جامعة أم القرى
UMM AL-QURA UNIVERSITY

CS3713T Programming Languages

L01: Preliminaries

Reviewed & Modified by: Dr. Majdah Alsharif 2025

1

Topics

- Reasons for Studying Concepts of Programming Languages
- Programming Domains
- Language Evaluation Criteria
- Influences on Language Design
- Language Categories & Design Trade-Offs
- Implementation Methods
- Programming Environments

2

2

Reasons for Studying Concepts of Programming Languages

- Increased ability to express ideas
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Better understanding of significance of implementation
- Better use of languages that are already known
- Overall advancement of computing

3

3

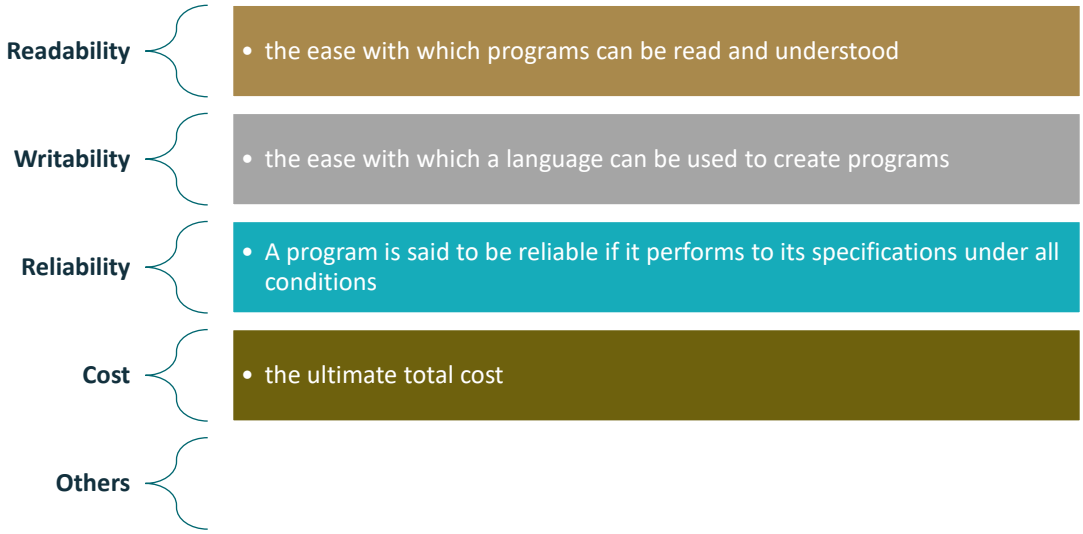
Programming Domains

Scientific applications	Business applications	Artificial intelligence	Web Software
• Large numbers of floating-point computations • use of arrays • Fortran	• Produce reports • use decimal numbers & characters • COBOL	• Symbols rather than numbers manipulated • use of linked lists • LISP	• Eclectic collection of languages: markup (e.g., HTML), scripting (e.g., PHP), general-purpose (e.g., Java)

4

4

Language Evaluation Criteria



5

5

Language Evaluation Criteria Readability

Overall simplicity	Orthogonality	Data types	Syntax Design
• A manageable set of features and constructs • Minimal feature **multiplicity** • Minimal **operator overloading**	• A relatively small set of primitive constructs can be combined in a relatively small number of ways • Every possible combination is legal	• Adequate predefined data types	• Special words • methods of forming compound statements • Form and meaning: self-descriptive constructs, meaningful keywords

6

6

Language Evaluation Criteria Writability

Simplicity and orthogonality	• Few constructs, a small number of primitives, a small set of rules for combining them
Support for abstraction	• The ability to define and use complex structures or operations in ways that allow details to be ignored
Expressivity	• A set of relatively convenient ways of specifying operations • Strength and number of operators and predefined functions

7

7

Language Evaluation Criteria Reliability

Type checking	Exception handling	Aliasing	Readability and writability
• Testing for type errors	• Intercept run-time errors and take corrective measures	• Presence of two or more distinct referencing methods for the same memory location	• A program written in a language that does not support natural ways to express the required algorithms will necessarily use unnatural approaches which are less likely to be correct

8

8

Language Evaluation Criteria
Cost

- Training programmers to use the language
- Writing programs
- Compiling programs
- Executing programs
- Language implementation system: availability of free compilers
- Reliability: poor reliability leads to high costs
- Maintaining programs

9

9

Language Evaluation Criteria
Others

Portability

The ease with which programs can be moved from one implementation to another

Generality

The applicability to a wide range of applications

Well-definedness

The completeness and precision of the language's official defining document

10

10

Influences on Language Design

- **Computer Architecture**
  - Languages are developed around the common computer architecture, known as the **von Neumann architecture**
- **Program Design Methodologies**
  - New software development methodologies (e.g., object-oriented software development) led to new programming paradigms and by extension, new programming languages

