S.Sridhar* et al.
(IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH
Volume No.5, Issue No.2, February – March 2017, 5681-5704.

A Study On Various Programming Languages To Keep Pace With Innovation

S.SRIDHAR
Professor & Director RV Centre for Cognitive & Central Computing
R.V.College of Engineering,
Mysore Road Bangalore-560059 India

Abstract: **A programming language is a formal computer language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behaviour of a machine or to express algorithms. The earliest known programmable machine preceded the invention of the digital computer and is the automatic flute player described in the 9th century by the brothers Musa in Baghdad, "during the Islamic Golden Age". From the early 1800s, "programs" were used to direct the behavior of machines such as Jacquard looms and player pianos. Thousands of different programming languages have been created, mainly in the computer field, and many more still are being created every year. Many programming languages require computation to be specified in an imperative form (i.e., as a sequence of operations to perform) while other languages use other forms of program specification such as the declarative form (i.e. the desired result is specified, not how to achieve it). The description of a programming language is usually split into the two components of syntax (form) and semantics (meaning). Some languages are defined by a specification document (for example, the C programming language is specified by an ISO Standard) while other languages (such as Perl) have a dominant implementation that is treated as a reference. Some languages have both, with the basic language defined by a standard and extensions taken from the dominant implementation being common. An attempt is made in this paper to have a study on various programming languages.**

Key words: **Programming Language; Formal Computer Language; Instructions; ISO Standard;**

I. INTRODUCTION

The earliest computers were often programmed without the help of a programming language, by writing programs in absolute machine language. The programs, in decimal or binary form, were read in from punched cards or magnetic tape or toggled in on switches on the front panel of the computer. Absolute machine languages were later termed *first-generation programming languages* (1GL). The next step was development of so-called *second-generation programming languages* (2GL) or assembly languages, which were still closely tied to the instruction set architecture of the specific computer. These served to make the program much more human-readable and relieved the programmer of tedious and error-prone address calculations. The first *high-level programming languages*, or *third-generation programming languages* (3GL), were written in the 1950s. An early high-level programming language to be designed for a computer was Plankalkül, developed for the German Z3 by Konrad Zuse between 1943 and 1945. However, it was not implemented until 1998 and 2000. John Mauchly's Short Code, proposed in 1949, was one of the first high-level languages ever developed for an electronic computer. Unlike machine code, Short Code statements represented mathematical expressions in understandable form. However, the program had to be translated into machine code every time it ran, making the process much slower than running the equivalent machine code. The Manchester Mark 1 ran programs written in Autocode from 1952. At the University of Manchester, Alick Glennie developed Autocode in the early 1950s. A programming language, it used a compiler to automatically convert the language into machine code. The first code and compiler was developed in 1952 for the Mark 1 computer at the University of Manchester and is considered to be the first compiled high-level programming language.

II. BEGINNER'S ALL-PURPOSE SYMBOLIC INSTRUCTION CODE

BASIC (an acronym for **Beginner's All-purpose Symbolic Instruction Code**) is a family of general-purpose, high-level programming languages whose design philosophy emphasizes ease of use. In 1964, John G. Kemeny and Thomas E. Kurtz designed the original BASIC language at Dartmouth College in the U.S. state of New Hampshire. They wanted to enable students in fields other than science and mathematics to use computers. At the time, nearly all use of computers required writing custom software, which was something only scientists and mathematicians tended to learn. Versions of BASIC became widespread on microcomputers in the mid-1970s and 1980s. Microcomputers usually shipped with BASIC, often in the machine's firmware. Having an easy-to-learn language on these early personal computers allowed small business owners, professionals, hobbyists, and consultants to develop custom software on computers they could afford. In

the 2010s, BASIC remains popular in many computing dialects and in new languages influenced by BASIC, such as Microsoft's Visual Basic.

ORIGIN

BASIC language was released on May 1, 1964 by John G. Kemeny and Thomas E. Kurtz and implemented under their direction by a team of Dartmouth College students. The acronym *BASIC* comes from the name of an unpublished paper by Thomas Kurtz. BASIC was designed to allow students to write mainframe computer programs for the Dartmouth Time-Sharing System. It was intended specifically for less technical users who did not have or want the mathematical background previously expected. Being able to use a computer to support teaching and research was quite novel at the time. The language was based on FORTRAN II, with some influences from ALGOL 60 and with additions to make it suitable for timesharing. Initially, BASIC concentrated on supporting straightforward mathematical work, with matrix arithmetic support from its initial implementation as a batch language, and character string functionality being added by 1965. Wanting use of the language to become widespread, its designers made the compiler available free of charge. (In the 1960s, software became a chargeable commodity; until then, it was provided without charge as a service with the very expensive computers, usually available only to lease.) They also made it available to high schools in the Hanover area, and put considerable effort into promoting the language. In the following years, as other dialects of BASIC appeared, Kemeny and Kurtz's original BASIC dialect became known as *Dartmouth BASIC*.

VISUAL BASIC

BASIC's fortunes reversed once again with the introduction in 1991 of Visual Basic ("VB") by Microsoft. This was an evolutionary development of QuickBasic, and included constructs from other languages such as block structured control statements including "With" and "For Each", parameterized subroutines, optional static typing, and a full object oriented language. But the language retained considerable links to its past, such as the Dim statement for declarations, "Gosub"/Return statements, and even line numbers which were still needed to report errors properly. An important driver for the development of Visual Basic was as the new macro language for Microsoft Excel, a spreadsheet program. Ironically, given the origin of BASIC as a "beginner's" language, and apparently even to the surprise of many at Microsoft who still initially marketed it as a language for hobbyists, the language had come into widespread use for small custom business applications shortly after the release of VB version 3.0, which is widely considered the first relatively stable version. While many advanced programmers still scoffed at its use, VB met the needs of small businesses efficiently wherever processing speed was less of a concern than ease of development. By that time, computers running Windows 3.1 had become fast enough that many business-related processes could be completed "in the blink of an eye" even using a "slow" language, as long as large amounts of data were not involved. Many small business owners found they could create their own small, yet useful applications in a few evenings to meet their own specialized needs. Eventually, during the lengthy lifetime of VB3, knowledge of Visual Basic had become a marketable job skill. Microsoft also produced VBScript in 1996 and Visual Basic .NET in 2001. The latter has essentially the same power as C# and Java but with syntax that reflects the original Basic language.

FORTRAN

Fortran (formerly **FORTRAN**, derived from "Formula Translation") is a general-purpose, imperative programming language that is especially suited to numeric computation and scientific computing. Originally developed by IBM in the 1950s for scientific and engineering applications, Fortran came to dominate this area of programming early on and has been in continuous use for over half a century in computationally intensive areas such as numerical weather prediction, finite element analysis, computational fluid dynamics, computational physics, crystallography and computational chemistry. It is a popular language for high-performance computing and is used for programs that benchmark and rank the world's fastest supercomputers.

HISTORY

In late 1953, John W. Backus submitted a proposal to his superiors at IBM to develop a more practical alternative to assembly language for programming their IBM 704 mainframe computer. Backus' historic FORTRAN team consisted of programmers Richard Goldberg, Sheldon F. Best, Harlan Herrick, Peter Sheridan, Roy Nutt, Robert Nelson, Irving Ziller, Lois Haibt, and David Sayre. Its concepts included easier entry of equations into a computer, an idea developed by J. Halcombe Laning and demonstrated in the Laning and Zierler system of 1952. A draft specification for *The IBM Mathematical Formula Translating System* was completed by mid-1954. The first manual for FORTRAN appeared in October 1956, with the first FORTRAN compiler delivered in April 1957. This was the first optimizing compiler, because customers were reluctant to use a high-level programming language unless its compiler could generate code with performance comparable to that of hand-coded assembly language. While the

community was particular that this new method could possibly outperform hand-coding, it reduced the number of programming statements necessary to operate a machine by a factor of 20, and quickly gained acceptance. John Backus said during a 1979 interview with *Think*, the IBM employee magazine, "Much of my work has come from being lazy. I didn't like writing programs, and so, when I was working on the IBM 701, writing programs for computing missile trajectories, I started work on a programming system to make it easier to write programs." The language was widely adopted by scientists for writing numerically intensive programs, which encouraged compiler writers to produce compilers that could generate faster and more efficient code. The inclusion of a complex number data type in the language made Fortran especially suited to technical applications such as electrical engineering. By 1960, versions of FORTRAN were available for the IBM 709, 650, 1620, and 7090 computers. Significantly, the increasing popularity of FORTRAN spurred competing computer manufacturers to provide FORTRAN compilers for their machines, so that by 1963 over 40 FORTRAN compilers existed. For these reasons, FORTRAN is considered to be the first widely used programming language supported across a variety of computer architectures. The development of FORTRAN paralleled the early evolution of compiler technology, and many advances in the theory and design of compilers were specifically motivated by the need to generate efficient code for FORTRAN programs. The initial release of FORTRAN for the IBM 704 contained 32 statements.

FORTRAN II

IBM's *FORTRAN II* appeared in 1958. The main enhancement was to support procedural programming by allowing user-written subroutines and functions which returned values, with parameters passed by reference. The COMMON statement provided a way for subroutines to access common (or global) variables. Six new statements were introduced:

- SUBROUTINE, FUNCTION, and END
- CALL and RETURN
- COMMON

Over the next few years, FORTRAN II would also add support for the DOUBLE PRECISION and COMPLEX data types. Early FORTRAN compilers supported no recursion in subroutines. Early computer architectures supported no concept of a stack, and when they did directly support subroutine calls, the return location was often stored in one fixed location adjacent to the subroutine code, which does not permit a subroutine to be called again before a prior call of the subroutine has returned. Although not specified in Fortran 77, many F77 compilers supported recursion as an option, while it became a standard in Fortran 90.

FORTRAN III

A FORTRAN coding form, printed on paper and intended to be used by programmers to prepare programs for punching onto cards by keypunch operators. Now obsolete. IBM also developed a *FORTRAN III* in 1958 that allowed for inline assembly code among other features; however, this version was never released as a product. Like the 704 FORTRAN and FORTRAN II, FORTRAN III included machine-dependent features that made code written in it unportable from machine to machine. Early versions of FORTRAN provided by other vendors suffered from the same disadvantage.

IBM 1401 FORTRAN

FORTRAN was provided for the IBM 1401 computer by an innovative 63-phase compiler that ran entirely in its core memory of only 8000 (6-bit) characters. The compiler could be run from tape, or from a 2200-card deck; it used no further tape or disk storage. It kept the program in memory and loaded overlays that gradually transformed it, in place, into executable form, as described by Haines. and in IBM document C24-1455. The executable form was not entirely machine language; rather, floating-point arithmetic, subscripting, input/output, and function references were interpreted, anticipating UCSD Pascal P-code by two decades. IBM later provided a FORTRAN IV compiler for the 1400 series of computers, described in IBM document C24-3322.

