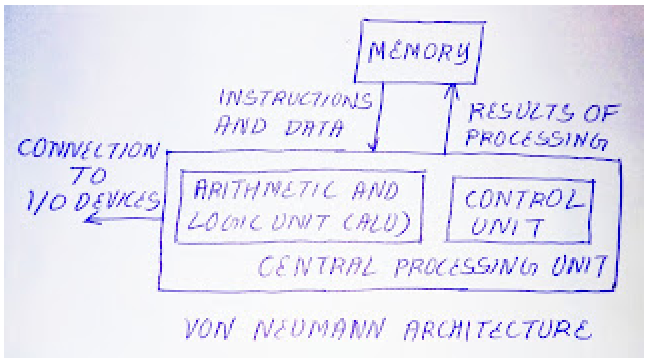
1. Computer Architecture
2. Programming Methodologies
3. Virtual Machine

1. Computer Architecture:



Computer architecture in based on Von Neumann Architecture.

A programming language is also affected by the architecture of computer.

But how ?

This is the question will see here. When we deploy/run a programs it reside in a memory and executed by the CPU. There are registers like program counter, instruction register etc. Each instruction going from memory to CPU is decided by program counter.

And program counter get instruction info from instruction register.

In this a way a program developed in languages passes trough this kind of cycle. Which affect the execution time of the language.

If system is multi user it will consume lot of time.

2. Programming Methodologies:

- Structured programming: This programming methodology also called top own design and step wise refinement. Its deficiency was incompleteness of type checking and inadequacy of control statements, which requires extensive use of go to statements. Ex. C.
- Data oriented programming: It focuses on data oriented methods emphasize data design, focusing on the use of abstract data types to solve the problem. Ex. Simula.
- Procedure oriented programming: It is the opposite of data oriented programming. Ex. C#.
- Object oriented programming: It gives support for Data abstraction, inheritance, polymorphism. Ex. C++.

3. Virtual Machine:

Virtual machines are software's on which other software's can be executed as they are executing on a physical machine.

There are two types of virtual machine:

1. Application/ process virtual machine.
2. System/hardware virtual machine.

1. Application/process virtual machine: In this we can take example of JVM. Because of JVM, Java is a platform independent language.

2. System/hardware virtual machine: In this we can take example of Virtual Box. Which gives as ability to run multiple of operating systems on a single physical machine. Here operating system is also a programming language which should be supported by Virtual Box.

The design of a programming language is influenced by various factors, including the following:

1. Problem Domain:

The problem domain for which the language is intended plays a significant role in its design. Different domains have different requirements and characteristics, which influence language features and constructs. For example, a language designed for scientific computing may prioritize numerical operations and mathematical expressions, while a language for web development may focus on handling HTTP requests and rendering web pages.

2. Target Audience:

The intended audience of a programming language, such as professional developers, beginners, or domain-specific users, affects its design. The language should be accessible and suited to the knowledge and expertise level of its target users. It may include language constructs, libraries, or tools that simplify common tasks, provide abstractions, or enforce best practices.

3. Existing Languages:

Existing programming languages serve as a source of inspiration and influence on language design. Language designers often study successful languages to understand their strengths and weaknesses and to incorporate proven concepts or innovations. They may adopt familiar syntax or semantics to make the language more approachable or leverage novel ideas to introduce unique features.

4. Paradigm and Programming Styles:

The choice of programming paradigm(s) (e.g., procedural, object-oriented, functional) and programming styles (e.g., imperative, declarative) greatly shapes the language design. Different paradigms and styles require specific language constructs and mechanisms to support their respective programming models effectively.

5. Language Goals:

The goals set by the language designers or the organization behind the language influence its design. Goals may include factors like performance, readability, expressiveness, safety, ease of use, portability, or interoperability. Balancing these goals often leads to trade-offs and decisions in language design.

6. Technical Constraints:

Technical constraints, such as hardware limitations, memory management requirements, or compatibility with existing systems or libraries, impact language design. For example, constrained environments like embedded systems may require languages with minimal runtime overhead or small memory footprints.

7. Industry Trends and Standards:

Language design is influenced by industry trends, technological advancements, and emerging standards. The need to address new computing platforms, integrate with existing technologies, or support emerging paradigms (e.g., cloud computing, mobile development, machine learning) can shape language features and tooling.

8. Community and User Feedback:

Input from the programming community and user feedback play a vital role in language design. Language designers often engage with the community to gather suggestions, identify pain points, and address usability issues. Feedback from early adopters and user communities helps shape the language’s evolution through updates, language extensions, or new versions.

9. Language Implementation:

The choice of language implementation (compiler, interpreter, runtime environment) affects design decisions. The capabilities and limitations of the implementation platform may influence language features, performance characteristics, memory management, and other implementation-specific considerations.

Language design is a complex and iterative process that involves carefully balancing various influences to create a language that meets the needs of its users, solves specific problem domains efficiently, and provides a pleasant development experience.

Related posts:

1. Sequence Control & Expression | PPL
2. PPL:Named Constants
3. Parse Tree | PPL | Prof. Jayesh Umre
4. Basic elements of Prolog
5. Loops | PPL | Prof. Jayesh Umre
6. Subprograms Parameter passing methods | PPL | Prof. Jayesh Umre
7. Programming Paradigms | PPL | Prof. Jayesh Umre
8. Subprograms Introduction | PPL | Prof. Jayesh Umre
9. Phases of Compiler | PPL | Prof. Jayesh Umre
10. Parse Tree | PPL
11. Influences on Language design | PPL | Prof. Jayesh Umre
12. Fundamentals of Subprograms | PPL | Prof. Jayesh Umre
13. Programming Paradigm
14. Language Evaluation Criteria
15. OOP in C++ | PPL
16. OOP in C# | PPL
17. OOP in Java | PPL
18. PPL: Abstraction & Encapsulation
19. PPL: Semaphores
20. PPL: Introduction to 4GL
21. PPL: Variable Initialization
22. PPL: Conditional Statements
23. PPL: Array
24. PPL: Strong Typing
25. PPL: Coroutines

26. PPL: Exception Handler in C++
27. PPL: OOP in PHP
28. PPL: Character Data Type
29. PPL: Exceptions
30. PPL: Heap based storage management
31. PPL: Primitive Data Type
32. PPL: Data types
33. Programming Environments | PPL
34. Virtual Machine | PPL
35. PPL: Local referencing environments
36. Generic Subprograms
37. Local referencing environments | PPL | Prof. Jayesh Umre
38. Generic Subprograms | PPL | Prof. Jayesh Umre
39. PPL: Java Threads
40. PPL: Loops
41. PPL: Exception Handling
42. PPL: C# Threads
43. Pointer & Reference Type | PPL
44. Scope and lifetime of variable
45. Design issues for functions
46. Parameter passing methods
47. Fundamentals of sub-programs
48. Subprograms
49. Design issues of subprogram
50. Garbage Collection
51. Issues in Language Translation
52. PPL Previous years solved papers

53. Type Checking | PPL | Prof. Jayesh Umre
54. PPL RGPV May 2018 solved paper discussion| Prof. Jayesh Umre
55. PPL Viva Voce
56. PPL RGPV June 2017 Solved paper | Prof. Jayesh Umre
57. Concurrency
58. Basic elements of Prolog
59. Introduction and overview of Logic programming
60. Application of Logic programming
61. PPL: Influences on Language Design
62. Language Evaluation Criteria PPL
63. PPL: Sequence Control & Expression
64. PPL: Programming Environments
65. PPL: Virtual Machine
66. PPL: Programming Paradigm
67. PPL: Pointer & Reference Type
68. try-catch block in C++