Lecture Notes 1

Why study programming languages?

- Increase your capacity to express ideas
- Improve ability to choose appropriate language
- Reduce time to learn new languages
- Understand the significance of language implementation
- Better utilize known languages
- Advance the collective knowledge of computing

Programming Domains

- Scientific Computing
- Business Applications
- Artificial Intelligence
- Web Software

Language Evaluation

- Language Criteria
  - Readability – Is the language easy to read and understand?
  - Writability – Is the language easy to write and express desired outcome?
  - Reliability – Does the language have features that allow the program to always perform according to specification?
- Characteristics
  - Simplicity – Does the language have a small number of basic constructs?
    - Feature Multiplicity – Having more than one way to accomplish an operation.
    - Operator Overloading – Single operator symbol has more than one meaning.
  - Orthogonality – Does the language have independent constructs that can be combined in many ways?
    - Orthogonality Example – Data types and procedures may be orthogonal because parameters and return types can be combined in many different ways.
    - Non-orthogonal Example – Language supports String variable + literal string to concatenate and return String, but literal string + String variable is not allowed:

      ```
      x = y + "Hello"; // Valid
      x = "Hello" + y; // Not valid
      ```
  - Data Types – Does the language have adequate data types and ability to define new ones?

- Syntax Design – Does the language have a syntax that is easily readable?
  - Special Words (or Keywords) – Choice of special words influences program readability.
    - Examples – `for`, `while`, `class`, etc.
  - Meaning (or Semantics) – The meaning of a construct should follow from directly from syntax whenever possible.
- Support for Abstraction – Does the language support creating layers of abstraction?
- Expressivity – How much computation can be expressed with small amount of code?
- Type Checking – Does the language check for type errors?
- Exception Handling – Does the language support the interception of run-time errors?
- Restricted Aliasing – Does the language limit some forms of aliasing (when there are two or more names to access same memory location)?
- Cost – What are the costs of a programing language?
  - Programmer Training – How difficult is the language to learn?
  - Programmer Productivity – How effective are programmers at writing applications?
  - Execution Cost – How much time/resources are required to execution program?
- Generality – Can the language be applied to a wide range of applications?
- Portability – Can a program in the language be easily ported to another target?

Influence on Language Design

- Computer Architecture – The functionality and organization of a computer system
- von Neuman Architecture – Data and program (or text) are stored in same memory with a CPU executing computation on data that is moved back and forth from memory.
  - Fetch-Decode-Execute Cycle – Cycle in which the next instruction is fetched from memory, decoded and then executed.
  - Program Counter – A register that holds the address of the current instruction
- Imperative Languages – A language in which the programmer defines a sequence of actions
- Programming Design Methodologies – The methodology used in designing a program
  - Procedure-Oriented – Focus on a set of procedures that must be completed for a task
  - Data-Oriented – Focus on the design of data and using abstract data types
  - Object-Oriented – Use of data abstraction, encapsulation, data access restriction, inheritance and dynamic method binding.

Language Categories

- Traditional Categories
  - Imperative – Computation is a sequence of actions.
  - Functional – Computation as a set of functions and function applications.
  - Logic – Computation as a verification of an assertion against a set of logical truths.
  - Object-Oriented – Computation as a set of autonomous interacting objects.
- Scripting Languages – More bound by the implementation than the language features
- Markup/Programming Hybrids – Some markup languages like HTML have some programming features that have crept in over time.

Language Design Trade-Offs

- Contradicting Criteria – Many of the language criteria directly conflict one another
  - Example – Reliability (error checking, etc.) comes at a cost of execution time.

Implementation Methods

- Compilation – Language is translated from code into machine instructions
- Pure Interpretation – Language is directly executed by an interpreter
- Hybrid Implementation Systems – Translation to easy interpret code for execution on a Virtual Machine (VM)
  - Byte Code – Intermediate format that is easily interpreted on a VM
  - Just-In-Time (JIT) Translation – Byte Code is translated into machine instructions "Just in time" for execution
- Preprocessors – Program that processes the code prior to compilation
- Traditional C Build Process
  Code → Preprocessor → Compiler → Assembler → Linker → Executable Image