FORTRAN IV

Starting in 1961, as a result of customer demands, IBM began development of a *FORTRAN IV* that removed the machine-dependent features of FORTRAN II (such as READ INPUT TAPE), while adding new features such as a LOGICAL data type, logical Boolean expressions and the *logical IF statement* as an alternative to the *arithmetic IF statement.* FORTRAN IV was eventually released in 1962, first for the IBM 7030 ("Stretch") computer, followed by versions for the IBM 7090, IBM 7094, and later for the IBM 1401 in 1966. By 1965, FORTRAN IV was supposed to be compliant with the *standard* being developed by the American Standards Association X3.4.3 FORTRAN Working Group. At about this time FORTRAN IV had started to become an important educational tool and implementations such as the University of Waterloo's WATFOR and WATFIV were created to simplify the complex compile and link processes of earlier compilers.

FORTRAN 66

Perhaps the most significant development in the early history of FORTRAN was the decision by the *American Standards Association* (now American National Standards Institute (ANSI)) to form a committee sponsored by BEMA, the Business Equipment Manufacturers Association, to develop an *American Standard Fortran*. The resulting two standards, approved in March 1966, defined two languages, *FORTRAN* (based on FORTRAN IV, which had served as a de facto standard), and *Basic FORTRAN* (based on FORTRAN II, but stripped of its machine-dependent features). The FORTRAN defined by the first standard, officially denoted X3.9-1966, became known as *FORTRAN 66* (although many continued to term it FORTRAN IV, the language on which the standard was largely based). FORTRAN 66 effectively became the first industry-standard version of FORTRAN.

FORTRAN 77

FORTRAN-77 program with compiler output, written on a CDC 175 at RWTH Aachen University, Germany, in 1987 . 4.3 BSD for the Digital Equipment Corporation (DEC) VAX, displaying the manual for FORTRAN 77 (f77) compiler. After the release of the FORTRAN 66 standard, compiler vendors introduced several extensions to *Standard Fortran*, prompting ANSI committee X3J3 in 1969 to begin work on revising the 1966 standard, under sponsorship of CBEMA, the Computer Business Equipment Manufacturers Association (formerly BEMA). Final drafts of this revised standard circulated in 1977, leading to formal approval of the new FORTRAN standard in April 1978. The new standard, called *FORTRAN 77* and officially denoted X3.9-1978, added a number of significant features to address many of the shortcomings of FORTRAN 66. In this revision of the standard, a number of features were removed or altered in a manner that might invalidate formerly standard-conforming programs. *(Removal was the only allowable alternative to X3J3 at that time, since the concept of "deprecation" was not yet available for ANSI standards.)* While most of the 24 items in the conflict list (see Appendix A2 of X3.9-1978) addressed loopholes or pathological cases permitted by the prior standard but rarely used, a small number of specific capabilities were deliberately removed.

FORTRAN 90

The much delayed successor to FORTRAN 77, informally known as *Fortran 90* (and prior to that, *Fortran 8X*), was finally released as ISO/IEC standard 1539:1991 in 1991 and an ANSI Standard in 1992. In addition to changing the official spelling from FORTRAN to Fortran, this major revision added many new features to reflect the significant changes in programming practice that had evolved since the 1978 standard.

FORTRAN 95

Fortran 95, published officially as ISO/IEC 1539-1:1997, was a minor revision, mostly to resolve some outstanding issues from the Fortran 90 standard. Nevertheless, Fortran 95 also added a number of extensions, notably from the High Performance Fortran specification:

- FORALL and nested WHERE constructs to aid vectorization
- User-defined PURE and ELEMENTAL procedures
- Default initialization of derived type components, including pointer initialization
- Expanded the ability to use initialization expressions for data objects
- Initialization of pointers to NULL()
- Clearly defined that ALLOCATABLE arrays are automatically deallocated when they go out of scope.

A number of intrinsic functions were extended (for example a dim argument was added to the maxloc intrinsic). Several features noted in Fortran 90 to be "obsolescent" were removed from Fortran 95:

- DO statements using REAL and DOUBLE PRECISION index variables
- Branching to an END IF statement from outside its block
- PAUSE statement
- ASSIGN and assigned GO TO statement, and assigned format specifiers
- H edit descriptor.

An important supplement to Fortran 95 was the ISO technical report *TR-15581: Enhanced Data Type Facilities*, informally known as the *Allocatable TR*. This specification defined enhanced use of ALLOCATABLE arrays, prior to the availability of fully Fortran 2003-compliant Fortran compilers. Such uses include ALLOCATABLE arrays as derived type components, in procedure dummy argument lists, and as function return values. (ALLOCATABLE arrays are preferable to POINTER-based arrays because ALLOCATABLE arrays are guaranteed by Fortran 95 to be deallocated automatically when they go out of scope, eliminating the possibility of memory leakage. In addition, elements of allocatable arrays are contiguous, and aliasing is not an issue for optimization of array references, allowing compilers to generate faster code than in the case of pointers.) Another important

supplement to Fortran 95 was the ISO technical report *TR-15580: Floating-point exception handling*, informally known as the *IEEE TR*. This specification defined support for IEEE floating-point arithmetic and floating point exception handling.

FORTRAN 2003

Fortran 2003, officially published as ISO/IEC 1539-1:2004, is a major revision introducing many new features. A comprehensive summary of the new features of Fortran 2003 is available at the Fortran Working Group (ISO/IEC JTC1/SC22/WG5) official Web site. From that article, the major enhancements for this revision include the following :-

- Derived type enhancements: parameterized derived types, improved control of accessibility, improved structure constructors, and finalizers
- Object-oriented programming support: type extension and inheritance, polymorphism, dynamic type allocation, and type-bound procedures, providing complete support for abstract data types
- Data manipulation enhancements: allocatable components (incorporating TR 15581), deferred type parameters, VOLATILE attribute, explicit type specification in array constructors and allocate statements, pointer enhancements, extended initialization expressions, and enhanced intrinsic procedures
- Input/output enhancements: asynchronous transfer, stream access, user specified transfer operations for derived types, user specified control of rounding during format conversions, named constants for preconnected units, the FLUSH statement, regularization of keywords, and access to error messages
- Procedure pointers
- Support for IEEE floating-point arithmetic and floating point exception handling (incorporating TR 15580)
- Interoperability with the C programming language
- Support for international usage: access to ISO 10646 4-byte characters and choice of decimal or comma in numeric formatted input/output
- Enhanced integration with the host operating system: access to command line arguments, environment variables, and processor error messages

An important supplement to Fortran 2003 was the ISO technical report *TR-19767: Enhanced module facilities in Fortran.* This report provided *submodules*, which make Fortran modules more similar to Modula-2 modules. They are similar to Ada private child subunits. This allows the specification and implementation of a module to be expressed in separate program units, which improves packaging of large libraries, allows preservation of trade secrets while publishing definitive interfaces, and prevents compilation cascades.

FORTRAN 2008

The most recent standard, ISO/IEC 1539-1:2010, informally known as Fortran 2008, was approved in September 2010. As with Fortran 95, this is a minor upgrade, incorporating clarifications and corrections to Fortran 2003, as well as introducing a select few new capabilities. The new capabilities include the following :-

- Submodules – additional structuring facilities for modules; supersedes ISO/IEC TR 19767:2005
- Coarray Fortran – a parallel execution model
- The DO CONCURRENT construct – for loop iterations with no interdependencies
- The CONTIGUOUS attribute – to specify storage layout restrictions
- The BLOCK construct – can contain declarations of objects with construct scope
- Recursive allocatable components – as an alternative to recursive pointers in derived types

The Final Draft international Standard (FDIS) is available as document N1830. An important supplement to Fortran 2008 is the ISO Technical Specification (TS) 29113 on *Further Interoperability of Fortran with C*, which has been submitted to ISO in May 2012 for approval. The specification adds support for accessing the array descriptor from C and allows ignoring the type and rank of arguments.

FORTRAN 2015

The next revision of the language (Fortran 2015) is intended to be a minor revision and is planned for release in mid-2018. It is currently planned to include further interoperability between Fortran and C, additional parallel features, and "the removal of simple deficiencies in and discrepancies between existing facilities."

FORTRAN AND SUPERCOMPUTERS

Although a 1968 journal article by the authors of BASIC already described Fortran as "old-

fashioned", since Fortran has been in use for many decades, there is a vast body of Fortran software in daily use throughout the scientific and engineering communities. Jay Pasachoff wrote in 1984 that "physics and astronomy students simply have to learn Fortran. So much exists in Fortran that it seems unlikely that scientists will change to Pascal, Modula-2, or whatever." In 1993, Cecil E. Leith called Fortran the "mother tongue of scientific computing" adding that its replacement by any other possible language "may remain a forlorn hope." It is the primary language for some of the most intensive supercomputing tasks, such as astronomy, weather and climate modeling, numerical linear algebra (LAPACK), numerical libraries (IMSL and NAG), structural engineering, hydrological modeling, optimization, satellite simulation and data analysis, computational fluid dynamics, computational chemistry, computational economics and computational physics. Many of the floating-point benchmarks to gauge the performance of new computer processors – such as CFP2006, the floating-point component of the SPEC CPU2006 benchmarks – are written in Fortran.

COBOL

COBOL (/ˈkoʊbɒl/, an acronym for *common business-oriented language*) is a compiled English-like computer programming language designed for business use. It is imperative, procedural and, since 2002, object-oriented. COBOL is primarily used in business, finance, and administrative systems for companies and governments. COBOL is still widely used in legacy applications deployed on mainframe computers, such as large-scale batch and transaction processing jobs. But due to its declining popularity and the retirement of experienced COBOL programmers, programs are being migrated to new platforms, rewritten in modern languages or replaced with software packages. Most programming in COBOL is now purely to maintain existing applications. COBOL was designed in 1959 by CODASYL and was partly based on previous programming language design work by Grace Hopper, commonly referred to as "the (grand)mother of COBOL". It was created as part of a US Department of Defense effort to create a portable programming language for data processing. Intended as a stopgap, the Department of Defense promptly forced computer manufacturers to provide it, resulting in its widespread adoption. It was standardized in 1968 and has since been revised four times. Expansions include support for structured and object-oriented programming. The current standard is *ISO/IEC 1989:2014*. COBOL has an English-like syntax, which was designed to be self-documenting and highly readable. However, it is verbose and uses over 300 reserved words. In contrast with modern, succinct syntax like y = x;, COBOL has a more English-like syntax (in this case, MOVE x TO y). COBOL code is split into four divisions (identification, environment, data and procedure) containing a rigid hierarchy of sections, paragraphs and sentences. Lacking a large standard library, the standard specifies 43 statements, 87 functions and just one class.

