COMP6411 COMPARATIVE STUDY OF PROGRAMMING LANGUAGES

Part 1:
History of Programming Languages

Learning Objectives

- History of the early programming languages
- Influential programming languages

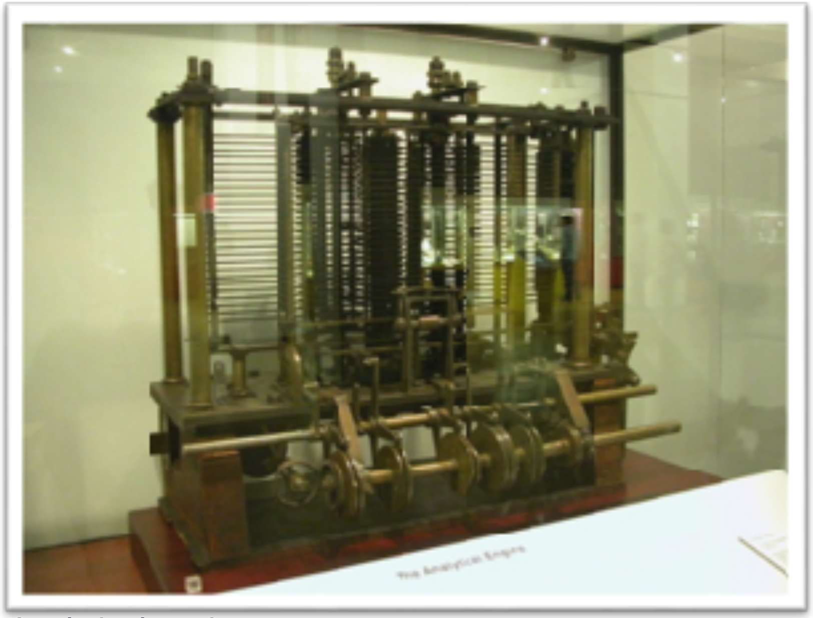
PRE-HISTORY

Analytical engine (1837)

- Important step in the history of computers.
- **Mechanical** general-purpose computer.
- Developed by the British mathematician **Charles Babbage**.
- **Ada Lovelace** has been credited with the first “program” to be executed by the machine.

Ada Lovelace

Charles Babbage


Analytical engine

Analytical engine (1837)

- In its logical design the machine was essentially **modern**, anticipating the first completed general-purpose computers by about 100 years.

- The **input** (programs and data) was to be provided via punched cards, a method being used at the time to direct mechanical looms such as the Jacquard loom.

- For **output**, the machine would have a printer, a curve plotter and a bell. The machine would also be able to punch numbers onto cards to be read in later.

Analytical engine (1837)

- It employed base-10 fixed-point arithmetic. There was to be a store (i.e., a **memory**) capable of holding 1,000 numbers of 50 decimal digits each, for a total capacity of 20.7KB.

- An **arithmetical unit** (the "mill") would be able to perform all four arithmetic operations, plus comparisons and optionally square roots.

- Like the central processing unit (CPU) in a modern computer, the mill would rely upon its own hard-coded internal **procedures**, to be stored in the form of pegs inserted into rotating drums called "barrels," in order to carry out some of the more complex instructions the user's program might specify.

Analytical engine (1837)

- The programming language to be employed by users was akin to modern day **assembly languages**.

- Loops and conditional branching were possible and so the language as conceived would have been **Turing-complete**, one hundred years before Alan Turing's concept of **Turing Machine** and **Turing Completeness**.

Alan Turing

BIRTH OF PROGRAMMING LANGUAGES

Plankalkül (1941)

- Plankalkül is a computer language developed for engineering purposes by **Konrad Zuse**, a German Civil Engineer and inventor.
- It was invented as the programming language for the Z3, the first **programmable** electromechanical computing machine, which Zuse built on his own in 1941.

Konrad Zuse

- It was the **first high-level** non-von Neumann programming language to be designed for a computer and was designed between 1943 and 1945.
- It was **not known** for a long time owing to a combination of factors such as conditions in wartime and postwar Nazi Germany.

Plankalkül (1941)

- By 1946, Zuse had written a book on the subject but it remained unpublished.
- In 1948 Zuse published a paper about Plankalkül in the "Archiv der Mathematik" but still did not attract much feedback - for a long time to come programming a computer would only be thought of as programming with machine code.
- Plankalkül was eventually more comprehensively published in **1972** and the first compiler for it was implemented in 1998.
- Another independent implementation followed in the year 2000 by the Free University of Berlin.

Plankalkül (1941)

- Plankalkül drew comparisons to APL and relational algebra.
- It includes assignment statements, subroutines, conditional statements, iteration, floating point arithmetic, arrays, hierarchical record structures, assertions, exception handling, and other advanced features.
- Thus, this language included many of the syntactical elements of structured programming languages that would be invented later, but it failed to be recognized widely.

Short Code (1949)

- Short Code was the first higher-level language ever developed **and used** for a computer.
- Short Code was designed in 1949 by **John Mauchly**, co-inventor of UNIVAC I, the first **commercial computer** produced in the United States.

John Mauchly

UNIVAC

Short Code (1949)

- Unlike machine code, Short Code statements represented **mathematical expressions** rather than a machine instruction.
- While Short Code represented expressions, the representation itself was not direct and required a conversion process then called automatic programming.
- Along with basic arithmetic, Short Code allowed for branching and calls to a **library of functions**.
- The language was **interpreted** and ran about 50 times slower than machine code.

A-0 (1951)

- The A-0 system, written by **Grace Hopper** in 1951 and 1952 for the UNIVAC I, was the **first compiler system** ever developed for a computer.

Grace Hopper

Grace Hopper and UNIVAC

A-0 (1951)

- A-0 functioned more as a loader/linker than the modern notion of a compiler.
- A program was specified as a sequence of **subroutines** and **arguments**. The subroutines were identified by a numeric code and the arguments to the subroutines were written directly after each subroutine code.
- The A-0 system converted the specification into machine code that could be fed into the computer in a second pass to execute the program.

A-0 (1951)

- A similar systems was invented independently by David Wheeler and **Maurice Wilkes** for the EDSAC computer in 1951.

