© LJohn Ceballos / SIS

Speaking in Tongues

Choosing the right programming language can be a trial, especially given their complex history. Resident language guru **Kevlin Henney** translates some of the jargon

THE DEMISE of the programming language has been fashionably predicted every decade since programming rose from first-generation machine code to second generation assembly language. But programming languages are as popular now as they ever have been, and are just as rich a source of disagreement among developers: language of choice, like editor of choice, still ranks high as something of a religious issue.

The current industry emphasis is not so much on the use of single languages for monolithic projects, but on the development and integration of components. This increasingly demands that developers obtain multilingual skills. They must also acquire the technical knowledge of many different tools, APIs, and languages, as well as the idioms and culture surrounding each of these.

C

The C programming language was originally created in 1972 as a high-level systems programming language. Ken Thompson of AT&T was after a language that could be used to portably develop much of the kernel, and all of the shell tools, for what became UNIX. He experimented with BCPL, developed by Martin Richards of Cambridge University, and evolved a simple variant known as B. Dennis Ritchie took this further and created C.

This pairing of UNIX and C, along with the distribution of UNIX to academic institutions, ensured that C became popular as a general-purpose programming language. This effect was felt on other platforms, most recently and noticeably Microsoft Windows and NT.

C has spawned a strong developer culture through its history. Characteristics of this culture include encouragement of structured programming, shunning of language extensions, a strong computer science bias, and a respect for efficiency. As a language, it is characterised by a syntax that emphasises brevity. This has given C something of a 'write-only' reputation that is encouraged, if anything, by the annual International Obfuscated C Coding Contest (IOCCC).

The emphasis on portability saw relatively little divergence in C dialects from the original informal standard, known commonly as K&R C after the authors of the original and definitive book *The C Programming Language*.[1] Some divergence and the increasing popularity of the language were enough to spawn a standardisation effort, which resulted in ANSI C in 1989. It was adopted as an ISO standard in 1990, and the ANSI standard was withdrawn as an alternative in 1991. In 1994 an addendum to the ISO standard improved its support for internationalisation.

In systems programming it has all but displaced assembler and other high-level systems languages as the preferred tool. Assembler now plays a supporting rather than a major role. Its rise as a general purpose programming language has also meant that it has become something of a *lingua franca* in programming: C is a common and expected sight on any developer's CV.

C is the principal language of APIs and, in addition to its wide availability, is the reason for its continued popularity. A new standard, dubbed C9X, is due out before 2000. This next increment will build on the previous standard, borrowing a couple of minor features from C++ and adding more support for numeric work, as well as a whole number of minor linguistic and library extensions.

C++

C++ started life in the early 1980s as 'C with Classes', acquiring its name in 1984, and first being released to the public in 1985. It was developed by Bjarne Stroustrup at AT&T, based on his experience of Simula 67, BCPL, Algol 68 and, latterly, C.

The language has gone through three stages in its life, each roughly corresponding to an edition of Stroustrup's book, *C++ Programming Language*.[2] The early language was very much an early adopter's language, and it appealed to a number of C programmers, typically under UNIX. The second stage saw the rise of C++ on the PC, and the acceptance of C++ into the mainstream. Each stage is characterised by new language features and techniques, but it is perhaps the third stage that exhibits this most. A C++ ISO standard was ratified in early August, and sees the consolidation of a number of language and library features. The effect of these features, introduced over the last five years, is to simplify many programming tasks. The effect of the standard is to stabilise the language, giving both vendors and users a chance to catch up with features and the new style.

Invented in the early 1990s by Jim Gosling and originally called Oak, Java was released with much fanfare to the public in 1995 as a language for the Web

There is little doubt that it is now a mainstream general purpose language. This can be gauged from its top position in the skills demand table—which makes it something of a CV++ language—and its use across almost every domain on almost every platform. In some domains, C++'s complexity has been a deterrent. C has a far stronger foothold in the embedded systems market than C++. This seems strangely at odds with C++'s systems programming roots, and is being addressed on two opposing fronts: a subset definition of C++ (Embedded C++) aimed solely at the embedded market and improved compiler, linker, and library optimisation.

Java

There has been more written about Java in the trade press than perhaps any other language. Invented in the early 1990s by Jim Gosling at Sun, and originally called Oak, it was released with much fanfare to the public in 1995 as a language for the Web. Gosling describes it as "C++ without the guns, knives, and clubs", referring to its simplified design and safe memory model.