III. BACKGROUND

In the late 1950s, computer users and manufacturers were becoming concerned about the rising cost of programming. A 1959 survey had found that in any data processing installation, the programming cost US$800,000 on average and that translating programs to run on new hardware would cost $600,000. At a time when new programming languages were proliferating at an ever-increasing rate, the same survey suggested that if a common business-oriented language were used, conversion would be far cheaper and faster. Grace Hopper, the inventor of FLOW-MATIC, a predecessor to COBOL In April 1959, Mary K. Hawes called a meeting of representatives from academia, computer users, and manufacturers at the University of Pennsylvania to organize a formal meeting on common business languages. Representatives included Grace Hopper, inventor of the English-like data processing language FLOW-MATIC, Jean Sammet and Saul Gorn. The group asked the Department of Defense (DoD) to sponsor an effort to create a common business language. The delegation impressed Charles A. Phillips, director of the Data System Research Staff at the DoD, who thought that they "thoroughly understood" the DoD's problems. The DoD operated 225 computers, had a further 175 on order and had spent over $200 million on implementing programs to run on them. Portable programs would save time, reduce costs and ease modernization.

COBOL 60

On May 28 and 29 of 1959 (exactly one year after the Zürich ALGOL 58 meeting), a meeting was held at the Pentagon to discuss the creation of a common programming language for business. It was attended by 41 people and was chaired by Phillips. The Department of Defense was concerned about whether it could run the same data processing programs on different computers. FORTRAN, the only mainstream language at the time, lacked the features needed to write such programs. Representatives enthusiastically described a language that could work in a wide variety of environments, from banking and insurance to utilities and inventory control. They agreed unanimously that more people should be able to program and that the new language should not be restricted by the limitations of contemporary technology. A majority agreed that the language

should make maximal use of English, be capable of change, be machine-independent and be easy to use, even at the expense of power. The meeting resulted in the creation of a steering committee and short-, intermediate- and long-range committees. The short-range committee was given to September (three months) to produce specifications for an interim language, which would then be improved upon by the other committees. Their official mission, however, was to identify the strengths and weaknesses of existing programming languages and did not explicitly direct them to create a new language. The deadline was met with disbelief by the short-range committee. One member, Betty Holberton, described the three-month deadline as "gross optimism" and doubted that the language really would be a stopgap.

COBOL-61 to COBOL-65

It is rather unlikely that Cobol will be around by the end of the decade. Many logical flaws were found in *COBOL 60*, leading GE's Charles Katz to warn that it could not be interpreted unambiguously. A reluctant short-term committee enacted a total cleanup and, by March 1963, it was reported that COBOL's syntax was as definable as ALGOL's, although semantic ambiguities remained. Early COBOL compilers were primitive and slow. A 1962 US Navy evaluation found compilation speeds of 3–11 statements per minute. By mid-1964, they had increased to 11–1000 statements per minute. It was observed that increasing memory would drastically increase speed and that compilation costs varied wildly: costs per statement were between $0.23 and $18.91. In late 1962, IBM announced that COBOL would be their primary development language and that development of COMTRAN would cease. The COBOL specification was revised three times in the five years after its publication. COBOL-60 was replaced in 1961 by COBOL-61. This was then replaced by the COBOL-61 Extended specifications in 1963, which introduced the sort and report writer facilities. The added facilities corrected flaws identified by Honeywell in late 1959 in a letter to the short-range committee. COBOL Edition 1965 brought further clarifications to the specifications and introduced facilities for handling mass storage files and tables.

COBOL-68

Efforts began to standardize COBOL to overcome incompatibilities between versions. In late 1962, both ISO and the United States of America Standards Institute (now ANSI) formed groups to create standards. ANSI produced *USA Standard COBOL X3.23* in August 1968, which became the cornerstone for later versions. This version was known as American National Standard (ANS) COBOL and was adopted by ISO in 1972.

COBOL-74

By 1970, COBOL had become the most widely used programming language in the world. Independently of the ANSI committee, the CODASYL Programming Language Committee was working on improving the language. They described new versions in 1968, 1969, 1970 and 1973, including changes such as new inter-program communication, debugging and file merging facilities as well as improved string-handling and library inclusion features. Although CODASYL was independent of the ANSI committee, the *CODASYL Journal of Development* was used by ANSI to identify features that were popular enough to warrant implementing. The Programming Language Committee also liaised with ECMA and the Japanese COBOL Standard committee. The Programming Language Committee was not well-known, however. The vice-president, William Rinehuls, complained that two-thirds of the COBOL community did not know of the committee's existence. It was also poor, lacking the funds to make public documents, such as minutes of meetings and change proposals, freely available. In 1974, ANSI published a revised version of (ANS) COBOL, containing new features such as file organizations, the DELETE statement and the segmentation module. Deleted features included the NOTE statement, the EXAMINE statement (which was replaced by INSPECT) and the implementer-defined random access module (which was superseded by the new sequential and relative I/O modules). These made up 44 changes, which rendered existing statements incompatible with the new standard. The report writer was slated to be removed from COBOL, but was reinstated before the standard was published. ISO later adopted the updated standard in 1978.

COBOL-85

In June 1978, work began on revising COBOL-74. The proposed standard (commonly called COBOL-80) differed significantly from the previous one, causing concerns about incompatibility and conversion costs. In January 1981, Joseph T. Brophy, Senior Vice-President of Travelers Insurance, threatened to sue the standard committee because it was not upwards compatible with COBOL-74. Mr. Brophy described previous conversions of their 40-million-line code base as "non-productive" and a "complete waste of our programmer resources". Later that year, the Data Processing Management Association (DPMA) said it was "strongly opposed" to the new standard, citing "prohibitive" conversion costs and enhancements that were "forced on the user". During the first public review period, the committee received 2,200 responses, of which 1,700 were negative form letters. Other responses

were detailed analyses of the effect COBOL-80 would have on their systems; conversion costs were predicted to be at least 50 cents per line of code. Fewer than a dozen of the responses were in favor of the proposed standard.

COBOL 2002 AND OBJECT-ORIENTED COBOL

In 1997, Gartner Group estimated that there were a total of 200 billion lines of COBOL in existence, which ran 80% of all business programs. In the early 1990s, work began on adding object-orientation in the next full revision of COBOL. Object-oriented features were taken from C++ and Smalltalk. The initial estimate was to have this revision completed by 1997, and an ISO Committee Draft (CD) was available by 1997. Some vendors (including Micro Focus, Fujitsu, and IBM) introduced object-oriented syntax based on drafts of the full revision. The final approved ISO standard was approved and published in late 2002. Fujitsu/GTSoftware, Micro Focus and RainCode introduced object-oriented COBOL compilers targeting the .NET Framework. There were many other new features, many of which had been in the *CODASYL COBOL Journal of Development* since 1978 and had missed the opportunity to be included in COBOL-85.

COBOL 2014

Between 2003 and 2009, three technical reports were produced describing object finalization, XML processing and collection classes for COBOL. COBOL 2002 suffered from poor support: no compilers completely supported the standard. Micro Focus found that it was due to a lack of user demand for the new features and due to the abolition of the NIST test suite, which had been used to test compiler conformance. The standardization process was also found to be slow and under-resourced.

PASCAL

Pascal is an imperative and procedural programming language, which Niklaus Wirth designed in 1968–69 and published in 1970, as a small, 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.

IV. 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. 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. 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. The 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). Apollo Computer used Pascal as the systems programming language for its operating systems beginning in 1980. Object Pascal (Embarcadero Delphi) is still used for developing Windows applications but also has the ability to cross compile the same code to Mac, iOS and Android. Another cross-platform version called Free Pascal, with the Lazarus IDE, is popular with Linux users since it also offers write once, compile anywhere development. CodeTyphon is a Lazarus distribution with more preinstalled packages and cross compilers.

V. BRIEF DESCRIPTION

Wirth's intention was to create an efficient language (regarding both compilation speed and generated code) based on structured programming, a recently popularized concept that he promoted in his book *Algorithms + Data Structures = Programs*. 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 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 subroutines (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 heading, which has one of these three keywords).

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 implemented in a C-like language (Scallop by Max Engeli) and then translated by hand (by R. Schild) to Pascal itself for boot-strapping. It 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 Information Computer Systems (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.

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 version termed Pascal-P1 was coined after the fact for the many different sources for Pascal-P that existed. The compiler was redesigned to enhance portability, and issued as Pascal-P2. This code was later enhanced to become Pascal-P3, with an intermediate code backward compatible with Pascal-P2, and Pascal-P4, which was not backward compatible. 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 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. The UCSD p-System was one of three operating systems available at the launch of the original IBM Personal Computer. UCSD Pascal used an intermediate code based on byte values, and thus was one of the earliest "byte code compilers". Pascal-P1 through Pascal-P4 was not, but rather based on the CDC 6600 60 bit word length. 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. Into the 1990s, Pascal was still running on VAX terminals at George Mason University to teach computer programming.

OBJECT PASCAL AND TURBO PASCAL

Apple Computer created its own Lisa Pascal for the Lisa Workshop in 1982, and ported the 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 and renamed *Turbo Pascal*. Turbo Pascal became hugely popular, thanks to: an aggressive pricing strategy; having one of the first full-screen Integrated development environments - and very fast turnaround-time (just seconds to compile, link, and run.) It was written and highly optimized entirely in assembly language, making it smaller and faster than much of the competition.

C-LANGUAGE

C (/ˈsiː/, as in the letter *c*) is a general-purpose, imperative computer programming language, supporting structured programming, lexical variable scope and recursion, while a static type system prevents many unintended operations. By design, C provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, including operating systems, as well as various application software for computers ranging from supercomputers to embedded systems. C was originally developed by Dennis Ritchie between 1969 and 1973 at Bell Labs, and used to re-implement the Unix operating system. It has since become one of the most widely used programming languages of all time, with C compilers from

various vendors available for the majority of existing computer architectures and operating systems. C has been standardized by the American National Standards Institute (ANSI) since 1989 (see ANSI C) and subsequently by the International Organization for Standardization (ISO).

VI. DESIGN

C is an imperative procedural language. It was designed to be compiled using a relatively straightforward compiler, to provide low-level access to memory, to provide language constructs that map efficiently to machine instructions, and to require minimal run-time support. Therefore, C was useful for many applications that had formerly been coded in assembly language, for example in system programming. Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant and portably written C program can be compiled for a very wide variety of computer platforms and operating systems with few changes to its source code. The language has become available on a very wide range of platforms, from embedded microcontrollers to supercomputers.