Maurice Wilkes and EDSAC

EVOLUTION OF PROGRAMMING LANGUAGES

FORTRAN

Fortran

- IBM Mathematical Formula Translating System, later popularly know as Fortran.
- Originally developed by a team lead by **John Backus** at **IBM** in the 1950s for scientific and engineering applications on the IBM704, introduced in 1954.
- General-purpose, **procedural**, **imperative** programming language that is especially suited to numeric computation and scientific computing.

John Backus

- Originally designed to improve on the **economics of programming**, as programming using low level languages had become to be more costly than the time it actually saved.

Fortran

- The solution was to design a **higher level** programming language enabling scientists to write programs using a **mathematical notation/ language**.
- Great emphasis was put on the **efficiency** of the translated machine code, which is still the case today, explaining why Fortran programs are still considered as a **benchmark** for execution speed.
- Emphasis was put on **number computations**. Only much later was it possible to have Fortran programs manipulate characters or strings.

Fortran

- **Features** introduced by earlier versions of Fortran:
  - Comments
  - Assignment statement using complex expressions
  - Control structures (conditional, loop)
  - Subroutines and functions used similarly to the mathematical notion of function
  - Formatting of input/output
  - Machine independent code
  - Procedural programming
  - Arrays
  - Early development of compilers
  - Showed the importance/possibility/relevance of higher-level programming languages
  - Compiler development tools/techniques
  - Optimizing compiler

Fortran

- Reasons explaining Fortran’s early **success**:
  - Made efficient use of programmer’s time
  - Easy to learn and use by scientists
  - Was fully supported by IBM for many years
  - At the time, most applications and users were scientific
  - Simplified many tedious tasks such as input/output formatting
  - Compiler was generating very well optimized code
- Fortran is still one of the most popular languages in the area of high-performance computing and is the language used for programs that benchmark and rank the world's fastest supercomputers.

Fortran

- Fortran 0 (1954):
  - assignment, if, goto, do loop, formatted/unformatted I/O, pause, stop, continue, notion of basic block (entry/exit point)
- Fortran I (1955-1957)
  - First implemented compiler
- Fortran II (1958):
  - procedural programming, pass by reference, return statement, independent compilation of subroutines
- Fortran IV (1961-62):
  - machine-independence, boolean expressions, logical operations, explicit type declarations, subroutines as parameters.
- Fortran 66 (1966):
  - first standard, based on Fortran IV

Fortran

- Fortran 77 (1977-78):
  - Fixed problems in Fortran 66, string handling. Historically most important dialect.
- Fortran 90 (1991-92):
  - Includes many features of modern programming languages: recursion, modules, generic procedures, operator overloading, abstract data types, dynamic memory allocation, dynamic data structures, case statement. Also removed many obsolescent features/constructs.
- Fortran 95 (1995):
  - minor revisions, but most notably included many features of High Performance Fortran, a data-parallel version of Fortran.
- Fortran 2003 (2003):
  - Major revision. object-orientation, constructors, finalizers, asynchronous I/O, procedures pointers, interoperability with C, enhanced modularity features.
- Fortran 2008 (2010):
  - Minor upgrade vs Fortran 2003, added more constructs for parallel execution.

COBOL

Cobol

- Acronym for **CO**mmon **B**usiness-**O**riented **L**anguage
- Primary domain in business, finance, and administrative systems.
- A specification of COBOL was initially created during the second half of 1959 by **Grace Hopper**.
- The specifications were to a great extent inspired by the Grace Hopper's FLOW-MATIC and IBM's COMTRAN languages.

Left to right: Ron Hamm, John L. Jones, Dr. Jan Prokop, Oliver Smoot, Tom Rice, Donald Nelson, Commodore Grace M. Hopper, Michael O'Connell and Howard Bromberg. © Computer History Museum

Cobol

- Initial meeting for the creation of COBOL (then named CBL) was held at the Pentagon in May 1959.
- The meeting agreed to the following requirements:
  - Use of English (as business is in English)
  - Ease of use (even at expense of performance)
  - Both these requirements aimed at its use by business people
- Emphasis was put on immediate action, as many other such languages/systems were attempted at the time.
- By December 1959, a first version was defined, which was refined in 1961 and 1962.
- A standard version was defined in 1968. Further standardized evolutions were defined in 1974, 1985 and 2002.

Cobol

- First programming language advocated by the United States Department of Defense, which contributed to its success.
- The development of Cobol paralleled that of Fortran.
- It is a data processing language, making it different from Fortran and Algol.
- Puts explicit emphasis on the **description of the manipulated data** at two levels:
  - **Machine-dependent** description (environment division) providing a connection between the Cobol program and the data files as stored on a specific media.
  - **Machine-independent logical** description of data as manipulated by procedures.

Cobol

- **Positive:**
  - Macros
  - Hierarchical data structures
  - Descriptive variable names
  - Elaborated Input/Output formatting
  - Built-in searching/sorting procedures
  - Backed-up by Department of Defense
- **Negative:**
  - Poor performance of early compilers
  - Restricted to business data applications

Cobol

- Was never intended as an innovative language. It had very **limited influence** on subsequent languages.

- Cobol was much more **standardized** compared to Fortran and Algol
  - Goal: orderly wide use of the same language in business data processing.
  - Adverse effect: language was limited in its evolution, and stayed mostly the same throughout the most part of its lifetime.

- Newest standard (1990s) includes object-orientation and garbage collection.

- Despite being more than 50 years old, **tens of billions** of lines of Cobol code are still in use behind the scenes in the applications of many businesses and government systems worldwide.

LISP

Lisp

- Lisp was created by **John McCarthy** in 1958 as a practical mathematical notation for computer programs, influenced by the notation of **Alonzo Church**'s **lambda calculus**.

- Lisp pioneered many ideas in computer science, including:
  - functional programming
  - tree data structures
  - automatic storage management
  - dynamic typing
  - self-hosting compiler

Alonzo Church

John McCarthy

- The name LISP derives from "**LIS**t **P**rocessing“, as linked lists are one of Lisp language's major data structures, and Lisp source code is itself made up of lists.

Lisp

- LISP was designed by McCarthy as a language to express **symbolic computations**, i.e. programs that aim at manipulating mathematical expressions, for example:
  - simplification of expressions
  - differentiation and integration
  - polynomial factorization

- Some of the requirements for such a language was the availability of:
  - **recursion:** lambda calculus uses recursion pervasively
  - **linked lists:** to implement lambda calculus, which implied dynamic memory allocation and deallocation.

