Friday, 14 March 2025

History of Programming Languages

Subject

—The second half of the software development process is the implementation phase. Here, programming language is a communication medium between the programmer and the computer.
—Human languages are unpredictable but programing languages have a definite structure, which the computer translates.
—But as time went by, only having structure was not enough. Computers were being used in crucial areas where accuracy, performance and reliability were crucial.
—Thus changes were made to legacy programming languages, as per the demand of the time.
—Time period, programming demands, and languages developed are as follows:

1950s - 1960s: Early computing, list processing. Regional Assembly Language, IPL, MATIC, FORTRAN, COMTRAN

1960s - 1970s : Multiprogtamming OS, syntax derived, optimized compilers. COBOL, ALGOL, PL/I

1970s - 1980s: Program verification, data abstraction, formal semantics embedded programming. ADA, C, Prolog, Pascal

1980s - 2000s : OOP, Interactive environments, open systems. C++, SML, PERL, HTML

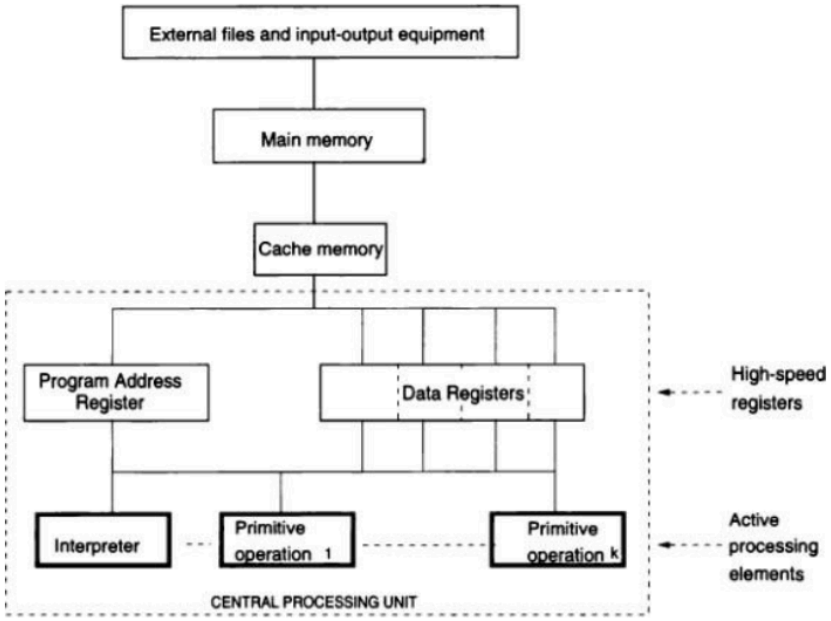
2000s+ : Data Science, E-commerce. Java, JS, XML, Python

— Even after all this progress, we do not expect a stable programming language which will take over the industry. The choice of Programming languages still remain highly dependent on nature of project.

Impact of Machine Architecture

—Early languages were designed to permit programs to run efficiently on expensive hardware.
—Therefore, the early languages had a design that translated into efficient machine code, even if the programs were hard to write.
—However, machines today are inexpensive, machine cycles are usually plentiful, but programmers are expensive. There is a greater emphasis in designing programs that are easy to write correctly even if they execute somewhat more slowly.
—In developing a programming language, the architecture of the software influences the design of a language in two ways:

1. The underlying computer on which programs written in the language will execute; and
2. The execution model, or virtual computer, that supports that language on the actual hardware.



- Hardware:

The structure of computer hardware greatly influences programming languages. Early languages like FORTRAN were designed for efficient execution on expensive hardware, prioritizing speed over ease of programming. Modern architectures, including cache memory, multi-core processors, and specialized instruction sets (e.g., RISC vs. CISC), affect how languages are optimized for execution.

- Firmware

Firmware acts as an intermediary between hardware and software, affecting language execution efficiency. It consists of low-level microcode and embedded routines that enable the functioning of machine instructions. Microprogramming

techniques allow firmware to simulate virtual architectures, enabling higher-level languages to function efficiently across different hardware platforms.

- Software

Software architecture shapes language design by defining execution models and runtime environments. Operating systems provide essential services such as memory management, input/output operations, and process scheduling, which influence language features. Additionally, multi-threading and concurrency models in programming languages are influenced by software architectures supporting parallel execution.

Role of Programming Languages

➡ There is a strong relationship between programming language and the software design methods.

➡ Older languages like FORTRAN did not support specific design methods.

➡ To understand this relationship, we need to remember that languages may follow a definite programming style called programming paradigm. But there also exist paradigm natural languages.

➡ Design methods turn into developer's guide in decomposing the system/algo. Into components that are to be coded. Different paradigms lead to different decompositions.

➡ Example: Procedural design method suggests breaking down system into modules that realize abstract operations and can be called in other procedures. An Object oriented approach guides to break system into objects and classes.

➡ If the design method and the language are same, then the design concepts can be easily mapped to the program modules. Otherwise if the design method and language clash, programming effort increases.

➡ Thus, a continuum between design method and programming language is essential.