VII. OVERVIEW

Like most imperative languages in the ALGOL tradition, C has facilities for structured programming and allows lexical variable scope and recursion, while a static type system prevents many unintended operations. In C, all executable code is contained within subroutines, which are called "functions" (although not in the strict sense of functional programming). Function parameters are always passed by value. Pass-by-reference is simulated in C by explicitly passing pointer values. C program source text is free-format, using the semicolon as a statement terminator and curly braces for grouping blocks of statements. The C language also exhibits the following characteristics:

- There is a small, fixed number of keywords, including a full set of flow of control primitives: for, if/else, while, switch, and do/while. User-defined names are not distinguished from keywords by any kind of sigil.
- There are a large number of arithmetical and logical operators, such as +, +=, ++, &, ~, etc.
- More than one assignment may be performed in a single statement.
- Function return values can be ignored when not needed.
- Typing is static, but weakly enforced: all data has a type, but implicit conversions may be performed.
- Declaration syntax mimics usage context. C has no "define" keyword; instead, a statement beginning with the name of a type is taken as a declaration. There is no "function" keyword; instead, a function is indicated by the parentheses of an argument list.
- User-defined (typedef) and compound types are possible.
    - Heterogeneous aggregate data types (struct) allow related data elements to be accessed and assigned as a unit.
    - Array indexing is a secondary notation, defined in terms of pointer arithmetic. Unlike structs, arrays are not first-class objects; they cannot be assigned or compared using single built-in operators. There is no "array" keyword, in use or definition; instead, square brackets indicate arrays syntactically, for example month[11].
    - Enumerated types are possible with the enum keyword. They are not tagged, and are freely interconvertible with integers.
    - Strings are not a separate data type, but are conventionally implemented as null-terminated arrays of characters.
- Low-level access to computer memory is possible by converting machine addresses to typed pointers.
- Procedures (subroutines not returning values) are a special case of function, with an untyped return type void.
- Functions may not be defined within the lexical scope of other functions.
- Function and data pointers permit *ad hoc* run-time polymorphism.
- A preprocessor performs macro definition, source code file inclusion, and conditional compilation.
- There is a basic form of modularity: files can be compiled separately and linked together, with control over which functions and data objects are visible to other files via static and extern attributes.
- Complex functionality such as I/O, string manipulation, and mathematical functions are consistently delegated to library routines.

While C does not include some features found in some other languages, such as object orientation or garbage collection, such features can be implemented or emulated in C, often by way of external libraries (e.g., the Boehm garbage collector or the GLib Object System).

RELATIONS TO OTHER LANGUAGES

Many later languages have borrowed directly or indirectly from C, including C++, D, Go, Rust, Java, JavaScript, Limbo, LPC, C#, Objective-C, Perl, PHP, Python, Swift, Verilog (hardware description language), and Unix's C shell. These languages have drawn many of their control structures and other basic features from C. Most of them (with Python being the most dramatic exception) are also very syntactically similar to C in general, and they tend to combine the recognizable expression and statement syntax of C with underlying type systems, data models, and semantics that can be radically different. Early developments. Ken Thompson (left) with Dennis Ritchie (right, the inventor of the C programming language). The origin of C is closely tied to the development of the Unix operating system, originally implemented in assembly language on a PDP-7 by Ritchie and Thompson, incorporating several ideas from colleagues. Eventually, they decided to port the operating system to a PDP-11. The original PDP-11 version of Unix was developed in assembly language. The developers were considering rewriting the system using the B language, Thompson's simplified version of BCPL. However B's inability to take advantage of some of the PDP-11's features, notably byte addressability, led to C. The name of C was chosen simply as the next after B. The development of C started in 1972 on the PDP-11 Unix system and first appeared in Version 2 Unix. The language was not initially designed with portability in mind, but soon ran on different platforms as well: a compiler for the Honeywell 6000 was written within the first year of C's history, while an IBM System/370 port followed soon. Also in 1972, a large part of Unix was rewritten in C. By 1973, with the addition of struct types, the C language had become powerful enough that most of the Unix's kernel was now in C.

ANSI C and ISO C

During the late 1970s and 1980s, versions of C were implemented for a wide variety of mainframe computers, minicomputers, and microcomputers, including the IBM PC, as its popularity began to increase significantly. In 1983, the American National Standards Institute (ANSI) formed a committee, X3J11, to establish a standard specification of C. X3J11 based the C standard on the Unix implementation; however, the non-portable portion of the Unix C library was handed off to the IEEE working group 1003 to become the basis for the 1988 POSIX standard. In 1989, the C standard was ratified as ANSI X3.159-1989 "Programming Language C". This version of the language is often referred to as ANSI C, Standard C, or sometimes C89. In 1990, the ANSI C standard (with formatting changes) was adopted by the International Organization for Standardization (ISO) as ISO/IEC 9899:1990, which is sometimes called C90. Therefore, the terms "C89" and "C90" refer to the same programming language. ANSI, like other national standards bodies, no longer develops the C standard independently, but defers to the international C standard, maintained by the working group ISO/IEC JTC1/SC22/WG14. National adoption of an update to the international standard typically occurs within a year of ISO publication.

C99 LANGUAGE

C99 introduced several new features, including inline functions, several new data types (including long long int and a complex type to represent complex numbers), variable-length arrays and flexible array members, improved support for IEEE 754 floating point, support for variadic macros (macros of variable arity), and support for one-line comments beginning with //, as in BCPL or C++. Many of these had already been implemented as extensions in several C compilers. C99 is for the most part backward compatible with C90, but is stricter in some ways; in particular, a declaration that lacks a type specifier no longer has int implicitly assumed. A standard macro __STDC_VERSION__ is defined with value 199901L to indicate that C99 support is available. GCC, Solaris Studio, and other C compilers now support many or all of the new features of C99. The C compiler in Microsoft Visual C++, however, implements the C89 standard and those parts of C99 that are required for compatibility with C++11.

C11 LANGUAGE

In 2007, work began on another revision of the C standard, informally called "C1X" until its official publication on 2011-12-08. The C standards committee adopted guidelines to limit the adoption of new features that had not been tested by existing implementations. The C11 standard adds numerous new features to C and the library, including type generic macros, anonymous structures, improved Unicode support, atomic operations, multi-threading, and bounds-checked functions. It also makes some portions of the existing C99 library optional, and improves compatibility with C++. The standard macro __STDC_VERSION__ is defined as 201112L to indicate that C11 support is available.

EMBEDDED C

Historically, embedded C programming requires nonstandard extensions to the C language in order to support exotic features such as fixed-point arithmetic, multiple distinct memory banks, and basic I/O operations. In 2008, the C Standards Committee published a technical report extending

the C language to address these issues by providing a common standard for all implementations to adhere to. It includes a number of features not available in normal C, such as fixed-point arithmetic, named address spaces, and basic I/O hardware addressing.

C++ LANGUAGE

C++ (pronounced *cee plus plus*, /ˈsiː plʌs plʌs/) is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications, including desktop applications, servers (e.g. e-commerce, web search or SQL servers), and performance-critical applications (e.g. telephone switches or space probes). C++ is a compiled language, with implementations of it available on many platforms and provided by various organizations, including the Free Software Foundation (FSF's GCC), LLVM, Microsoft, Intel and IBM. C++ is standardized by the International Organization for Standardization (ISO), with the latest standard version ratified and published by ISO in December 2014 as *ISO/IEC 14882:2014* (informally known as C++14). The C++ programming language was initially standardized in 1998 as *ISO/IEC 14882:1998*, which was then amended by the C++03, *ISO/IEC 14882:2003*, standard. The current C++14 standard supersedes these and C++11, with new features and an enlarged standard library. Before the initial standardization in 1998, C++ was developed by Bjarne Stroustrup at Bell Labs since 1979, as an extension of the C language as he wanted an efficient and flexible language similar to C, which also provided high-level features for program organization.

VIII. HISTORY

In 1979, Bjarne Stroustrup, a Danish computer scientist, began work on "C with Classes", the predecessor to C++. The motivation for creating a new language originated from Stroustrup's experience in programming for his Ph.D. thesis. Stroustrup found that Simula had features that were very helpful for large software development, but the language was too slow for practical use, while BCPL was fast but too low-level to be suitable for large software development. When Stroustrup started working in AT&T Bell Labs, he had the problem of analyzing the UNIX kernel with respect to distributed computing. Remembering his Ph.D. experience, Stroustrup set out to enhance the C language with Simula-like features. C was chosen because it was general-purpose, fast, portable and widely used. As well as C and Simula's influences, other languages also influenced C++, including ALGOL 68, Ada, CLU and ML. Initially, Stroustrup's "C with Classes" added features to the C compiler, Cpre, including classes, derived classes, strong typing, inlining and default arguments. In 1983, "C with Classes" was renamed to "C++" ("++" being the increment operator in C), adding new features that included virtual functions, function name and operator overloading, references, constants, type-safe free-store memory allocation (new/delete), improved type checking, and BCPL style single-line comments with two forward slashes (//). Furthermore, it included the development of a standalone compiler for C++, Cfront. In 1985, the first edition of *The C++ Programming Language* was released, which became the definitive reference for the language, as there was not yet an official standard. The first commercial implementation of C++ was released in October of the same year. In 1989, C++ 2.0 was released, followed by the updated second edition of *The C++ Programming Language* in 1991. New features in 2.0 included multiple inheritance, abstract classes, static member functions, const member functions, and protected members. In 1990, *The Annotated C++ Reference Manual* was published. This work became the basis for the future standard. Later feature additions included templates, exceptions, namespaces, new casts, and a boolean type. After the 2.0 update, C++ evolved relatively slowly until, in 2011, the C++11 standard was released, adding numerous new features, enlarging the standard library further, and providing more facilities to C++ programmers. After a minor C++14 update released in December 2014, various new additions are planned for 2017 and 2020.

ETYMOLOGY

According to Stroustrup: "the name signifies the evolutionary nature of the changes from C". This name is credited to Rick Mascitti (mid-1983) and was first used in December 1983. When Mascitti was questioned informally in 1992 about the naming, he indicated that it was given in a tongue-in-cheek spirit. The name comes from C's "++" operator (which increments the value of a variable) and a common naming convention of using "+" to indicate an enhanced computer program. During C++'s development period, the language had been referred to as "new C" and "C with Classes" before acquiring its final name.

PHILOSOPHY

Throughout C++'s life, its development and evolution has been informally governed by a set of rules that its evolution should follow:

- It must be driven by actual problems and its features should be useful immediately in real world programs.
- Every feature should be implementable (with a reasonably obvious way to do so).
- Programmers should be free to pick their own programming style, and that style should be fully supported by C++.
- Allowing a useful feature is more important than preventing every possible misuse of C++.
- It should provide facilities for organising programs into well-defined separate parts, and provide facilities for combining separately developed parts.
- No implicit violations of the type system (but allow explicit violations; that is, those explicitly requested by the programmer).
- User-created types need to have the same support and performance as built-in types.
- Unused features should not negatively impact created executables (e.g. in lower performance).
- There should be no language beneath C++ (except assembly language).
- C++ should work alongside other existing programming languages, rather than fostering its own separate and incompatible programming environment.
- If the programmer's intent is unknown, allow the programmer to specify it by providing manual control.