Java is an umbrella term covering a set of technologies: the language is one part; the Java Virtual Machine (JVM) provides an execution environment suited to the Java language, and the extensive runtime libraries form the third main part. The JVM is not tied to the Java language, and it can be used to execute programs written in other languages. For example, Intermetrics Ada 95 compiler compiles to JVM byte code.

The design of the language, libraries, and machine has been strictly guided by Sun, but in the near future it is likely to be recognised as an ISO standard through Sun's acceptance by ISO as a PAS (Publicly Available Specification) submitter. Yet to be resolved—and this will be done by lawyers rather than programmers or users—is the dispute between Sun and Microsoft over Microsoft's proprietary implementation of Java technologies. Sun and HP are also at odds over HP's embedded implementation of the JVM, which contradicts Sun's own embedded Java strategy.

And there is the inevitable Java question: will Java kill off C++? There are a number of ways to answer this question. The first is to look at history. Although many languages have bitten the dust in the last few decades, there are fewer that have reached widespread popularity and then become extinct, and fewer still that have become popular and then died by being completely displaced. Another perspective is to analyse the skills market. The uptake in Java has been meteoric, but the demand for C++ skills continues to rise and heads the table.

Java offers promise as a general-purpose programming language for almost any existing domain, but its most significant successes are likely to be those that define new markets, or in relatively youthful or rapidly growing domains. Smart cards and embedded technology in general are examples of this, and this return to Java's roots will certainly contribute more to its uptake in the long run than the Web.

Smalltalk

Smalltalk was the product of a decade of development at Xerox's Palo Alto Research Center (PARC) led by Alan Kay. Smalltalk 80 was the first major public release of the language, and showed it to be a fully object-oriented language from the ground up. It also supported a rich development environment that was ahead of its time, supporting class and method browsing panels in a full point-and-click environment.

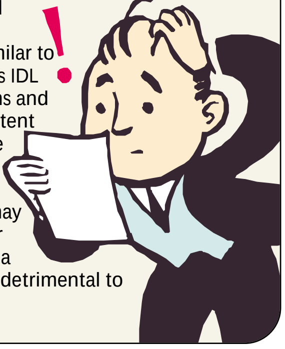
Although its OO model has often been the sole basis for praise of the language, its syntax has deterred a number of developers.

Idioms—Breaking the Language Barrier

Many programming languages and development environments inspire a developer culture particular to that language, resulting in a set of well understood idioms and common practices. Idioms are language-level patterns, where patterns are documented and empirical solutions to problems in a given context.[3]

All patterns are described as having a context in which they are applicable. For idioms the language is a part of this context[4,5]. Idioms are, literally, what the locals speak. When you start learning a language, part of the challenge is to learn the syntax and semantics of the language. Another part, as with learning a natural language, is to develop a familiarity with the common way in which it is used.

Developers working in many languages on a single project face a challenge in balancing both the need for design consistency and the need for appropriate local use of language. This is especially true when they are bridging through declarative middleware languages such as IDL (Interface Definition Language). Although derived from C++, and therefore also similar to C and Java, OMG's IDL has its own idioms and has a style and intent different than the implementation and client languages that may be used on either side of it. Indeed, a C++ style can be detrimental to IDL design.



While it is perfectly logical and extremely consistent, it has been seen by many as a major stumbling block in a relatively conservative computing industry. Smalltalk's early versions proved to be inefficient in terms of memory usage and garbage collection strategy; a reputation that has stayed with it despite significant evolution.

It has long been poised for the big time, but just as it appeared to pick up momentum a couple of years ago, Java arrived. For all intents and purposes, Java has killed off any chances of a Smalltalk revival, and by implication, Smalltalk itself. A tick list between Java and Smalltalk looks much the same, and their underlying object models have much in common. However, the mindshare, syntax, vendor support, and growing skills pool make Java a more strategic choice. Smalltalk vendors have become strongly caffeinated, adapting their development tools with relative ease from Smalltalk to Java.

Although its OO model has often been the sole basis for praise of the language, Smalltalk's syntax has deterred a number of developers

Pascal

Pascal was developed by Niklaus Wirth in 1971 as a language for teaching programming, and it is here that Pascal has had its greatest impact. From that base, it has also grown as a commercial language, but not so that it has ever been considered dominant.

So, whither Pascal? The once popular combination of Pascal and Assembler for systems programming has been replaced by C and C++. The use of Pascal in high-integrity systems has largely been displaced by Ada, a language derived from Pascal. In education, where it has been so prevalent, Pascal's position in the last decade has been eroded at different times by Modula 2, Ada, C, C++, and now, Java. Pascal's role as the *lingua franca* of algorithms and, indeed, programming, has also been undermined by the C family. The last Pascal standardisation effort came and went without either vendors or users noticing it.

