What is Pascal?
Part 1

“It is a High Level Language that was designed for teaching purposes, and is a very structured and syntactically-strict language”.

Origins

Pascal grew out of ALGOL, a programming language intended for scientific computing. Meeting in Zurich, an international committee designed ALGOL as a platform independent language. This gave them comparatively free rein in the features they could design into ALGOL, but also made it more difficult to write compilers for it. Those were the days when many computers lacked hardware features that we now take for granted. The lack of compilers on many platforms, combined with its lack of pointers and many basic data types such as characters, led to ALGOL not being widely accepted.

Scientists and engineers flocked to FORTRAN, a programming language which was available on many platforms. ALGOL mostly faded away except as a language for describing algorithms.

Wirth Invents Pascal

In the 1960s, several computer scientists worked on extending ALGOL. One of these was Dr. Niklaus Wirth of the Swiss Federal Institute of Technology (ETH-Zurich), a member of the original group that created ALGOL. In 1971, he published his specification for a highly-structured language which resembled ALGOL in many ways. He named it Pascal after the 17th-century French philosopher and mathematician who built a working mechanical digital computer. Pascal is very data-oriented, giving the programmer the ability to define custom data types. With this freedom comes strict type-checking, which prevented data types from being mixed up. Pascal was intended as a teaching language, and was widely adopted as such. Pascal is free-flowing, unlike FORTRAN, and reads very much like a natural language, making it very easy to understand code written in it.

UCSD Pascal

One of the things that killed ALGOL was the difficulty of creating a compiler for it. Dr. Wirth avoided this by having his Pascal compiler compile to an intermediate, platform independent object code stage. Another program turned this intermediate code into executable code. Prof. Ken Bowles at the University of California at San Diego (UCSD) seized on the opportunity this offered to adapt the Pascal compiler to the Apple II, the most popular microcomputer of the day. UCSD P-System became a standard, and was widely used at universities. This was aided by the low cost of Apple II's compared to mainframes, which were necessary at the time to run other languages such as FORTRAN. Its impact on computing can be seen in IBM's advertisements for its revolutionary Personal Computer, which boasted that the PC supported three operating systems: Digital Research's CP/M-86, Softech's UCSD P-system, and Microsoft's PC-DOS.
Pascal Becomes Standard
By the early 1980's, Pascal had already become widely accepted at universities. Two events conspired to make it even more popular. First, the Educational Testing Service, the company which writes and administers the principal college entrance exam in the United States, decided to add a Computer Science exam to its Advanced Placement exams for high school students. For this exam, it chose the Pascal language. Because of this, secondary-school students as well as college students began to learn Pascal. Pascal remained the official language of the AP exams until 1999, when it was replaced by C++, which was quickly replaced by Java. Second, a small company named Borland International released the Turbo Pascal compiler for the IBM Personal Computer. The compiler was designed by Anders Hejlsberg, who would later head the group at Microsoft that developed C# and (re)introduced Managed Code back to the world of computing. Turbo Pascal was truly revolutionary. It did take some shortcuts and made some modifications to standard Pascal, but these were minor and helped it achieve its greatest advantage: speed. Turbo Pascal compiled at a dizzying rate: several thousand lines per minute. At the time, the available compilers for the PC platform were slow and bloated.

When Turbo Pascal came out, it was a breath of fresh air. Soon, Turbo Pascal became the de facto standard for programming on the PC. When PC Magazine published source code for utility programs, it was usually in either Assembly or Turbo Pascal.

At the same time, Apple came out with its Macintosh series of computers. As Pascal was the preeminent structured programming language of the day, Apple chose Pascal as the standard programming language for the Mac. When programmers received the API and example code for Mac programming, it was all in Pascal.

Extensions
From version 1.0 to 7.0 of Turbo Pascal, Borland continued to expand the language. One of the criticisms of the original version of Pascal was its lack of separate compilation for modules. Dr. Wirth even created a new programming language, Modula-2, to address that problem. Borland added modules to Pascal with its units feature. By version 7.0, many advanced features had been added. One of these was DPMI (DOS Protected Mode Interface), a way to run DOS programs in protected mode, gaining extra speed and breaking free of the 640K barrier for accessing memory under DOS. Turbo Vision, a text-based windowing system, allowed programmers to create great-looking interfaces in practically no time at all. Pascal even became object-oriented, as version 5.5 adopted the Apple Object Pascal extensions. When Windows 3.0 came out, Borland created Turbo Pascal for Windows, bringing the speed and ease of Pascal to the graphical user interface. It seemed that Pascal's future was secure.