STANDARDIZATION

Year	C++ Standard	Informal name
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2011	ISO/IEC 14882:2011	C++11
2014	ISO/IEC 14882:2014	C++14
2017	to be determined	C++17
2020	to be determined	C++20

C++ is standardized by an ISO working group known as JTC1/SC22/WG21. So far, it has published four revisions of the C++ standard and is currently working on the next revision, C++17. In 1998, the ISO working group standardized C++ for the first time as *ISO/IEC 14882:1998*, which is informally known as *C++98*. In 2003, it published a new version of the C++ standard called *ISO/IEC 14882:2003*, which fixed problems identified in C++98. The next major revision of the standard was informally referred to as "C++0x", but it was not released until 2011. C++11 (14882:2011) included many additions to both the core language and the standard library. In 2014, C++14 (also known as C++1y) was released as a small extension to C++11, featuring mainly bug fixes and small improvements. The Draft International Standard ballot procedures completed in mid-August 2014. After C++14, a major revision, informally known as C++17 or C++1z, is planned for 2017, which is almost feature-complete. As part of the standardization process, ISO also publishes technical reports and specifications:

- ISO/IEC TR 18015:2006 on the use of C++ in embedded systems and on performance implications of C++ language and library features,
- ISO/IEC TR 19768:2007 (also known as the C++ Technical Report 1) on library extensions mostly integrated into C++11,
- ISO/IEC TR 29124:2010 on special mathematical functions,
- ISO/IEC TR 24733:2011 on decimal floating point arithmetic,
- ISO/IEC TS 18822:2015 on the standard filesystem library,
- ISO/IEC TS 19570:2015 on parallel versions of the standard library algorithms,
- ISO/IEC TS 19841:2015 on software transactional memory,
- ISO/IEC TS 19568:2015 on a new set of library extensions, some of which are already integrated into C++17,
- ISO/IEC TS 19217:2015 on the C++ Concepts

More technical specifications are in development and pending approval, including concurrency library extensions, a networking standard library, ranges, and modules.

JAVA

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. As of 2016,

Java is one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers. Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them. The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licences. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java (bytecode compiler), GNU Classpath (standard libraries), and IcedTea-Web (browser plugin for applets).

IX. HISTORY

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. Java was originally designed for interactive television, but it was too advanced for the digital cable television industry at the time. The language was initially called *Oak* after an oak tree that stood outside Gosling's office. Later the project went by the name *Green* and was finally renamed *Java*, from Java coffee. Gosling designed Java with a C/C++-style syntax that system and application programmers would find familiar. Sun Microsystems released the first public implementation as Java 1.0 in 1995. It promised "Write Once, Run Anywhere" (WORA), providing no-cost run-times on popular platforms. Fairly secure and featuring configurable security, it allowed network- and file-access restrictions. Major web browsers soon incorporated the ability to run *Java applets* within web pages, and Java quickly became popular, while mostly outside of browsers, that wasn't the original plan. In January 2016, Oracle announced that Java runtime environments based on JDK 9 will discontinue the browser plugin. The Java 1.0 compiler was re-written in Java by Arthur van Hoff to comply strictly with the Java 1.0 language specification. With the advent of *Java 2* (released initially as J2SE 1.2 in December 1998 – 1999), new versions had multiple configurations built for different types of platforms. *J2EE* included technologies and APIs for enterprise applications typically run in server environments, while *J2ME* featured APIs optimized for mobile applications. The desktop version was renamed *J2SE*. In 2006, for marketing purposes, Sun renamed new *J2* versions as *Java EE*, *Java ME*, and *Java SE*, respectively. In 1997, Sun Microsystems approached the ISO/IEC JTC 1 standards body and later the Ecma International to formalize Java, but it soon withdrew from the process. Java remains a *de facto* standard, controlled through the Java Community Process. At one time, Sun made most of its Java implementations available without charge, despite their proprietary software status. Sun generated revenue from Java through the selling of licenses for specialized products such as the Java Enterprise System. On November 13, 2006, Sun released much of its Java virtual machine (JVM) as free and open-source software, (FOSS), under the terms of the GNU General Public License (GPL). On May 8, 2007, Sun finished the process, making all of its JVM's core code available under free software/open-source distribution terms, aside from a small portion of code to which Sun did not hold the copyright.

JAVA PLATFORM

One design goal of Java is portability, which means that programs written for the Java platform must run similarly on any combination of hardware and operating system with adequate runtime support. This is achieved by compiling the Java language code to an intermediate representation called Java bytecode, instead of directly to architecture-specific machine code. Java bytecode instructions are analogous to machine code, but they are intended to be executed by a virtual machine (VM) written specifically for the host hardware. End users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a web browser for Java applets. Standard libraries provide a generic way to access host-specific features such as graphics, threading, and networking. The use of universal bytecode makes porting simple. However, the overhead of interpreting bytecode into machine instructions makes interpreted programs almost always run more slowly than native executables. However, just-in-time (JIT) compilers that compile bytecodes to machine code during runtime were introduced from an early stage. Java itself is platform-independent, and is adapted to the particular platform it is to run on by a Java virtual machine for it, which translates the Java bytecode into the platform's machine language.

IMPLEMENTATIONS

Oracle Corporation is the current owner of the official implementation of the Java SE platform, following their acquisition of Sun Microsystems on January 27, 2010. This implementation is based on the original implementation of Java by Sun. The Oracle implementation is available for Microsoft Windows (still works for XP, while only later versions currently officially supported), Mac OS X, Linux and Solaris. Because Java lacks any formal

standardization recognized by Ecma International, ISO/IEC, ANSI, or other third-party standards organization, the Oracle implementation is the de facto standard. The Oracle implementation is packaged into two different distributions: The Java Runtime Environment (JRE) which contains the parts of the Java SE platform required to run Java programs and is intended for end users, and the Java Development Kit (JDK), which is intended for software developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger. OpenJDK is another notable Java SE implementation that is licensed under the GNU GPL. The implementation started when Sun began releasing the Java source code under the GPL. As of Java SE 7, OpenJDK is the official Java reference implementation. The goal of Java is to make all implementations of Java compatible. Historically, Sun's trademark license for usage of the Java brand insists that all implementations be "compatible". This resulted in a legal dispute with Microsoft after Sun claimed that the Microsoft implementation did not support RMI or JNI and had added platform-specific features of their own. Sun sued in 1997, and in 2001 won a settlement of US$20 million, as well as a court order enforcing the terms of the license from Sun. As a result, Microsoft no longer ships Java with Windows.

PERFORMANCE

Programs written in Java have a reputation for being slower and requiring more memory than those written in C++. However, Java programs' execution speed improved significantly with the introduction of just-in-time compilation in 1997/1998 for Java 1.1, the addition of language features supporting better code analysis (such as inner classes, the StringBuilder class, optional assertions, etc.), and optimizations in the Java virtual machine, such as HotSpot becoming the default for Sun's JVM in 2000. With Java 1.5, the performance was improved with the addition of the java.util.concurrent package, including Lock free implementations of the ConcurrentMaps and other multi-core collections, and it was improved further Java 1.6. Some platforms offer direct hardware support for Java; there are microcontrollers that can run Java in hardware instead of a software Java virtual machine, and ARM based processors can have hardware support for executing Java bytecode through their Jazelle option (while its support is mostly dropped in current implementations of ARM).

AUTOMATIC MEMORY MANAGEMENT

Java uses an automatic garbage collector to manage memory in the object lifecycle. The programmer determines when objects are created, and the Java runtime is responsible for recovering the memory once objects are no longer in use. Once no references to an object remain, the unreachable memory becomes eligible to be freed automatically by the garbage collector. Something similar to a memory leak may still occur if a programmer's code holds a reference to an object that is no longer needed, typically when objects that are no longer needed are stored in containers that are still in use. If methods for a nonexistent object are called, a "null pointer exception" is thrown. One of the ideas behind Java's automatic memory management model is that programmers can be spared the burden of having to perform manual memory management. In some languages, memory for the creation of objects is implicitly allocated on the stack, or explicitly allocated and deallocated from the heap. In the latter case the responsibility of managing memory resides with the programmer. If the program does not deallocate an object, a memory leak occurs. If the program attempts to access or deallocate memory that has already been deallocated, the result is undefined and difficult to predict, and the program is likely to become unstable and/or crash. This can be partially remedied by the use of smart pointers, but these add overhead and complexity. Note that garbage collection does not prevent "logical" memory leaks, *i.e.*, those where the memory is still referenced but never used.

SYNTAX

The syntax of Java is largely influenced by C++. Unlike C++, which combines the syntax for structured, generic, and object-oriented programming, Java was built almost exclusively as an object-oriented language. All code is written inside classes, and every data item is an object, with the exception of the primitive data types, *i.e.* integers, floating-point numbers, boolean values, and characters, which are not objects for performance reasons. Java reuses some popular aspects of C++ (such as printf() method). Unlike C++, Java does not support operator overloading or multiple inheritance for *classes*, though multiple inheritance is supported for interfaces. This simplifies the language and aids in preventing potential errors and anti-pattern design. Java uses comments similar to those of C++. There are three different styles of comments: a single line style marked with two slashes (//), a multiple line style opened with /* and closed with */, and the Javadoc commenting style opened with /** and closed with */. The Javadoc style of commenting allows the user to run the Javadoc executable to create documentation for the program.

MySQL

MySQL (officially pronounced as /maɪ ˌɛskjuːˈɛl/ "My S-Q-L",) is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder

Michael Widenius' daughter, and "SQL", the abbreviation for Structured Query Language. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation. For proprietary use, several paid editions are available, and offer additional functionality. MySQL is a central component of the LAMP open-source web application software stack (and other "AMP" stacks). LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python". Applications that use the MySQL database include: TYPO3, MODx, Joomla, WordPress, phpBB, MyBB, and Drupal. MySQL is also used in many high-profile, large-scale websites, including Google (though not for searches), Facebook, Twitter, Flickr, and YouTube.