This paints a pretty bleak picture, but, in truth, Pascal's future seems secure in Delphi's Object Pascal. Ultimately, this has evolved from Borland's original and classic Turbo Pascal, but as a tool has been focused on RAD and client server development based around a cleanly designed library and intuitive development environment.

BASIC

BASIC was conceived as a teaching language for students in nonnumerate disciplines. John Kemeny and Thomas Kurtz developed the language in 1964 for this purpose, and it has grown from there. It was the first programming system to take advantage of time sharing and remote terminal access. At the other end of the scale, BASIC was popularised in the late 1970s and early 1980s on home computers. In this environment, BASIC was the starting point for Bill Gates's fortunes and the rest, as they say, is history.

Its growth has not been without pains and criticism. Traditional BASIC lacked adequate data, control, and program structuring facilities to be considered as a viable language for large-scale development. Given its origins and features, BASIC acquired a reputation as something of a toy language. This has not stopped it from being used heavily for scripting on PCs or for business calculations, often as a competitor to COBOL. With these environmental influences, BASIC has splintered into many dialects, each supporting proprietary extensions.

With one very notable exception, the market for BASIC has shrunk and will continue to do so. That notable exception is Microsoft's Visual Basic, which encompasses both a visual development environment and, since the early 1990s, has evolved into the language that takes on board better support for structured programming, data structuring, and OO facilities.

COBOL

COBOL started life as an interim measure. The original version was designed by

The Language Graveyard

There are many languages that have fallen by the wayside, and listing them all is far from practical. However, there have been a few languages in particular that were destined for the big time—if the hype surrounding them was anything to go by—but have only reached the limelight, and are now beginning to fade. Many of these languages have an ardent, die-hard core of supporters. However, in commercial terms we would consider them dead. Their curve is either diving or has flattened out at low altitude, and their successes, if any, are behind them.

To some degree, the popularity or demise of a particular language is tied to the popularity of its language model, and vice versa. For instance, procedural languages such as C, FORTRAN, and COBOL have at various times won the hearts, minds, and legacy of developers and corporates, outlasting the elegant but ivory tower purity of functional and logic programming languages. Performance issues, and the lack of a legacy-installed base (either of code or skills), have often isolated newer languages from commercial developers.

Logic programming—typified by Prolog—was based on the belief that a program's requirements could be expressed in a declarative manner using predicate logic without procedural implementation detail, and that such a specification could be executed directly via an inference engine. This approach, fostered in the warm glow surrounding artificial intelligence (AI) in the late 1970s and early 1980s, underpinned Japan's ambitious Fifth Generation computing programme and the UK's more modest Alvey programme. The most notable survivor of the logic-based approach is SQL (Structured Query Language).

Functional programming was another paradigm tipped for great things. The oldest and best known language in this category is LISP, which was originally developed in 1960 by John McCarthy, and is the darling of the AI community. The ANSI Common LISP standard is perhaps the definitive, all-encompassing version. This appeared in 1984, in time to witness the demise of LISP's chances in the mainstream as the AI bubble burst and a whole slew of newer technologies and techniques captured the market. The most visible survivor in this category is ELISP, which is both the scripting language and the implementation language for GNU Emacs.

What helped to sweep away the idealism of the declarative languages was a shift in development scale and platform—the large user-focused applications deployed on graphical workstations and PCs—and the arrival of OO technologies in the mainstream. The popularity of particular languages supporting OO has often built on the popularity of their relatives, such as Java on C++, and C++ on C.

For other languages, there has always been great interest, but they have inevitably been left out in the cold. CLOS (Common LISP Object System) was often seen as offering an alternative to the conventional OO model. The Dylan language from Apple has many conceptual similarities with CLOS, but in the search for perfection has fumbled the ball. Conceived in the early 1990s, it has spent most of its life on the drawing board. Eiffel is often held up as a pure, strongly typed, procedural OO language developed by Bertrand Meyer in the mid-1980s. Its greatest success has been as the vehicle of Meyer's two editions of *Object-Oriented Software Construction*.[6] In terms of actual presence, however, it has never really appeared on the radar.

committee, strongly influenced by the work of the late Grace Hopper, with the expectation that the language they were developing would temporarily fill the gap in the business programming arena until a better language was developed by another committee. COBOL's durability and widespread use—perhaps the most of any language—bears witness to the fact that in programming, there is no such thing as a temporary fix.

Although strong on data structuring, COBOL has often been held up as a counterexample of programming language design with respect to its support for structured control flow and modularity. The remit that it should be based on English has led to a language often criticised for its verbosity; that it should be comprehensible by managers has been realised only in COBOL programmers promoted to management positions.