The World Changes
However, this was not to be. In the 1970s, Dennis Ritchie and Brian Kernighan of AT&T Bell Laboratories created the C Programming Language. Ritchie then collaborated with Ken Thompson to design the UNIX operating system. At the time, AT&T had a government-sanctioned monopoly on telephone service in the United States. In return for the monopoly, its telephone business was regulated and it was prohibited from entering the computer business. AT&T, seeing no market for a research operating system, gave UNIX away to universities for free, complete with source code. Thus, a whole generation of computer science students learned C in their university courses on languages and operating systems. Slowly but surely, C began to filter into the computer programming world.
Pascal was finally killed by object orientation and the move to Windows on the industry standard PC platform. In the 1980s, Bjarne Stroustrup, also of Bell Labs, popularized object-orientation by developing C++, which kept the familiar syntax of C while extending it for object orientation. C++ came to define object orientation to a generation of programmers, and remains a strong force even today. Also in the 1980s, Microsoft Windows adopted C as its standard programming language. In contrast to MacOS and Pascal, the Windows API samples were all in K&R (pre-ANSI) C, complete with variable lists after the function prototype. As object orientation and Windows took hold, the natural language for applications migrating to Windows was C++. Many colleges and universities moved away from Pascal, choosing C++ or newer languages for their programming courses. Finally, the AP exam moved to C++, ending Pascal's dominance in American high schools.

**So Why Learn Pascal?**

Despite its fading away as a de facto standard, Pascal is still quite useful. C and C++ are very symbolic languages. Where Pascal chooses words (e.g. begin-end), C/C++ instead uses symbols ({-}). Also, C was designed for systems programming. In Pascal, mixing types leads to an error and is very infrequently done. In C/C++, type-casting and pointer arithmetic is common, making it easy to crash programs and write in buffer overruns. When the AP exam switched to C++, only a subset of C++ was adopted. Many features, like arrays, were considered too dangerous for students, and ETS provided its own "safe" version of these features. Another reason: speed and size. The Borland Pascal compiler is still lightning-fast. Borland has revitalized Pascal for Windows with Delphi, a Rapid-Application-Development environment. Instead of spending several hours writing a user interface for a Windows program in C/C++, you could do it in ten minutes with Delphi’s graphical design tools. Delphi is to Pascal what Visual BASIC did to BASIC. Borland is still developing Delphi, and the open-source community has created a largely Borland compatible compiler called Free Pascal.

Also, Pascal remains preferred at many universities, especially in areas where students are first exposed to computers at school rather than at home. In addition, Pascal was well-suited for teaching programming, and remains so. There is less overhead and fewer ways for a student to get a program into trouble. For teaching simple procedural programming, Pascal remains a good choice. Pascal has hung on longer in education outside the United States, and remains an official language of the International Informatics Olympiad. A basic programming background is useful in many technical occupations, and the overhead of learning an object-oriented language is not necessarily the best application of resources. Thus, even after C, C++, and Java took over the programming world, Pascal retains a niche in the market. Many small-scale freeware, shareware, and open-source programs are written in Pascal/Delphi. So enjoy learning it while it lasts. It's a great introduction to computer programming. It's not scary like C, dangerous like C++, or abstract like Java. In another twenty years, you'll be one of the few computer programmers to know and appreciate Pascal.

**Teacher's Notes:** The reason why I chose to give you this in-depth information about Pascal; it can serve as a reference for your future programming development skills and also serves as a foundation for your understanding and appreciation for this language that CXC has assigned to the CSEC Information Technology Course. Be vigilant in your reading and make your own personal notes as you go along.
When talking about computer languages, there are basically three major terms that will be used.

1. **Machine language** -- actual binary code that gives basic instructions to the computer's CPU. These are usually very simple commands like adding two numbers or moving data from one memory location to another.

2. **Assembly language** -- a way for humans to program computers directly without memorizing strings of binary numbers. There is a one-to-one correspondence with machine code. For example, in Intel x86 machine language, ADD and MOV are mnemonics for the addition and move operations.

3. **High-level language** -- permits humans to write complex programs without going step-by-step. High-level languages include Pascal, C, C++, FORTRAN, Java, BASIC, and many more. One command in a high level language, like writing a string to a file, may translate to dozens or even hundreds of machine language instructions.

Microprocessors can only run machine language programs directly. Assembly language programs are assembled, or translated into machine language. Likewise, programs written in high-level languages, like Pascal, must also be translated into machine language before they can be run. To do this translation is to compile a program.

The program that accomplishes the translation is called a compiler. This program is rather complex since it not only creates machine language instructions from lines of code, but often also optimizes the code to run faster, adds error-correction code, and links the code with subroutines stored elsewhere.

For example, when you tell the computer to print something to the screen, the compiler translates this as a call to a pre-written module. Your code must then be linked to the code that the compiler manufacturer provides before an executable program results.

With high-level languages, there are again three basic terms to remember:

1. **Source code** -- the code that you write. This typically has an extension that indicates the language used. For example, Pascal source code usually ends in ".pas" and C++ code usually ends in ".cpp"

2. **Object code** -- the result of compiling. Object code usually includes only one module of a program, and cannot be run yet since it is incomplete. On DOS/Windows systems, this usually has an extension of ".obj"

3. **Executable code** -- the end result. All the object code modules necessary for a program to function are linked together. On DOS/Windows systems, this usually has an extension of ".exe", exe stands for executable.

This ends part 1 of the documentation, in part 2 we will discuss the types of compilers to use and dive in the syntax of the language. Please be sure to email me at mrj@stmonicascollege.org for further info. A lot of notes that I have compiled were taken from taoyue.com, they are a very helpful resource.