Lisp

- Lisp implementation relies on a self-hosting compiler (*eval* function). As a result, a Lisp program can manipulate its own source code as a data structure, giving rise to the macro systems that allow programmers to create new syntax or even new domain-specific languages embedded in Lisp.
- This feature is a part of what is now called “reflective programming”.
- Lisp has been used extensively for artificial intelligence. It has been the subject of many revisions and expansions, the latest and most common ones being **Scheme** and **Common Lisp**.
- Common Lisp includes the Common Lisp Object System (**CLOS**), a very original implementation of object-oriented programming.

Lisp

- Lisp pioneered the notion of **functional** programming language.
  - All computations are done by applying functions to arguments.
  - Does not have variables or assignment operator.
  - Repetitive processes are achieved by recursion.

- Other functional programming languages include:
  - **ML**: Robin Milner, University of Edinburgh, 1980s, including many other variants.
  - **Miranda**: David Turner, University of Kent, 1980s.
  - **Haskell**: Evolution of Miranda, 1990s.
  - **F#**: .NET language that provides functional and object-oriented imperative language features

ALGOL

Algol

- ALGOL (ALGOrithmic Language) is a family of imperative computer programming languages originally developed in the mid 1950s that **greatly influenced** many other programming languages.

- Introduced by a joint committee of **American** (ACM) and **European** (GAMM) experts .

- GAMM: Gesellschaft für Angewandte Mathematik und Mechanik ("Society of Applied Mathematics and Mechanics")

- ACM: Association for Computing Machinery

Some of the original designers or Algol.
Top row: John McCarthy, Fritz Bauer, Joe Wegstein.
Bottom row: John Backus, Peter Naur, Alan Perlis.

Algol

- According to John Backus, the original goals of the original meeting in ETH Zurich were:
  - To provide a means of **communicating** numerical methods and other procedures between people
  - To provide a means of realizing a **stated process** on a **variety of machines**.

- Original objectives of Algol:
  - Machine independence
  - As close as possible to mathematical notation
  - Usable for the description of algorithms in publications
  - Easily translatable to machine instructions
  - Avoid some of the perceived problems with FORTRAN
  - Language simplicity

Algol

- Original contributions of the various versions of Algol:
  - Code blocks, local variables, static scoping
  - Compound statement
  - Structured control statements improved from Fortran (for, do, if, switch)
  - Recursion
  - Multidimensional and dynamic arrays
  - Dynamic memory allocation (stack)
  - Pass by name, pass by value parameter passing
  - Nested functions
  - User-defined data types
  - Parallelism/concurrency/semaphores
  - Type casting
  - Language translation techniques such as syntax-directed translation
  - First language to be defined using a formal notation (BNF)

Algol

- John Backus developed the Backus Normal Form method of describing programming languages syntax specifically for ALGOL 58.
- It was revised and expanded by Peter Naur for ALGOL 60, and at **Donald Knuth**'s suggestion renamed **Backus–Naur Form**.

Donald Knuth

Peter Naur

- The Backus-Naur Form has been used ever since in the design of programming languages.

Algol

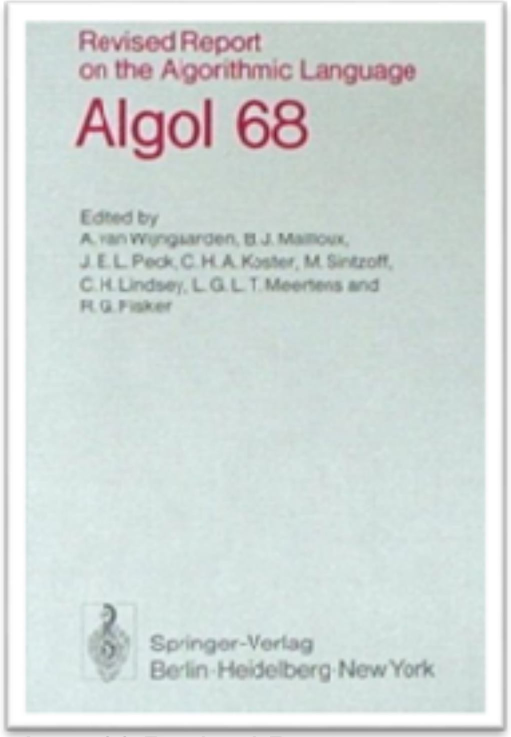
- Algol-58:
  - First version, originally named **IAL**: International Algorithmic Language.
  - Was implemented first in Europe, but met with some resistance in the United States due to the predominance of Fortran at IBM.
  - Starting in 1959 in Communications of the ACM, Algol was used to represent algorithms, and this for the next 30 years.

- Algol-60:
  - First language to include lexical scoping, pass by value and pass by name, recursion, dynamic arrays.
  - First language to be standardized (with COBOL)
  - ALGOL 60 inspired many languages that followed it. **C. A. R. Hoare** remarked: *"Here is a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors."*

Tony Hoare

Algol

- Algol-68:
  - Evolution of Algol-60 that incorporated many innovative and important features that became part of many other languages to come.
    - elaborated types (short, long, loc, heap, reference, etc), type casting, union type.
    - array slicing
    - operators
    - I/O formatting: “transput”
    - parallel processing
  - Resulted in a much more complex language, which eventually was detrimental to the success of Algol, as reported by C.A.R. Hoare, **Edsger Dijkstra** and **Nicklaus Wirth**.
- Algol-W (1966):
  - Nicklaus Wirth early development of **Pascal**.



Revised Report
on the Algorithmic Language
Algol 68

Edited by
A. van Wijngaarden, B. J. Mailloux,
J. E. L. Peck, C. H. A. Koster, M. Sintzoff,
C. H. Lindsey, L. G. L. T. Meertens and
R. G. Fisker

Springer-Verlag
Berlin · Heidelberg · New York

Algol 68 Revised Report

Nicklaus Wirth

Edsger Dijkstra

ALGOL VS. FORTRAN

Algol vs. Fortran

- Algol introduced many more **innovative features** than Fortran.
- Yet, Fortran met with **more success** than Algol.

- But did it, really?
  - Success is very much subjective and multifaceted.
  - If we consider **commercial** success, Fortran was as huge success.
  - However, if we consider **influential** success, Algol was much more successful.