X. OVERVIEW

MySQL is written in C and C++. Its SQL parser is written in yacc, but it uses a home-brewed lexical analyzer. MySQL works on many system platforms, including AIX, BSDi, FreeBSD, HP-UX, eComStation, i5/OS, IRIX, Linux, macOS, Microsoft Windows, NetBSD, Novell NetWare, OpenBSD, OpenSolaris, OS/2 Warp, QNX, Oracle Solaris, Symbian, SunOS, SCO OpenServer, SCO UnixWare, Sanos and Tru64. A port of MySQL to OpenVMS also exists. The MySQL server software itself and the client libraries use dual-licensing distribution. They are offered under GPL version 2, beginning from 28 June 2000 (which in 2009 has been extended with a FLOSS License Exception) or to use a proprietary license. Support can be obtained from the official manual. Free support additionally is available in different IRC channels and forums. Oracle offers paid support via its MySQL Enterprise products. They differ in the scope of services and in price. Additionally, a number of third party organisations exist to provide support and services, including MariaDB and Percona. MySQL has received positive reviews, and reviewers noticed it "performs extremely well in the average case" and that the "developer interfaces are there, and the documentation (not to mention feedback in the real world via Web sites and the like) is very, very good". It has also been tested to be a "fast, stable and true multi-user, multi-threaded sql database server".

XI. HISTORY

MySQL was created by a Swedish company, MySQL AB, founded by David Axmark, Allan Larsson and Michael "Monty" Widenius. Original development of MySQL by Widenius and Axmark began in 1994. The first version of MySQL appeared on 23 May 1995. It was initially created for personal usage from mSQL based on the low-level language ISAM, which the creators considered too slow and inflexible. They created a new SQL interface, while keeping the same API as mSQL. By keeping the API consistent with the mSQL system, many developers were able to use MySQL instead of the (proprietarily licensed) mSQL antecedent.

MILESTONES

Additional milestones in MySQL development included:

- First internal release on 23 May 1995
- Version 3.19: End of 1996, from www.tcx.se
- Version 3.20: January 1997
- Windows version was released on 8 January 1998 for Windows 95 and NT
- Version 3.21: production release 1998, from www.mysql.com
- Version 3.22: alpha, beta from 1998
- Version 3.23: beta from June 2000, production release 22 January 2001
- Version 4.0: beta from August 2002, production release March 2003 (unions)
- Version 4.01: beta from August 2003, Jyoti adopts MySQL for database tracking
- Version 4.1: beta from June 2004, production release October 2004 (R-trees and B-trees, subqueries, prepared statements)
- Version 5.0: beta from March 2005, production release October 2005 (cursors, stored procedures, triggers, views, XA transactions) . The developer of the Federated Storage Engine states that "The Federated Storage Engine is a proof-of-concept storage engine", but the main distributions of MySQL version 5.0 included it and turned it on by default. Documentation of some of the short-comings appears in "MySQL Federated Tables: The Missing Manual".
- Sun Microsystems acquired MySQL AB in 2008.
- Version 5.1: production release 27 November 2008 (event scheduler, partitioning, plugin API, row-based replication, server log tables) . Version 5.1 contained 20 known crashing and wrong result bugs in addition to the 35 present in version 5.0 *(almost all fixed as of release 5.1.51)*. MySQL 5.1 and 6.0-alpha showed poor performance when used for data warehousing – partly due to its inability to

utilize multiple CPU cores for processing a single query.

- Oracle acquired Sun Microsystems on 27 January 2010.
- The day Oracle announced the purchase of Sun, Michael "Monty" Widenius forked MySQL, launching MariaDB, and took a swath of MySQL developers with him.
- MySQL Server 5.5 was generally available (as of December 2010). Enhancements and features include:
  - The default storage engine is InnoDB, which supports transactions and referential integrity constraints.
  - Improved InnoDB I/O subsystem
  - Improved SMP support
  - Semisynchronous replication.
  - SIGNAL and RESIGNAL statement in compliance with the SQL standard.
  - Support for supplementary Unicode character sets utf16, utf32, and utf8mb4.
  - New options for user-defined partitioning.
- MySQL Server 6.0.11-alpha was announced on 22 May 2009 as the last release of the 6.0 line. Future MySQL Server development uses a New Release Model. Features developed for 6.0 are being incorporated into future releases.
- The general availability of MySQL 5.6 was announced in February 2013. New features included performance improvements to the query optimizer, higher transactional throughput in InnoDB, new NoSQL-style memcached APIs, improvements to partitioning for querying and managing very large tables, TIMESTAMP column type that correctly stores milliseconds, improvements to replication, and better performance monitoring by expanding the data available through the PERFORMANCE_SCHEMA. The InnoDB storage engine also included support for full-text search and improved group commit performance.
- The general availability of MySQL 5.7 was announced in October 2015.
- MySQL Server 8.0.0-dmr (Milestone Release) was announced 12 September 2016.

LEGAL DISPUTES AND ACQUISITIONS

On 15 June 2001, NuSphere sued MySQL AB, TcX DataKonsult AB and its original authors Michael ("Monty") Widenius and David Axmark in U.S District Court in Boston for "breach of contract, tortious interference with third party contracts and relationships and unfair competition". In 2002, MySQL AB sued Progress NuSphere for copyright and trademark infringement in United States district court. NuSphere had allegedly violated MySQL's copyright by linking MySQL's GPL'ed code with NuSphere Gemini table without being in compliance with the license. After a preliminary hearing before Judge Patti Saris on 27 February 2002, the parties entered settlement talks and eventually settled. After the hearing, FSF commented that "Judge Saris made clear that she sees the GNU GPL to be an enforceable and binding license." In October 2005, Oracle Corporation acquired Innobase OY, the Finnish company that developed the third-party InnoDB storage engine that allows MySQL to provide such functionality as transactions and foreign keys. After the acquisition, an Oracle press release mentioned that the contracts that make the company's software available to MySQL AB would be due for renewal (and presumably renegotiation) some time in 2006. During the MySQL Users Conference in April 2006, MySQL issued a press release that confirmed that MySQL and Innobase OY agreed to a "multi-year" extension of their licensing agreement. In February 2006, Oracle Corporation acquired Sleepycat Software, makers of the Berkeley DB, a database engine providing the basis for another MySQL storage engine. This had little effect, as Berkeley DB was not widely used, and was dropped (due to lack of use) in MySQL 5.1.12, a pre-GA release of MySQL 5.1 released in October 2006. In January 2008, Sun Microsystems bought MySQL for $1 billion. In April 2009, Oracle Corporation entered into an agreement to purchase Sun Microsystems, then owners of MySQL copyright and trademark. Sun's board of directors unanimously approved the deal. It was also approved by Sun's shareholders, and by the U.S. government on 20 August 2009. On 14 December 2009, Oracle pledged to continue to enhance MySQL as it had done for the previous four years.

LIMITATIONS

When using some storage engines other than the default of InnoDB, MySQL does not comply with the full SQL standard for some of the implemented functionality, including foreign key references and check constraints. Up until MySQL 5.7, triggers are limited to one per action / timing, meaning that at most one trigger can be defined to be executed after an INSERT operation, and one before INSERT on the same table. No triggers can be defined on views. MySQL database's inbuilt functions like UNIX_TIMESTAMP() will return 0 after 03:14:07 UTC on 19 January 2038.

HYPERTEXT MARKUP LANGUAGE (**HTML**) is the standard markup language for

creating web pages and web applications. With Cascading Style Sheets (CSS), and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web. Web browsers receive HTML documents from a webserver or from local storage and render them into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document. HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects, such as interactive forms may be embedded into the rendered page. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by *tags*, written using angle brackets. Tags such as <img /> and <input /> introduce content into the page directly. Others such as ... surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page. HTML can embed programs written in a scripting language such as JavaScript which affect the behavior and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), maintainer of both the HTML and the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

DEVELOPMENT

In 1980, physicist Tim Berners-Lee, a contractor at CERN, proposed and prototyped ENQUIRE, a system for CERN researchers to use and share documents. In 1989, Berners-Lee wrote a memo proposing an Internet-based hypertext system. Berners-Lee specified HTML and wrote the browser and server software in late 1990. That year, Berners-Lee and CERN data systems engineer Robert Cailliau collaborated on a joint request for funding, but the project was not formally adopted by CERN. In his personal notes from 1990 he listed "some of the many areas in which hypertext is used" and put an encyclopedia first. The first publicly available description of HTML was a document called "HTML Tags", first mentioned on the Internet by Tim Berners-Lee in late 1991. It describes 18 elements comprising the initial, relatively simple design of HTML. Except for the hyperlink tag, these were strongly influenced by SGMLguid, an in-house Standard Generalized Markup Language (SGML)-based documentation format at CERN. Eleven of these elements still exist in HTML 4. HTML is a markup language that web browsers use to interpret and compose text, images, and other material into visual or audible web pages. Default characteristics for every item of HTML markup are defined in the browser, and these characteristics can be altered or enhanced by the web page designer's additional use of CSS. Many of the text elements are found in the 1988 ISO technical report TR 9537 *Techniques for using SGML*, which in turn covers the features of early text formatting languages such as that used by the RUNOFF command developed in the early 1960s for the CTSS (Compatible Time-Sharing System) operating system: these formatting commands were derived from the commands used by typesetters to manually format documents.

EXTENSIBLE MARKUP LANGUAGE (XML)

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The W3C's XML 1.0 Specification and several other related specifications—all of them free open standards—define XML. The design goals of XML emphasize simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services. Several schema systems exist to aid in the definition of XML-based languages, while programmers have developed many application programming interfaces (APIs) to aid the processing of XML data.

APPLICATIONS OF XML

As of 2009, 100 document formats using XML syntax had been developed, including RSS, Atom, SOAP, and XHTML. XML-based formats became the default for many office-productivity tools, including Microsoft Office (Office Open XML), OpenOffice.org and LibreOffice (OpenDocument), and Apple's iWork. XML has also provided the base language for communication protocols such as XMPP. Applications for the Microsoft .NET Framework use XML files for configuration. Apple has an implementation of a registry based on XML. XML has come into common use for the interchange of data over the Internet. IETF RFC 7303 gives rules for the construction of Internet Media Types for use when sending XML. It also defines the media types *application/xml* and *text/xml*, which say only that the data is in XML, and nothing about its semantics. The use of *text/xml* has been criticized as a potential source of encoding problems and it has been suggested that it should be deprecated. RFC 7303 also recommends that XML-based languages be given media types ending in *+xml*; for example *image/svg+xml* for SVG. Further guidelines for the use of XML in a networked context appear in RFC 3470, also known as IETF BCP 70, a document covering

many aspects of designing and deploying an XML-based language.