The suitability for the business environment perhaps supports a claim that COBOL can be considered the first 4GL. The main installed base is in the IT departments of medium-to-large businesses, rather than in the small business market that has tended to use off-the-shelf packages, a more PC-based environment, and tools such as Visual Basic.

The weaknesses have been addressed to some extent with each passing standard of the language—1968, 1974, and 1985, with a new increment soon promising OO support—but whether such a radical change in direction will be embraced in practice by the existing COBOL community genuinely remains to be seen. The current growth in the skills market for COBOL, and also a source for renewed criticism of the language, is clearly the year 2000 problem. The renewed, if unintended and unavoidable, investment in such code is likely to assure COBOL's continued place in the new millennium.

FORTRAN

FORTRAN started life as an experiment. In 1954, John Backus and his team at IBM started developing compiled language for the IBM 704. The design aims of the language included the concept of formula translation, from which FORTRAN derives its name, and generated code that was close to handcrafted assembler in its efficiency.

By modern language standards, the resemblance of FORTRAN to actual mathematical expressions is remote at best, but at the time it was close enough to attract the scientific and engineering community, where it still has its greatest stronghold.

Traditional BASIC lacked adequate data, control, and program structuring facilities to be considered as a viable language for large-scale development. Given its origins and features, BASIC acquired a reputation as something of a toy language.

In the 1960s and 1970s, the language evolved. One of the most widely used versions was identified by its IBM version number, FORTRAN IV. FORTRAN 77 was the standard that saw the language acquire some modern language features, such as structured `IF ELSE` and string handling. The FORTRAN 90 standard can be characterised as something old, something new, something borrowed, and something Big Blue: although it builds on FORTRAN 77, an idiomatic FORTRAN 90 program bears little resemblance to what many would recognise as FORTRAN. FORTRAN 90 includes support for data structures, full structured control flow, dynamic memory, pointers, and modules.

Although still taught and widely used, FORTRAN has become something of a niche language. It used to be considered a general-purpose language, but its sole advocates are now to be found in science and engineering. It is a classic legacy language, with a large installed base in a particular community that itself is becoming increasingly multilingual. The uptake of FORTRAN 90 has not been as great as that of FORTRAN 77, with competition from C, C++, and other specifically targeted tools, like Mathematica.

Conclusion

The deliverance from coding promised by 4GL, visual programming environments, and code generators has failed to materialise. All the effort saved in developing one part of a system is either pushed to another or raises the expectation of functionality, which in turn increases the amount of code.

The consolidation of the languages market has also failed to happen. Although there have been attempts at a final solution (PL/1, Algol 68, Ada), none have prevailed. In the 1970s there was something of a Cambrian explosion in languages that only a few languages survived. It was expected that this trend would continue and the market would cluster around only a few mainstream languages.

In the late 1990s, we can see that this static equilibrium is unlikely to be achieved, and that the market is experiencing a more dynamic equilibrium. Older languages, such as FORTRAN and COBOL, sit cheek by jowl with the more modern C++ and Java, and there is a proliferation of 'little languages' in the form of script languages, inspired by the rise of the Web. Proprietary and domain-specific languages and environments are often on an equal, if not a more equal, footing with general purpose, open programming languages. ■

Kevlin Henney works for QA Training as a senior technologist, specialising in programming languages and architectures. He can be contacted at **khenney@qatraining.com**.

References

1. Kernighan, B., and D. Ritchie. *The C Programming Language*, Prentice Hall, 1978.
2. Stroustrup, B. *C++ Programming Language*, Addison–Wesley, 1985.
3. Gamma, E. *et al. Design Patterns: Elements of Reusable Object-Oriented Software*, Addison–Wesley, 1995.
4. Beck, K. *Smalltalk Best Practice Patterns*, Prentice Hall, 1997.
5. Coplien, J. *Advanced C++ Programming Styles and Idioms*, Addison–Wesley, 1992.
6. Meyer, B. *Object-Oriented Software Construction*, Prentice Hall, 1998.

Online Resources

- Yahoo's programming language pages. **www.yahoo.com/Computers_and_Internet/Programming_Languages**
- C++ information site, includes copies of the committee drafts for the standard. **www.maths.warwick.ac.uk/c++**
- Association of C and C++ Users. **www.accu.org**
- Embedded C++ home page. **www.caravan.net/ec2plus**
- Sun's Javasoft home page. **www.javasoft.com**
- Online version of Java Report. **www.javareport.com**