- To make a parallel with spoken languages:
  - **Latin** is not used anymore.
  - However, many languages used now are based on Latin.
  - So, Latin itself is “dead”, but its **successors** are striving.

Algol vs. Fortran

- Some reasons why Algol did not become more popular than Fortran:
  - Fortran was already established as Algol was under development.
  - IBM initially supported Algol, but later stopped supporting it.
  - Language adoption follows “**snowball effect**”.
  - Algol had more syntax features, with the following consequences:
    - Harder to learn than Fortran.
    - Compilers were much more complex to develop. Advanced compiler design methods and tools were not available at the time.
    - Complexity led to less efficient generated machine code.

Algol vs. Fortran

- Some reasons why Algol did not become more popular than Fortran:
  - Fortran was seen as a **stable**, **practical** and very **efficient** tool.
  - Algol was introducing more and more advanced “intellectual” features, whose implementation was **problematic for performance**.
  - Fortran was developed by people who were all focused on the **practical** aspects of its use, and concentrated on making a stable version of Fortran work on the new hardware as it became available.
  - Algol was developed by scientists that were concerned with the **theoretical** definition of the language, developing new translation techniques, and exploring new avenues for programming. These endeavors were not seen as immediately practical by many computer users.

Algol vs. Fortran

- Some reasons why Algol was more influential than Fortran
  - Algol was a landmark in the development of programming language theory and implementation, including the **Backus-Naur form**, **syntax-directed translation**, and **dynamic memory allocation**.
  - Algol was a direct descendent of **Pascal**, which soon became one of the main language used to teach programming all over the world.
  - Another direct descendent of Algol was **Ada**, which was adopted as mandatory language by the United States Department of Defence.
  - Algol was used for more than 30 years in scientific publications for the expression of algorithms.

Algol vs. Fortran

- Most influential original contributions of Algol:
  - User-defined data structures
  - Reference types
  - Compound statement, code block, scoping
  - Concurrency
  - Type casting
  - Dynamic arrays
  - Operator overloading
- All of these language features are used in modern programming languages.

PL/I

PL/I

- With **Fortran** and **Cobol**, there were two distinct “kinds” of programmers. Fortran emphasized floating point arithmetic, arrays, procedures, fast computation. Cobol emphasized decimal arithmetic, fast asynchronous input/output, string handling, efficient search/sort routines.
- More and more, there came a need for all these features to be incorporated in a single language.

- When IBM designed the famous IBM360 computer, it saw the opportunity to design a new **general-purpose language** designed specifically for it.
- This language was named PL/I (Programming Language 1). It was originally developed in 1964-1966.

IBM 360

PL/I

- Goals of PL/I:
  - Provide features of Fortran, augmented with the capacities of Cobol.
  - Clarity of language.
  - Incorporating current programming languages **innovations**.
  - Performance of compiled code competitive with that of Fortran.
  - Extensibility to new hardware and other **application areas**.
  - Improve programming productivity - transferring effort from programmer to compiler.
  - Machine and operating system independence.

PL/I

- This created a language with very numerous features and requirements, including many innovative ones:
  - Reference data types
  - User-defined data types
  - Concurrency
  - Dynamic arrays
  - Auto-correcting compiler
  - String handling
  - Machine independence
  - Efficiency
  - Exception handling
  - I/O control system
  - Report-generation
  - Recursion

PL/I

- The first attempts at meeting these goals was mostly based on an evolution of Fortran (name Fortran VI).

- It was quickly realised to be infeasible, and a new language was designed and named **NPL** (New Programming Language), defined in 1964, implemented in the UK IBM Hursley Laboratory, then renamed PL/I in 1965.

- Although the idea of creating a general-purpose language was a noble idea, the PL/I experience now serves as an example as to the difficulties involved in designing a general-purpose language.

- Though the language was easy to learn and use, implementing a PL/I compiler was difficult and time-consuming.

PL/I

- It has been claimed that a PL/I compiler was two to four times as **large** as comparable Fortran or COBOL compilers, and also that much **slower** - fortunately offset by gains in programmer productivity. This was anticipated in IBM before the first compilers were written.
- The effort needed to produce good and **efficient object code** was underestimated during the initial design of the language.
- It contained many rarely used features, such as multitasking and exception handling, which added cost and complexity to the compiler, for features seldom used.

PL/I

- On the positive side, full support for pointers to all data types (including pointers to structures), recursion, multitasking, string handling, and extensive built-in functions PL/I was indeed quite a leap forward compared to the programming languages of its time.
- However, these were not enough to convince a majority of programmers or companies/organizations to switch to PL/I.
- Despite all this, PL/I was both a significant advance in programming languages and a good commercial and academic success.

PASCAL

Pascal

- Pascal is a very influential imperative and procedural programming language, designed in 1968-1969 and published in 1970 by **Niklaus Wirth**.

- Small and efficient language intended to encourage **good programming practices** using structured programming and data structuring.

Nicklaus Wirth

- Pascal is based on the ALGOL-60 programming language and named in honor of the French mathematician and philosopher Blaise Pascal.

Pascal

- Pascal, in its original form, is a purely procedural language and includes the traditional array of ALGOL-60 control structures.

- Pascal also has many data structuring facilities and other abstractions which were not included in the original ALGOL 60, such as:
  - user-defined type definitions
  - records
  - case statement
  - pointers
  - enumerations
  - sets

- Such constructs were inspired from Simula 67, ALGOL 68, and Wirth and Hoare’s ALGOL W.

Pascal

- Before, and leading up to Pascal, Wirth developed the language **Euler**, followed by **Algol-W**.
- Wirth subsequently developed the **Modula-2** and **Oberon**, languages similar to Pascal.
- Initially, Pascal was largely, but not exclusively, intended to teach students **structured programming**, due to a great combination of **simplicity** and **expressivity**.
- Generations of students worldwide have used Pascal as an introductory language in undergraduate courses, lasting from ~1975 to ~1995.
- A derivative known as Object Pascal was designed for object oriented programming. Object Pascal is the main programming language used in the **Delphi** programming environment.

SIMULA

Simula

- Simula was developed in the 1960s at the Norwegian Computing Center in Oslo, by **Ole-Johan Dahl** and **Kristen Nygaard**.
- Syntactically, it is a superset of Algol 60.

Ole-Johan Dahl and Kristen Nygaard

- Simula 67 introduced **objects**, **classes**, **subclasses**, **virtual methods**, **coroutines**, **discrete event simulation**, and featured **garbage collection**.
- Simula is considered the first **object-oriented** programming language.