Characteristics of a Good Programming Language

- Following are the characteristics of a good programming language:

    1. Clarity, Simplicity, Unity
    2. Orthogonality
    3. Naturalness

4. Abstraction
5. Program Verification
6. Programming environment
7. Portability
8. Cost of use

- Clarity, Simplicity
    - PL should provide the framework for thinking about the algorithm.
    - Should provide a set of concepts that can be used as primitives while building algo.
    - Syntax should be such that readability can be increased.
- Orthogonality
    - A PL is said to orthogonal if its constructs can be freely used in combination with each other.
    - For eg: A pointer should be able to point to any type of variable/data structure.
    - Makes PL easier to learn.
- Naturalness
    - Syntax of PL should naturally follow the logic of the algorithm.
    - Should provide appropriate data structures, operations, control structures for the problem to be solved.
- Abstraction
    - Abstraction = hiding the implementation details.
    - Allows programmer to design a complicated structure without hindering the implementation details.
- Verification
    - Reliability of a program written is very important. There are various methods to check it.
    - One of it is test input data, checking the output against the specifications.
    - I semantics and syntax are simple, verification is simplified.

- Environment
    - Should include well documented implementations, special editors, test packages, version control services and so on.
- Portability
    - Program should be able to port to a different platform if need arises.
- Cost of use
    - Cost of Execution:
        - Important for large production programs, executed repeatedly.
        - Methods are optimized compilers, efficient memory allocations, etc.
    - Cost of Translation
        - Time for compiling large programs should be low.
        - Should have a fast compiler rather than one which produces optimized code.

Binding Time

- Any program contains various entities such as variable, routines, subroutines, control statements and so on, which have special attributes/properties.
- Binding refers to the process of converting these identifiers into addresses.
- In programming, binding is an API.
- Binding is static if it occurs before program execution, and stays the same throughout it.
- Binding is dynamic if it occurs/changes during the course of program execution.
- There are various types of binding classes, they are as follows:

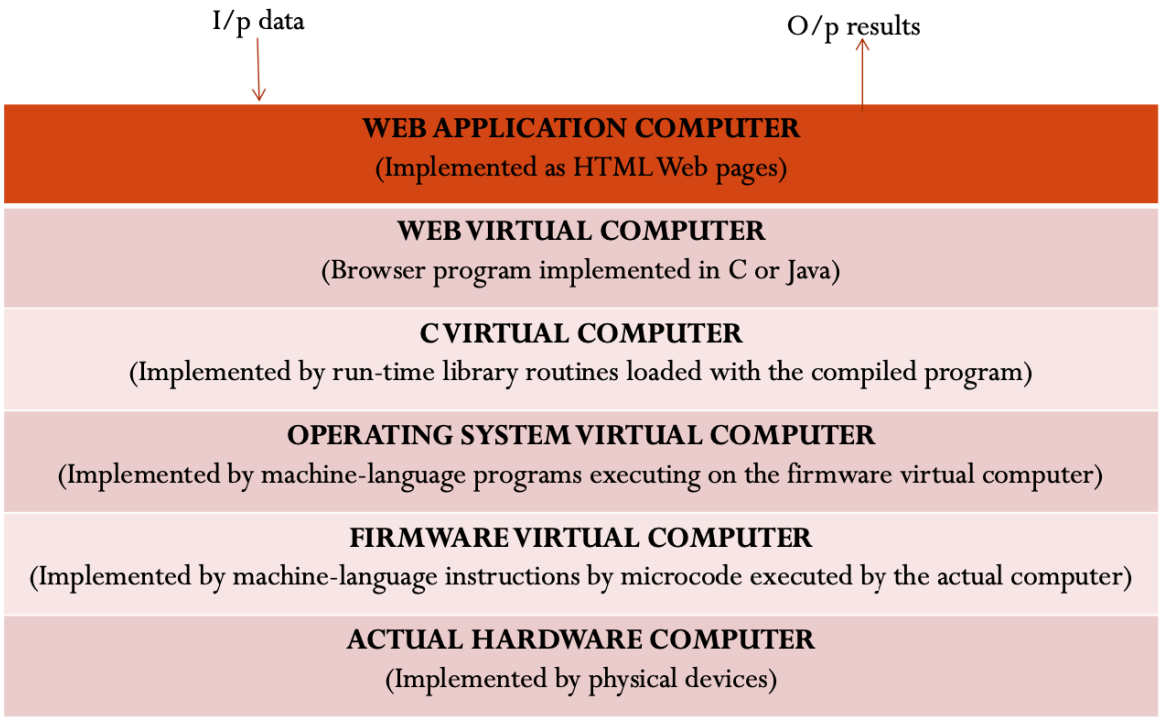
1. Run Time
    - On entry into subprogram
        - Binding of formal and actual parameters, and to storage locations
    - At arbitrary points
        - Binding of variables to values

    - Binding of names to storage locations.

2. Compile Time
    - Bindings chose by the programmer
        - Choice of variable name, types
        - Choice of program statement
    - Bindings chosen by Translator
        - Relative location of data objects
    - Bindings chosen by the linker
        - Relative location of object modules
3. Implementation Time (When compiler/interpreter is written)
    - Representation of numbers, usually determined by underlying computer.
    - Binding of statements to semantics
4. Definition Time
    - Data structure types
    - Array storage layout
    - Alternative statement forms

Computer Construction

- A computer is an integrated set of algorithms and data structures capable of storing and executing programs.
- Different ways in which a computer may be realized:
  - Hardware - Representing data structures and algorithms directly with physical device.
  - Firmware - Representing the data structures and algorithms by microprogramming a suitable hardware computer.
  - Virtual Computer - Representing data structures and algorithms in some other programming language.
  - Through a combination of these techniques or by software simulation.



Layers of virtual computers for a Web application