and much easier to maintain than APL. However, as I discovered, NOMAD lacks much of the structured programming architecture that was built into languages like C and Pascal, making it more difficult to implement complex functionality.

In 1987, I was tasked with full responsibility of designing an entirely new manufacturing system from the ground up. System requirements included responsiveness, stability, flexibility, and ease of maintenance. Additionally, it would operate in manufacturing production shops and would be used by the production line flow assemblers, few of which had prior computer experience. Barcode technology, batch processing of components, as well as an interactive link to a German-made light-guided assembly machine all needed to be controlled by the system. This was a crucial high profile system which would revolutionize the way the shop conducted business, becoming an embedded production component upon which the shop would heavily depend.

After careful analysis of the system requirements, it was obvious that we needed both a stable operating system as well as a versatile and highly maintainable computer language. We chose a dedicated VAX computer running VMS, and the programming language Pascal. The applications needed to implement this system were extremely complex and diverse, ranging from database management to network interfacing with other company systems. Each application would have to be extremely maintainable to support the changing dynamics of a production shop. Of all the languages at our disposal, only Pascal and C afforded the versatility required; however, only Pascal provided the maintainability.

In March of 1988, the system went on-line and was a tremendous success. As shop floor personnel with little computer experience began to work with the system, new ideas continually were inspired. There was an explosion of programming requests, and the system expanded to manage work flow, traceability, hazardous materials usage, shop supplies ordering, tool calibrations, and much more. The system grew to interface with over a dozen other Lockheed systems, and eventually replaced most of the paper processing in the shop, including the replacement of paper timecards with an electronic timecard system. The system supports production shops in Sunnyvale, California, and in Kings Bay, Georgia.

There are many dynamics of the production shops that require a system to be rock solid stable, and capable of easily evolving with ever-changing requirements. This system, now over a decade old with a staff of just two programmers, has successfully done both. Very old applications are maintained and enhanced in a timely manner, including programs written as far back as 1988. New applications are implemented that provide new functionality, often requiring updates to existing library functions while not breaking old code. Through all the changes, it was DEC Pascal (now Compaq Pascal) that played a pivotal role in the success of this system. It provided the required power and flexibility, while simultaneously affording a structured language that helped prevent the system from becoming a maintenance nightmare. It was the right choice then, and is still the right choice today.

In addition to the Pascal programming I do for Lockheed Martin, I also maintain Pascal Central, http://pascal-central.com, a web site dedicated to Pascal programmers all over the world.

Programming language summary: *APL, Assembly, BASIC, C, C++, COBOL, DCL, EXEC, FORTRAN, Javascript, NOMAD, Pascal, Perl, RATFOR.*

## Origins of Pascal

The Pascal language was named for Blaise Pascal, a French mathematician who was a pioneer in computer development history. In 1641, at the age of eighteen, Pascal constructed the first arithmetical machine, arguably the first computer. He would improve upon the instrument eight years later. In 1650, Pascal left the world of geometry and physics, and shifted his focus towards religious studies, or, as Pascal wrote, to "contemplate the greatness and the misery of man." Pascal died in Paris on August 19, 1662.

The earliest computers were programmed in machine code and assembly. This type of programming is tedious and error prone, as well as extremely difficult to understand and modify. Programming is a time-consuming and expensive process. High level languages were developed to resolve this problem. High level languages provide a set of instructions that read like English, but can be translated by a program called a compiler into machine code. Pascal is one such language.

Other high level languages developed in the early years of the computer were FORTRAN (1957), COBOL (1959), ALGOL (1960), APL (1962), BASIC (1964), C (1972) and Ada (1983), to name a few. One problem with many of the early languages (e.g., FORTRAN and BASIC) was the heavy dependency on the use of "goto" instructions. "Goto" instructions tell the computer to jump from one step to another, enabling the computer to skip steps or to go back to repeat earlier steps. This type of sporadic branching increases the difficulty of debugging code. Additionally, languages like COBOL were designed with over-elaborate definitions, weak data structures support, and a lack of flexibility, making programs tedious to code and difficult to enhance.

Niklaus Wirth completed development of the original Pascal programming language in 1970. He based it upon the block structured style of the Algol programming language. There were two original goals for Pascal. According to the Pascal Standard (ISO 7185), these goals were to a) make available a language suitable for teaching programming as a systematic discipline based on fundamental concepts clearly and naturally reflected by the language, and b) to define a language whose implementations could be both reliable and efficient on then-available computers.

Pascal went far beyond its original design goals, with commercial use of the language often exceeding academic interest. Pascal provides rich data structures, including both the enumerated and record data types, and defined with a pleasing and powerful clarity. It provided an orthogonal and recursive approach to data structures, with arrays of arrays, arrays of records, records containing arrays, files of records, files of arrays, files of records containing arrays of records, and so on. Pascal's popularity exploded in the 1970's, as it was used in writing both system and application software. For this reason, the International Standards committee decided that a formal standard was needed to promote the stability of the Pascal language (the ISO 7185 Pascal Standard was originally published in 1983). By the end of the 1970's, more than 80 computer systems had Pascal implementations in use.

One of the more popular Pascal's of the 1970's and early 1980's was UCSD Pascal on the UCSD P-System operating system. The UCSD P-System was developed at the

Institute for Information Studies at the University of California - San Diego, under the direction of Kenneth Bowles. In fact, the P-System operating system itself was written in UCSD Pascal. As Wirth writes in his 1985 Turing Award Lecture, F*rom Programming Language Design To Computer Construction*, "But Pascal gained truly widespread recognition only after Ken Bowles in San Diego recognized that the P-system could well be implemented on the novel microcomputers. His efforts to develop a suitable environment with integrated compiler, filer, editor, and debugger caused a breakthrough: Pascal became available to thousands of new computer users who were not burdened with acquired habits or stifled by the urge to stay compatible with software of the past."

In 1978, Richard Gleaves and Mark Allen, working on-campus in San Diego, used UCSD Pascal to develop the 6502 interpreter which became the basis for Apple Pascal. By the 1980's, Pascal was used by most universities to teach programming, while still invading the commercial markets. It became so popular that even FORTRAN began to change, taking advantage of Pascal's innovations.

Due to the strong popularity of the Pascal language in system and application software development, and in response to the many cited drawbacks of the original Pascal implementation, an Extended Pascal evolved to address the needs of commercial development. In 1990, the ISO 10206 Extended Pascal Standard was published to support this new version of the language.

In addition to Extended Pascal, in 1986, Apple Computer released the first Object Pascal implementation, a version of its Apple Pascal that supported object-oriented programming. In 1993, the Pascal Standards Committee published an *Object-Oriented Extensions to Pascal* technical report which was based upon Apple's Object Pascal implementation.

---

Return to Table of Contents Next Chapter

---