Simula

- As its name implies, Simula was designed for doing simulation.
- Simulation systems are modelled by a series of **state changes** that occur in **parallel** through the **interaction** of the **elements** of the system as they compete for restricted system resources.
- Each type of element of the system is composed of its internal state, as well as a set of procedures that define its own behaviour.
- Elements interact with one another as a system by calling some of the other elements' procedures.
- This is the baseline of what we now call object-oriented programming.

Simula

- Simula was based on Algol 60 with the addition of the concept of **class**.
- Simula allowed to declare **classes** of objects, create **instances** of these classes, name the instances, and even form a **hierarchical structure** of class declarations.
- Simula was designed solely as a language to be used for simulations. However, in retrospect, its concept can clearly be used as a general-purpose language.
- The impact of Simula on subsequent languages is great but little recognized, as most people think that object-oriented programming is a relatively new concept.

ISWIM

ISWIM

- ISWIM is an abstract computer programming language devised by **Peter J. Landin** and first described in his article, *The Next 700 Programming Languages*, published in the Communications of the ACM in 1966.

Peter Landin

- The acronym stands for "**I**f you **S**ee **W**hat **I** **M**ean".

- Although not implemented, it has proved very influential in the development of subsequent programming languages, especially functional programming languages such as SASL, Miranda, ML, Haskell and their successors.

ISWIM

- Landin was inspired by LISP’s relationship with Lambda Calculus for the creation of ISWIM.
- An ISWIM program is a single expression qualified by '**where**' clauses (auxiliary definitions including equations among variables), conditional expressions and function definitions. ‘Where' clauses are now used widely in functional and declarative programming languages.
- Peter Landin was also involved in the creation of Algol.
- In his Turing Award lecture, Tony Hoare credited John Backus, Edsger Dijkstra and Peter Landin with introducing him to Algol and recursion, which led him to invent his famous **Quicksort** algorithm.

C

C

- C is a general-purpose programming language developed between 1969 and 1973 by **Dennis Ritchie** at the Bell Telephone Laboratories for use with the Unix operating system.
- Although C was designed for implementing **system software**, it is also widely used for developing portable application software.

Dennis Ritchie

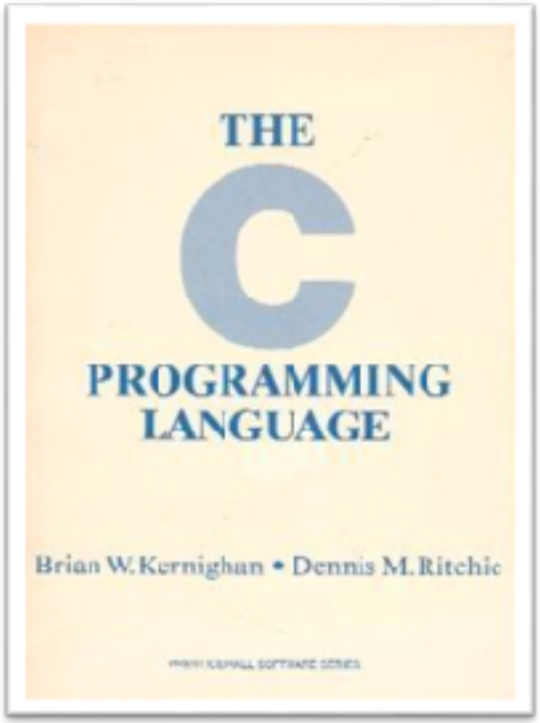
- C has become one of the most **popular** programming languages. It is widely used on many different software platforms, and there are few computer architectures for which a C compiler does not exist. C has greatly **influenced** many other programming languages, most notably C++, which originally began as an extension to C.

C

- C was influenced by the languages BCPL and B (systems languages), who were influenced by Algol and Fortran.

- C was originally designed in 1969-1972 with the following goals in mind:
  - 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
  - to require **minimal run-time** support

- In 1977-1979, portability emerged when portability of the Unix operating system was being demonstrated.



THE C PROGRAMMING LANGUAGE

Brian W.Kernighan • Dennis M.Ritchie

- In 1978, the famous book ***The C Programming Language*** was published, written by Brian Kernigan and Dennis Ritchie.

C

- The **Unix** operating system was and is still widely used on many computers.
- All Unix operating systems included a C compiler and was itself developed using the C language.
- Thus, C was readily **available** on virtually all machines and was proven to be powerful enough to create an operating system.
- Thus, many programmers started using it, creating an ever-growing code base and libraries and tools written in C.
- C is a typical example of “**snowball-effect**” programming language popularity.

C

- C exhibits the following specific characteristics:
    - variables may be hidden in nested blocks
    - partially weak typing
    - low-level access to computer memory using typed pointers
    - function and data pointers supporting ad hoc run-time polymorphism
    - array indexing as a secondary notion, defined in terms of pointer arithmetic
    - a preprocessor for macro definition, source code file inclusion, and conditional compilation
    - complex functionality such as I/O, string manipulation, and mathematical functions consistently delegated to library routines

C

- C does not have some features that are available in some other programming languages:
  - No direct assignment of arrays or strings
  - No automatic garbage collection
  - No requirement for bounds checking of arrays
  - No operations on whole arrays
  - No syntax for ranges, such as the A..B notation used in several languages
  - No functions as parameters (only function and variable pointers)
  - No exception handling
  - Only rudimentary support for modular programming
  - No compile-time polymorphism
  - Only rudimentary support for generic programming (using macros)
  - Limited support for encapsulation
  - No native support for multithreading and networking

C

- At the time of its creation, C was useful for many applications that had formerly been coded in assembly language.
- Despite its **low-leve**l programming capabilities, the language was designed to encourage **machine-independent** programming.
- A standards-compliant and portably written C program can be compiled for a very wide variety of computer platforms and operating systems with little or no change to its source code.
- C has become **available** on virtually any computing platform: from embedded microcontrollers to supercomputers.

SMALLTALK

Smalltalk

- Smalltalk is an object-oriented, dynamically typed, reflective programming language.

- It was designed and created in part for educational use at the Learning Research Group of **Xerox PARC** by **Alan Kay**, **Dan Ingalls**, **Adele Goldberg**, Ted Kaehler, Scott Wallace, and others during the 1970s.