SOURCES

XML is a profile of an ISO standard SGML, and most of XML comes from SGML unchanged. From SGML comes the separation of logical and physical structures (elements and entities), the availability of grammar-based validation (DTDs), the separation of data and metadata (elements and attributes), mixed content, the separation of processing from representation (processing instructions), and the default angle-bracket syntax. Removed were the SGML declaration (XML has a fixed delimiter set and adopts Unicode as the document character set). Other sources of technology for XML were the TEI (Text Encoding Initiative), which defined a profile of SGML for use as a "transfer syntax"; and HTML, in which elements were synchronous with their resource, document character sets were separate from resource encoding, the xml:lang attribute was invented, and (like HTTP) metadata accompanied the resource rather than being needed at the declaration of a link. The ERCS(Extended Reference Concrete Syntax) project of the SPREAD (Standardization Project Regarding East Asian Documents) project of the ISO-related China/Japan/Korea Document Processing expert group was the basis of XML 1.0's naming rules; SPREAD also introduced hexadecimal numeric character references and the concept of references to make available all Unicode characters. To support ERCS, XML and HTML better, the SGML standard IS 8879 was revised in 1996 and 1998 with WebSGML Adaptations. The XML header followed that of ISO HyTime. Ideas that developed during discussion that are novel in XML included the algorithm for encoding detection and the encoding header, the processing instruction target, the xml:space attribute, and the new close delimiter for empty-element tags. The notion of well-formedness as opposed to validity (which enables parsing without a schema) was first formalized in XML, although it had been implemented successfully in the Electronic Book Technology "Dynatext" software; the software from the University of Waterloo New Oxford English Dictionary Project; the RISP LISP SGML text processor at Uniscope, Tokyo; the US Army Missile Command IADS hypertext system; Mentor Graphics Context; Interleaf and Xerox Publishing System.

CRITICISM

XML and its extensions have regularly been criticized for verbosity and complexity. Mapping the basic tree model of XML to type systems of programming languages or databases can be difficult, especially when XML is used for exchanging highly structured data between applications, which was not its primary design goal. Other criticisms attempt to refute the claim that XML is a self-describing language (though the XML specification itself makes no such claim). JSON, YAML, and S-Expressions are frequently proposed as alternatives (see Comparison of data serialization formats); that focus on representing highly structured data rather than documents, which may contain both highly structured and relatively unstructured content.

BREAKDOWN OF THE 9 MOST IN-DEMAND PROGRAMMING LANGUAGES

1. SQL

It's no surprise SQL (pronounced 'sequel') tops the job list since it can be found far and wide in various flavors. Database technologies such as MySQL, PostgreSQL and Microsoft SQL Server power big businesses, small businesses, hospitals, banks, universities. Indeed, just about every computer and person with access to technology eventually touches something SQL. For instance, all Android phones and iPhones have access to a SQL database called SQLite and many mobile apps developed Google, Skype and DropBox use it directly.

2. JAVA

The tech community recently celebrated the 20th anniversary of Java. It's one of the most widely adopted programming languages, used by some 9 million developers and running on 7 billion devices worldwide. It's also the programming language used to develop all native Android apps. Java's popularity with developers is due to the fact that the language is grounded in readability and simplicity. Java has staying power since it has long-term compatibility, which makes sure older applications continue to work now into the future. It's not going anywhere anytime soon and is used to power company websites like LinkedIn.com, Netflix.com and Amazon.com.

3. JAVASCRIPT

JavaScript – not to be confused with Java – is another one of the world's most popular and powerful programming languages, and is used to spice up web pages by making them interactive. For example, JavaScript can be used to add effects to web pages, display pop-up messages or to create games with basic functionality. It's also worth noting that JavaScript is the scripting language of the World Wide Web and is built right into all major web browsers including Internet Explorer, FireFox and Safari. Almost every website incorporates some element of JavaScript to add to the user experience, adding to the demand for JavaScript developers**.** In recent years JavaScript has also gained use as the foundation of Node.js,

a server technology that among other things enables real-time communication.

4. C#

Dating from 2000, C# (pronounced C-sharp) is a relatively new programming language designed by Microsoft for a wide range of enterprise applications that run on the .NET Framework. An evolution of C and C++, the C# language is simple, modern, type safe and object oriented.

5. C++

C++ (pronounced C-plus-plus) is a general purpose object-oriented programming language based on the earlier 'C' language. Developed by Bjarne Stroustrup at Bell Labs, C++ was first released in 1983. Stroustrup keeps an extensive list of applications written in C++. The list includes Adobe and Microsoft applications, MongoDB databases, large portions of Mac OS/X and is the best language to learn for performance-critical applications such as "twitch" game development or audio/video processing.

6. PYTHON

Python is a general purpose programming language that was named after the Monty Python (so you know it's fun to work with)! Python is simple and incredibly readable since closely resembles the English language. It's a great language for beginners, all the way up to seasoned professionals. Python recently bumped Java as the language of choice in introductory programming courses with eight of the top 10 computer science departments now using Python to teach coding, as well as 27 of the top 39 schools. Because of Python's use in the educational realm, there are a lot of libraries created for Python related to mathematics, physics and natural processing. PBS, NASA and Reddit use Python for their websites.

7. PHP

Created by Danish-Canadian programmer Rasmus Lerdorf in 1994, PHP was never actually intended to be a new programming language. Instead, it was created to be a set of tools to help Rasmus maintain his Personal Home Page (PHP). Today, PHP (Hypertext Pre-Processor) is a scripting language, running on the server, which can be used to create web pages written in HTML. PHP tends to be a popular languages since its easy-to use by new programmers, but also offers tons of advanced features for more experienced programmers.

8. RUBY ON RAILS

Like Java or the C language, Ruby is a general purpose programming language, though it is best known for its use in web programming, and Rails serves as a framework for the Ruby Language. Ruby on Rails has many positive qualities including rapid development, you don't need as much code, and there are a wide variety of 3rd party libraries available. It's used from companies ranging from small start-ups to large enterprises and everything in-between. Hulu, Twitter, Github and Living Social are using Ruby on Rails for at least one of their web applications.

9. IOS/SWIFT

In 2014, Apple decided to invent their own programming language. The result was Swift – a new programming language for iOS and OS X developers to create their next killer app. Developers will find that many parts of Swift are familiar from their experience of developing in C++ and Objective-C. Companies including American Airlines, LinkedIn, and Duolingo have been quick to adopt Swift, and we'll see this language on the rise in the coming years.

XII.CONCLUSIONS

Any great craftsman has a belt full of tools, each a perfect choice for certain situations. Similarly, there will never be just a single programming language, and each language will evolve and improve over time to keep pace with innovation. If anyone is interested in becoming a developer, it's important to be well-versed in a number of programming languages so that one can be versatile and adaptable – and then continue to learn/master languages throughout the career.

XIII. AUTHOR AND AFFILIATION

DR. S. SRIDHAR IS WORKING AS PROFESSOR IN R. V. COLLEGE OF ENGINEERING AND DIRECTOR - R. V. CENTRE FOR COGNITIVE TECHNOLOGIES, BANGALORE, KARNATAKA, INDIA. HE HOLDS A PH.D(1984) IN CSE AND HAS GUIDED SO FAR 20 PH.D'S AND ONE FROM CRANFIELD UNIVERSITY, U.K. HE HAS PUBLISHED 160 RESEARCH PAPERS IN INTERNATIONAL JOURNALS. HE HAS AUTHORED TWO BOOKS, NAMELY 'PRINCIPLES OF DISTRIBUTED DBMS' AND 'DATAMINING' UNDER 'PEARSON PUBLICATIONS' JOINTLY WITH CANADIAN AUTHORS. HE HAS CARRIED OUT 42 MNC PROJECTS FOR INTERNATIONAL AIRPORTS , 30 R & D PROJECTS FOR ONGC, INDIA. FORMERLY , HE WAS VICE CHANCELLOR-DSI-RAK, UAE , DIRECTOR-TECHNICAL P.B.COLLEGE OF ENGINEERING, ST.PETER'S ENGINEERING COLLEGE, DIRECTOR ARUNAI ENGINEERING COLLEGE, PROFESSOR & HEAD (CSE) SRM EASWARI ENGINEERING COLLEGE, PROFESSOR (CSE) ALGHURAIR UNIVERSITY, DUBAI,UAE / SKYLINE COLLEGE, SHARJAH, UAE, DIRECTOR BHARATH ENGINEERING COLLEGE, (V) PROFESSOR-COMPUTER SCIENCE & ENGINEERING ANNA UNIVERSITY, PROFESSOR AND DEAN - COMPUTER SCIENCE & ENGINEERING ST.JOSEPH ENGINEERNG COLLEGE, PROFESSOR AND DEAN - COMPUTER SCIENCE & ENGINEERING HINDUSTAN COLLEGE OF ENGINEERING, (MOSTLY ON DEPUTATION) . HE HAS

BEEN AWARDED A LOT OF MERIT CERTIFICATES IN ONGC FOR BEST PERFORMANCE, GOLD SHIELDS BY SHARJAH AIRPORT AUTHORITY, UAE FOR BEST SOFTWARE DEVELOPMENT WORK AND SELECTED AS BEST IT PROFESSIONAL BY USA BODY 3 TIMES. HE HAS CHAIRED IT SEMINARS IN CALIFORNIA ORGANISED BY INFORMIX, UK, PARIS, NICE,ACI CONFERENCE . EMAIL: DEANCCCF@RVCE.EDU.IN

XIV. REFERENCES

[1]. A Manual for BASIC, Dartmouth College Computation Center. 1964. Archived from the original (PDF) on 2012-07-16.

[2]. John G. Kemeny and Thomas E. Kurtz (1968). BASIC (4th Edition).

[3]. Sammet, Jean E. (1969). Programming languages: History and fundamentals. Englewood Cliffs, N.J.: Prentice-Hall.

[4]. Kurtz, Thomas E. (1981). "BASIC" in Richard Wexelblatt (ed.) History of programming languages I. ACM. pp. 515–537. ISBN 0-12-745040-8.

[5]. Kemeny, John G.; Kurtz, Thomas E. (1985). Back To BASIC: The History, Corruption, and Future of the Language. Addison-Wesley. pp. 141 pp. ISBN 978-0-201-13433-9.

[6]. Lien, David A. (1986). The Basic Handbook: Encyclopedia of the BASIC Computer Language (3rd ed.). Compusoft Publishing. ISBN 978-0-932760-33-3.

[7]. Adams, Jeanne C.; Brainerd, Walter S.; Hendrickson, Richard A.; Maine, Richard E.; Martin, Jeanne T.; Smith, Brian T. (2009). The Fortran 2003 Handbook (1st ed.). Springer. ISBN 978-1-84628-378-9.

[8]. Akin, Ed (2003). Object Oriented Programming via Fortran 90/95 (1st ed.). Cambridge University Press. ISBN 0-521-52408-3.

[9]. Chapman, Stephen J. (2007). Fortran 95/2003 for Scientists and Engineers (3rd ed.). McGraw-Hill. ISBN 978-0-07-319157-7.

[10]. Chivers, Ian; Sleightholme, Jane (2015). Introduction to Programming with Fortran (3rd ed.). Springer. ISBN 978-3-319-17700-7.