Adele Goldberg

Alan Kay

Dan Ingalls

Smalltalk

- Smalltalk emerged with Alan Kay’s great “Dynabook” insight, predicting the creation of desktop and portable computers, and their availability to everyone.
- Kay envisioned that such computers would need to have very intuitive graphics-oriented human-interface capabilities.
- Smalltalk was a programming environment developed following this vision.
- The first operational computer resulting from Kay’s vision was the Xerox Alto.

Smalltalk

- After the preliminary research versions Smalltalk-71, Smalltalk-72 and Smalltalk-76, Smalltalk was publicly released as Smalltalk-80 and has been widely used since.

- One of the early versions, Smalltalk-76, was one of the first programming languages to be implemented along with a **development environment** featuring most of the now familiar tools, including a class library code browser/editor.

- Smalltalk-like languages are in continuing active development, and have gathered loyal communities of users.

Smalltalk

- Smalltalk was the first programming environment based on a **WIMP** system. WIMP interaction was developed at Xerox PARC with the Xerox Alto in 1973 :
  - A **window** runs a self-contained program, isolated from other programs that run at the same time in other windows.
  - An **icon** acts as a shortcut to an action the computer performs.
  - A **menu** is a selection system that allows the user to execute programs or tasks associated with a window or icon.
  - The **pointer** is an onscreen symbol that represents movement of a physical device that the user controls to select windows, icons, and menu items.

Xerox Alto

Smalltalk

- In Smalltalk, **everything is an object**: from integer constants to entire software systems.
- All computing in Smalltalk is done using the same principle: message passing.
- A Smalltalk object is a data structure holding a state (through object composition), and containing methods that can be used to reply to messages sent to the object.
- A message is sent from an object sent to another object, and a reply to a message is itself also and object.
- The main difference with a procedure call is that a message is sent to a data object, who then decides with which method to reply to the received message.
- Message passing can be implemented asynchronously.
- The message passing model developed for Smalltalk was an inspiration for the development of the **Actor Model** of concurrent computing.

Smalltalk

- A Smalltalk object can do exactly three things:
  - Hold state (references to other objects).
  - Receive a message from itself or another object.
  - In the course of processing a message, send messages to itself or another object.

- The state an object holds is always private to that object.

- Other objects can query or change that state only by sending requests (messages) to the object to do so.

- Alan Kay has commented that despite the attention given to objects, messaging is the most important concept in Smalltalk.

Smalltalk

- Smalltalk-80 added **metaclasses**, to help maintain the "everything is an object" paradigm by associating properties and behavior with individual classes.
- In this view, all values are objects, even the classes themselves are objects. Each class is an instance of the metaclass of that class. Metaclasses in turn are also objects, and are all instances of a class called Metaclass. Code blocks are also objects.
- This makes Smalltalk one of the only “pure” object-oriented languages.

Smalltalk

- Smalltalk programs are usually compiled to bytecode, which is then interpreted by a virtual machine or dynamically translated into machine-native code as in just-in-time compilation.

- Smalltalk has influenced the wider world of computer programming in four main areas.
  - It inspired the syntax and semantics of other computer programming languages.
  - It was a prototype for a model of computation known as message passing, who inspired the Actor Model of concurrent computing.
  - Its WIMP GUI inspired the windowing environments of personal computers, so much so that the windows of the first Macintosh desktop looked almost identical to the windows of Smalltalk-80.
  - Its integrated development environment was the model for a generation of visual programming tools that look like Smalltalk's code browsers and debuggers.
  - The architecture of the Smalltalk programming environment developed the Model-View-Controller architecture widely used nowadays in many applications.

Smalltalk

- Smalltalk-80 is a totally **reflective** system, implemented in Smalltalk-80 itself. Smalltalk-80 provides both **structural** and **computational** reflection.
- Smalltalk is a **structurally reflective** system whose structure is defined by Smalltalk-80 objects.
- The classes and methods that define the system are themselves objects and fully part of the system that they help define.
- The Smalltalk compiler compiles textual source code into method objects, typically instances of the CompiledMethod class.
- These get added to classes by storing them in a class's method dictionary. The part of the class hierarchy that defines classes can add new classes to the system. The system is extended by running Smalltalk-80 code that creates or defines classes and methods. In this way a Smalltalk-80 system is a "living" system, carrying around the ability to extend itself at run time.

Smalltalk

- Since the classes are themselves objects, they can be asked questions such as "what methods do you implement?" or "what instance variables do you define?".
- So objects can easily be inspected, copied, (de)serialized and so on with generic code that applies to any object in the system.
- Smalltalk-80 also provides **computational reflection**, the ability to observe the computational state of the system.
- In languages derived from the original Smalltalk-80 the current activation of a method is accessible as an object named via a keyword: *thisContext*. By sending messages to *thisContext* a method activation can ask questions like "who sent this message to me".
- These facilities make it possible to implement co-routines or Prolog-like back-tracking without modifying the virtual machine.

Smalltalk

- When an object is sent a message that it does not implement, the virtual machine sends the object the *doesNotUnderstand* message with a reification of the message as an argument.
- The message (another object, an instance of Message) contains the selector of the message and an Array of its arguments. In an interactive Smalltalk system the default implementation of *doesNotUnderstand* is one that opens an error window reporting the error to the user.
- Through this and the reflective facilities the user can examine the context in which the error occurred, redefine the offending code, and continue, all within the system, using Smalltalk-80's reflective facilities.

ADA

Ada

- Ada is a structured, statically typed, imperative, wide-spectrum, and object-oriented high-level programming language, extended mainly from Pascal.

- Ada was originally designed by a team led by **Jean Ichbiah** of CII Honeywell Bull in France under contract to the **United States Department of Defense** from 1977 to 1983.

- Ada was named after Ada Lovelace (1815–1852), who is credited as being the first computer programmer.

- Notable features of Ada include: strong typing, packages, run-time checking, concurrency, exception handling, and generics. Ada 95 added support for object-oriented programming, including dynamic dispatch.

Ada

- In the 1970s, the **US Department of Defense** was concerned by the number of different programming languages being used for its embedded computer system projects, many of which were obsolete, hardware-dependent, unsafe and did not support modular programming.

- Most of the languages used were assembly languages particular to the specific processors being used in the embedded system.

- In 1975, a working group, the High Order Language Working Group (HOLWG), was formed with the intent to solve these problems by finding or creating a programming language generally suitable for the department's requirements. The result was Ada.

- Following the adoption of Ada, the total number of high-level programming languages in use by the US DoD fell significantly.

Ada

- Ada was designed following the state of the art of **software engineering** principles of the 1970s.
- It included many advanced features that were deemed necessary to reach its intended goals, such as concurrency, exception handling, modularity and safety.
- These many features proved to be **very complex to implement**, so that the first complete compiler were **only available in 1985**.
- Many detractors (including Tony Hoare) criticised Ada for its size and complexity.
- At the same time, it was regarded by many as the epitome of programming language design.

Ada

- The original Ada (Ada 83) was improved with better object-oriented features and concurrency and defined in a new standard as Ada 95.
- In 1997, following a change in strategy, the Department of Defense **removed** its requirement to use Ada in its projects.
- This, with the **emergence of C++** as a major object-oriented language at the time reduced the popularity of Ada.
- However, due to some of its compilers to require security certification, Ada is used in many **safety-critical applications**:
  - Aviation, rail, and air traffic control
  - Rocket, satellites and space systems

C++

C++

- C++ is a statically typed, multi-paradigm, compiled, general-purpose programming language.
- It is a middle-level language, as it comprises a combination of both high-level and low-level language features.
- It was developed by **Bjarne Stroustrup** starting in 1979 at Bell Laboratories as an enhancement to the **C** programming language following the object-oriented principles pioneered by **Simula**.
- Goals:
  - Augment C with the notion of classes and inheritance
  - Keep the same performance as C
  - Keep same applicability as C

Bjarne Strousroup

C++

- Bjarne Stroustrup describes some **rules** that he used for the design of C++:
  - **Statically typed**, **general-purpose** language, as **efficient and portable as C**
  - Direct and comprehensive support for **multiple programming styles** (procedural programming, data abstraction, object-oriented programming, and generic programming)
  - Give the programmer **choice**, even if this makes it possible for the programmer to choose incorrectly
  - As compatible with C as possible, providing a **smooth transition** from C
  - Avoid features that are **platform specific** or not general purpose
  - **Not incur overhead** for features that are not used
  - Function without a sophisticated **programming environment**

C++

- Stroustrup began to work on C++ in 1979. The idea of 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.
- 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. Besides C and Simula, some other languages that inspired him were ALGOL 68, Ada, CLU and ML.

C++

- Initial modifications to C included (1980):
  - type checking and conversion for function parameters
  - classes: data structures (struct) encapsulating functions
  - derived classes (inheritance)
  - public/private access modifiers
  - constructors/destructors
  - friend classes

- Later added (1981):
  - inline functions
  - default function parameters
  - overloading of assignment operator

- The resulting language was named "C with Classes".

C++

- Further evolutions (1983-1984):
    - virtual methods and dynamic binding mechanism
    - method and operator overloading
    - reference types
- After this, the language was renamed as “C++”.
- First implementation: Release 1.0 (1985):
    - implemented as a C++ to C language translation system : Cfront
- Release 2.0 (1989):
    - multiple inheritance
    - abstract classes
- Release 3.0 (1990):
    - templates
    - exception handling

C++

- **Factors for popularity:**
  - C++ is a superset of C: a C++ compiler can compile C programs, and C code and libraries can be reused in C++ classes and programs
  - C++ was designed for performance similar with C
  - A great number of reliable compilers are available for most platforms
  - C++ was the first object-oriented programming language suitable for the development of large commercial software

- **Negative aspects:**
  - C++ is a very large and complex language
  - Its low level capacities and focus on efficiency make it an insecure language
  - Some C++ development, such as the Standard Template Library add features that add security and free the programmer from low-level programming, but to the detriment of performance.

C++

- C++ is **widely used** in the software industry and in terms of usage, is **one of the most popular languages ever created**.
- Some of its application domains include systems software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games.
- C++ continues to be used and is one of the preferred programming languages to develop professional applications.
- The language has gone from being mostly Western to attracting programmers from all over the world.

JAVA

Java

- Java is a programming language originally developed by **James Gosling** at **Sun Microsystems** and released in 1995 as a core component of Sun Microsystems' **Java platform**.

James Gosling

- Java is a general-purpose, concurrent, class-based, and object-oriented, and is specifically designed to have as **few implementation dependencies** as possible.

- It is intended to let application developers **"write once, run anywhere“**, meaning that once it has been compiled, it can be executed on any platform.

Java

- It was originally designed as a language to be used for embedded electronic devices, with the following goals:
  - Simplicity
  - Reliability
  - Object-orientation
- The main languages available for that purpose were C and C++.
- It was found that these were not suitable given the stated requirements.
- Java was thus defined as a **simpler** and **safer** version of **C++**.

Java

- Java thus derives much of its syntax from C++ but has a simpler object model and fewer low-level facilities:
  - No free functions
  - No multiple inheritance
  - No pointers
  - No explicit memory management
  - Array boundary checking
  - No structs and unions
- Some additions enhance Java’s scope of application:
  - Extreme portability
  - Concurrency using threads
  - Graphical user interface libraries
  - Database adapters
  - Networking, distributed computing

Java

- Eventually, Java was not used for its original purpose.
- At the same time, the potential of the **World Wide Web** was being realized.
- Java's features were an ideal candidate to develop **browser-based applications**.
- Since it started to be used for this purpose, **the use of Java increased faster than that of any other programming language**.
- It is now used to develop applications for a wide variety of domains and platforms, and is used as a language of choice to teach programming by many institutions.

Java

- Java applications are compiled to bytecode that can be interpreted on any **Java Virtual Machine** (JVM) regardless of computer architecture.
- Due to interpretation, Java programs were initially 10 times slower than their C++ counterparts.
- 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, and optimizations in the Java Virtual Machine itself, such as **HotSpot** becoming the default for Sun's JVM in 2000.
- Since these optimizations were put in place, Java programs became as efficient as their C++ counterparts.