[11]. Etter, D. M. (1990). Structured FORTRAN 77 for Engineers and Scientists (3rd ed.). The Benjamin/Cummings Publishing Company, Inc. ISBN 0-8053-0051-1.

[12]. Ellis, T. M. R.; Phillips, Ivor R.; Lahey, Thomas M. (1994). Fortran 90 Programming (1st ed.). Addison Wesley. ISBN 0-201-54446-6.

[13]. Kupferschmid, Michael (2002). Classical Fortran: Programming for Engineering and Scientific Applications. Marcel Dekker (CRC Press). ISBN 0-8247-0802-4.

[14]. McCracken, Daniel D. (1961). A Guide to FORTRAN Programming. New York: Wiley. LCCN 61016618.

[15]. Metcalf, Michael; John Reid; Malcolm Cohen (2011). Modern Fortran Explained. Oxford University Press. ISBN 0-19-960142-9.

[16]. Nyhoff, Larry; Sanford Leestma (1995). FORTRAN 77 for Engineers and Scientists with an Introduction to Fortran 90 (4th ed.). Prentice Hall. ISBN 0-13-363003-X.

[17]. Bemer, Bob (1971). "A View of the History of COBOL" (PDF). Honeywell Computer Journal. Honeywell. **5** (3). Retrieved 28 June 2014.

[18]. Beyer, Kurt (2009). Grace Hopper and the Invention of the Information Age. MIT Press. ISBN 978-0262013109. LCCN 2008044229.

[19]. Brown, William R. (1 December 1976). "COBOL". In Belzer, Jack; Holzman, Albert G.; Kent, Allen. Encyclopedia of Computer Science and Technology: Volume 5. CRC Press. ISBN 978-0824722555.

[20]. Carr, Donald E.; Kizior, Ronald J. (31 December 2003). "Continued Relevance of COBOL in Business and Academia: Current Situation and Comparison to the Year 2000 Study" (PDF). Information Systems Education Journal. AITP. **1** (52). ISSN 1545-679X. Retrieved 4 August 2014.

[21]. CODASYL (July 1969). "CODASYL COBOL Journal of Development 1968". National Bureau of Standards. ISSN 0591-0218. LCCN 73601243.

[22]. Conner, Richard L. (14 May 1984). "Cobol, your age is showing". Computerworld. International Data Group. **18** (20): ID/7–ID/18. ISSN 0010-4841.

[23]. Cutler, Gary (9 April 2014). "GNU COBOL Programmer's Guide" (PDF) (3rd ed.). Retrieved 25 February 2014.

[24]. Garfunkel, Jerome (1987). The COBOL 85 Example Book. Wiley. ISBN 0471804614.

[25]. ISO/IEC JTC 1/SC 22/WG 4 (4 December 2001). "ISO/IEC IS 1989:2001 – Programming language COBOL". ISO.

Archived from the original (ZIP of PDF) on 23 January 2002. Retrieved 2 September 2014.

[26]. ISO/IEC JTC 1/SC 22/WG 4 (31 October 2014). INCITS/ISO/IEC 1989:2014 [2014] – Programming language COBOL. INCITS.

[27]. Klein, William M. (4 October 2010). "The History of COBOL" (PDF). Archived from the original (PDF) on 7 January 2013. Retrieved 7 January 2014.

[28]. Niklaus Wirth: The Programming Language Pascal. 35–63, Acta Informatica, Volume 1, 1971.

[29]. 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.

[30]. C. A. R. Hoare, Niklaus Wirth: An Axiomatic Definition of the Programming Language Pascal. 335–355, Acta Informatica, Volume 2, 1973.

[31]. Kathleen Jensen and Niklaus Wirth: PASCAL – User Manual and Report. Springer-Verlag, 1974, 1985, 1991, ISBN 0-387-97649-3 and ISBN 3-540-97649-3.

[32]. Niklaus Wirth: Algorithms + Data Structures = Programs. Prentice-Hall, 1975, ISBN 0-13-022418-9.

[33]. Niklaus Wirth: An assessment of the programming language PASCAL. 23–30 ACM SIGPLAN Notices Volume 10, Issue 6, June 1975. References

[34]. "Changes in Release 8.0.0 (Development Milestone)". MySQL 8.0 Reference Manual. Oracle Corporation. 12 September 2016. Retrieved 12 December 2016.

[35]. "MySQL: Project Summary". Ohloh. Black Duck Software. Retrieved 17 September 2012.

[36]. "Supported Platforms: MySQL Database". Oracle. Retrieved 24 March 2014.

[37]. "What is MySQL?". MySQL 5.1 Reference Manual. Oracle. Retrieved 17 September 2012. The official way to pronounce "MySQL" is "My Ess Que Ell" (not "my sequel")

[38]. "History of MySQL". MySQL 5.1 Reference Manual. MySQL AB. Retrieved 26 August 2011.

[39]. Urlocker, M. Zack (13 December 2005). "Google Runs MySQL". The Open Force. M. Zack Urlocker. Retrieved 3 August 2010. AdWords was built using the MySQL database

[40]. Claburn, Thomas (24 April 2007). "Google Releases Improved MySQL Code". InformationWeek. InformationWeek. Retrieved 30 November 2008.

[41]. Callaghan, Mark (13 April 2010). MySQL at Facebook. YouTube. Google. Retrieved 3 August 2010. x,000 servers, ... Master-slave replication, InnoDB

[42]. Sobel, Jason (21 December 2007). "Keeping Up". The Facebook Blog. Facebook. Retrieved 30 October 2008.

[43]. Malik, Om (25 April 2008). "Facebook's Insatiable Hunger for Hardware". GigaOM. GigaOmniMedia. Retrieved 30 October 2008.

[44]. Cole, Jeremy (14 April 2011). Big and Small Data at @Twitter. YouTube. Google. Retrieved 20 October 2011.

[45]. N. Wirth, and A. I. Wasserman, ed: Programming Language Design. IEEE Computer Society Press, 1980

[46]. D. W. Barron (Ed.): Pascal – The Language and its Implementation. John Wiley 1981, ISBN 0-471-27835-1

[47]. Peter Grogono: Programming in Pascal, Revised Edition, Addison-Wesley, 1980

[48]. Richard S. Forsyth: Pascal in Work and Play, Chapman and Hall, 1982

[49]. Banahan, M.; Brady, D.; Doran, M. (1991). The C Book (2nd ed.). Addison-Wesley.

[50]. King, K. N. (April 2008). C Programming: A Modern Approach (2nd ed.). Norton. ISBN 978-0-393-97950-3.

[51]. Thompson, Ken. "A New C Compiler" (PDF). Murray Hill, New Jersey: AT&T Bell Laboratories.

[52]. Feuer, Alan R. (1998). The C Puzzle Book (1st, revised printing ed.). Addison-Wesley. ISBN 978-0-201-60461-0. Abrahams, David; Gurtovoy, Aleksey. C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond. Addison-Wesley. ISBN 0-321-22725-5.

[53]. Alexandrescu, Andrei (2001). Modern C++ Design: Generic Programming and Design Patterns Applied. Addison-Wesley. ISBN 0-201-70431-5.

[54]. Alexandrescu, Andrei; Sutter, Herb (2004). C++ Design and Coding Standards: Rules

and Guidelines for Writing Programs. Addison-Wesley. ISBN 0-321-11358-6.

[55]. Becker, Pete (2006). The C++ Standard Library Extensions : A Tutorial and Reference. Addison-Wesley. ISBN 0-321-41299-0.

[56]. Brokken, Frank (2010). C++ Annotations. University of Groningen. ISBN 90-367-0470-7.

[57]. Coplien, James O. (1994) [reprinted with corrections, original year of publication 1992]. Advanced C++: Programming Styles and Idioms. ISBN 0-201-54855-0.

[58]. Dewhurst, Stephen C. (2005). C++ Common Knowledge: Essential Intermediate Programming. Addison-Wesley. ISBN 0-321-32192-8.

[59]. Information Technology Industry Council (15 October 2003). Programming languages – C++ (Second ed.). Geneva: ISO/IEC. 14882:2003(E).

[60]. Josuttis, Nicolai M. (2012). The C++ Standard Library, A Tutorial and Reference (Second ed.). Addison-Wesley. ISBN 0-321-62321-5.

[61]. Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, Alex (2014). The Java® Language Specification (PDF) (Java SE 8 ed.).

[62]. Gosling, James; Joy, Bill; Steele, Guy L., Jr.; Bracha, Gilad (2005). The Java Language Specification (3rd ed.). Addison-Wesley. ISBN 0-321-24678-0.

[63]. Lindholm, Tim; Yellin, Frank (1999). The Java Virtual Machine Specification (2nd ed.). Addison-Wesley. ISBN 0-201-43294-3.

[64]. "HTML 4.0 Specification — W3C Recommendation — Conformance: requirements and recommendations". World Wide Web Consortium. December 18, 1997. Retrieved July 6, 2015.

[65]. Tim Berners-Lee, "Information Management: A Proposal." CERN (March 1989, May 1990). W3.org

[66]. "Tags used in HTML". World Wide Web Consortium. November 3, 1992. Retrieved November 16, 2008.

[67]. "First mention of HTML Tags on the www-talk mailing list". World Wide Web Consortium. October 29, 1991. Retrieved April 8, 2007.

[68]. "Index of elements in HTML 4". World Wide Web Consortium. December 24, 1999. Retrieved April 8, 2007.

[69]. Tim Berners-Lee (December 9, 1991). "Re: SGML/HTML docs, X Browser (archived www-talk mailing list post)". Retrieved June 16, 2007.

[70]. Berners-Lee, Tim; Connolly, Daniel (June 1993). "Hypertext Markup Language (HTML): A Representation of Textual Information and MetaInformation for Retrieval and Interchange". w3.org. Retrieved 2017-01-04. Annex A of ISO 8879:1986 (SGML)

[71]. Lawrence A. Cunningham (2005). "Language, Deals and Standards: The Future of XML Contracts". Washington University Law Review. SSRN 900616.

[72]. Bosak, Jon; Bray, Tim (May 1999). "XML and the Second-Generation Web". Scientific American. Archived from the original on 1 October 2009.

[73]. Kelly, Sean (February 6, 2006). "Making Mistakes with XML". Developer.com. Retrieved 26 October 2010.

[74]. St. Laurent, Simon (February 12, 2003). "Five years later, XML..". O'Reilly XML Blog. O'Reilly Media. Retrieved 26 October 2010.

[75]. "W3C XML is Ten!". World Wide Web Consortium. 12 February 2008. Retrieved 26 October 2010.

[76]. "Introduction to XML" (PDF). Course Slides. Pierre Geneves. October 2012.