REFERENCES

References

1. Ada Lovelace. Women in Computer Science, American university in Bulgaria. http://cssu-bg.org/WomenInCS/ada_lovelace.php
2. John A.N. Lee. International Biographical Dictionary of Computer Pioneers. Routledge, 1995.
3. Bauer, F. L. and Wossner, H., The “Plankalkül” of Konrad Zuse: a forerunner of today's programming languages. Communications of the ACM, Vol. 15, No. 7, July 1972, pp.678-685.
4. Raúl Rojas, Cüneyt Göktekin, Gerald Friedland, Mike Krüger, Olaf Langmack, Denis Kuniß. Plankalkül: The First High-Level Programming Language and its Implementation. Technical Report B-3/2000, Freie Universität Berlin, Department of Mathematics and Computer Science, February 2000.
5. John W. Backus: The IBM 701 Speedcoding System J. ACM 1(1): 4-6 (1954). http://dl.acm.org/citation.cfm?doid=320764.320766
6. Huub de Beer. The History of the ALGOL Effort. Masters Thesis. Eindhoven University of Technology, Department of Mathematics and Computer Science, 2006. http://heerdebeer.org/ALGOL/The_History_of_ALGOL.pdf
7. Woodger, Michael. Algol. Electronic Computers, IEEE Transactions on, vol.EC-13, no.4, pp.377-381, Aug. 1964.
8. John W. Backus, Friedrich L. Bauer, Julien Green, C. Katz, John McCarthy, Alan J. Perlis, Heinz Rutishauser, Klaus Samelson, Bernard Vauquois, Joseph Henry Wegstein, Adriaan van Wijngaarden, Michael Woodger: Report on the algorithmic language ALGOL 60. Commun. ACM 3(5): 299-314 (1960). http://dl.acm.org/citation.cfm?doid=367236.367262
9. John W. Backus: The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference. IFIP Congress 1959: 125-131
10. Ronald Morrison. On the Development of Algol. PhD Thesis. University of St-Andrews, 1979. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.1475
11. Algol 60 References. http://www.masswerk.at/algol60/
12. B. Randell, L.J. Russell. Algol 60 Implementation. Academic Press, 1964. http://www.softwarepreservation.org/projects/ALGOL/book/Randell_ALGOL_60_Implementation_1964.pdf

References

13. Backus, J.W. The Syntax and Semantics of the Proposed International Algebraic Language of Zürich ACM-GAMM Conference. *Proceedings of the International Conference on Information Processing*. UNESCO. pp. 125–132, 1959.
14. Sammet, Jean. The Early History of COBOL. ACM SIGPLAN Notices (Association for Computing Machinery, Inc.) 13 (8): 121–161, 1978.
15. Radin, G. The Early History and Characteristics of PL/I. ACM SIGPLAN Notices, Vol 13, No.8, August 1978.
16. Harry Henderson. Encyclopedia of Computer Science and Technology, Revised Edition. Facts on File science library. Infobase Publishing, 2009.
17. Robert P. Rich, Internal Sorting Methods Illustrated with PL/I Programs. Englewood Cliffs: Prentice-Hall, 1972.
18. Niklaus Wirth, Helmut Weber: EULER: a generalization of ALGOL, and its formal definition: Part I. Communications of the ACM 1966; 9 (1): p. 13-25
19. Niklaus Wirth, Helmut Weber: EULER: a generalization of ALGOL, and its formal definition: Part II. Communications of the ACM 1966; 9 (2): p. 89-99
20. Niklaus Wirth: The Programming Language Pascal. 35–63, Acta Informatica, Volume 1, 1971.
21. 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.
22. Peter Grogono: Programming in Pascal, Revised Edition, Addison-Wesley, 1980.
23. Niklaus Wirth: Algorithms + Data_Structures = Programs Prentice-Hall, 1975.
24. Peter J. Landin. The next 700 programming languages. Communications of the ACM 9 (3): 157–166, March 1966.
25. C. Antony R. Hoare, ACM Turing Award Lecture: The Emperor's Old Clothes. Published in the Communications of the ACM, 1980.
26. Dennis M. Ritchie. The development of the C language. In Proceeding of HOPL-II The second ACM SIGPLAN conference on History of programming languages, pp 201 – 208, ACM New York, NY, USA, 1993

References

27. Backus, J.W. The Syntax and Semantics of the Proposed International Algebraic Language of Zürich ACM-GAMM Conference. *Proceedings of the International Conference on Information Processing*. UNESCO. pp. 125–132, 1959.
28. Brian W. Kernighan, Dennis M. Ritchie. The C Programming Language. Englewood Cliffs, NJ: Prentice Hall, 1978.
29. Goldberg, Adele; Robson, David. Smalltalk-80: The Language and its Implementation. Addison-Wesley, 1983.
30. Alan C. Kay. The Early History of Smalltalk. ACM SIGPLAN Notices (ACM) 28 (3): 69–95, March 1993.
31. M.V. Wilkes, D.J Wheeler, S. Gill. The Preparation of Programs for an Electronic Digital Computer, with Special Reference to the EDSAC and the Use of a Library of Subroutines. Addison-Wesley, Reading, MA, 1951.
32. IBM. Preliminary Report, Specifications for the IBM Mathematical FORmula TRANslation System, FORTRAN. IBM Corporation, 1954.
33. IBM. Programmer's Reference Manual, The FORTRAN Automatic Coding System for the IBM 704 EDPM. IBM Corporation, New York, 1956.
34. A. Perlis, K. Samelson. Preliminary Report – International Algebraic Language. Communications of the ACM, Vol. 1, No. 12, pp. 8-22, 1958.
35. Department of Defense. COBOL, Initial Specifications for a Common Business Oriented Language. U.S. Department of Defense, Washington, D.C., 1960.
36. IBM. The New Programming Language. IBM UK Laboratories, 1964.
37. N. Wirth, C.A.R. Hoare. A Contribution to the Development of Algol. Communications of the ACM, Vol. 9, No. 6, pp. 413-431, 1966.
38. B. Stroustrup. Adding Classes to C: An Exercise in Language Evolution. Software – Practice and Experience, Vol. 13, pp. 139-161, 1983.
39. B. Stroustrup. The Design and Evolution of C++. Addison-Wesley, Reading, MA, 1994.
40. Jean Ichbiah. Ada: Past, Present, Future — An Interview with Jean Ichbiah, the Principal Designer of Ada. Communications of the ACM 27 (10): 990–997, October 1984.
41. Jean Ichbiah, John G. P. Barnes, Robert J. Firth, and Mike Woodger. Rationale for the Design of the Ada Programming Language. Ada Companion Series. Cambridge University Press, Cambridge, England, 1991.