11

11

Influences on Language Design Computer Architecture

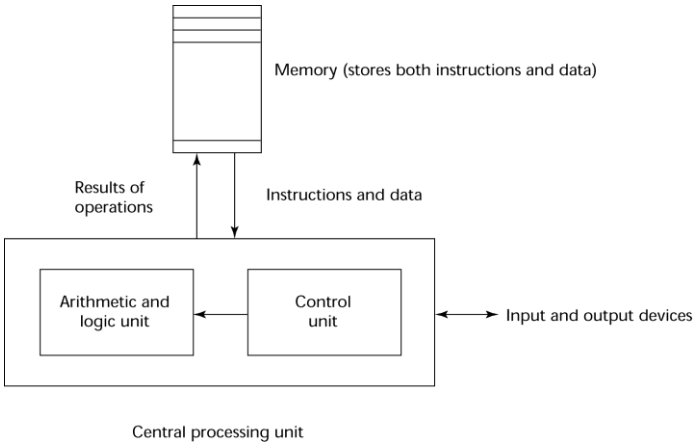
- Most of the languages of the past 60 years have been designed around the **von Neumann architecture** where:
  - Data and programs are stored in memory
  - Memory is separate from CPU
  - Instructions and data are piped from memory to CPU
- These languages are called **imperative languages**

12

12

Influences on Language Design
Computer Architecture

- **The von Neumann Architecture**



13

13

Influences on Language Design
Computer Architecture

- **The Fetch-execute-cycle (on a von Neumann architecture computer)**
  - The execution of a machine code program on a von Neumann architecture computer occurs in a process called the **fetch-execute cycle**
  - As stated earlier, programs reside in memory but are executed in the CPU
  - Each instruction to be executed must be moved from memory to the processor
  - The address of the next instruction to be executed is maintained in a register called the **program counter**

14

14

Programming Methodologies Influences

- **Late 1960s and Early 1970s**
  - intense analysis of both the software development and programming language design
  - Larger and more complex problems
  - new methodologies: top-down design and stepwise refinement
- **Late 1970s**
  - Process-oriented to data-oriented
- **Middle 1980s**
  - Object-oriented programming

15

15

Language Categories

- **Imperative**
  - Central features are variables, assignment statements, and iteration
  - Include languages that support object-oriented programming
  - Include scripting languages
  - Include the visual languages
  - Examples: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- **Functional**
  - Computations are done by applying functions to parameters
  - Examples: LISP, Scheme, ML, F#
- **Logic**
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog

16

16

Language Categories

- **Markup/programming hybrid**
  - Markup languages extended to support some programming
  - Examples: HTML, JSTL, XSLT

17

17

Language Design Trade-Offs

- **Reliability vs. cost of execution**
  - Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs
- **Readability vs. writability**
  - Example: APL provides many powerful operators allowing complex computations to be written in a compact program but at the cost of poor readability
- **Writability vs. reliability**
  - Example: C++ pointers are powerful and very flexible but are unreliable

18

18

Implementation Methods

- **Compilation**
  - Programs are translated into machine language
  - Use: Large commercial applications
- **Pure Interpretation**
  - Programs are interpreted by another program known as an interpreter
  - Use: Small programs or when efficiency is not an issue
- **Hybrid Implementation Systems**
  - A compromise between compilers and pure interpreters
  - Use: Small and medium systems when efficiency is not the first concern

19

19

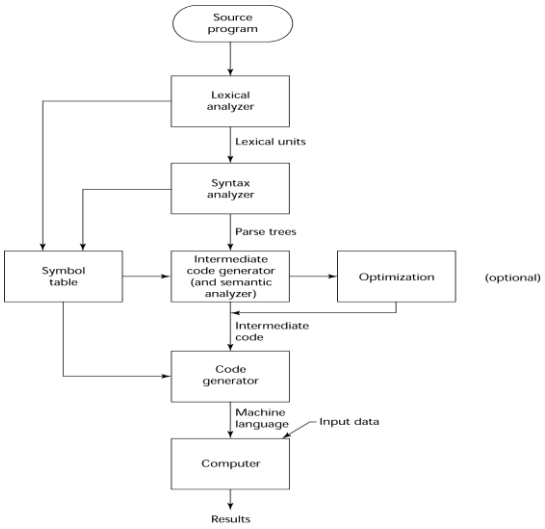
Compilation

- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution
- **Compilation process has several phases**
  - lexical analysis: converts characters in the source program into lexical units
  - syntax analysis: transforms lexical units into parse trees
  - Semantics analysis: generate intermediate code
  - code generation: machine code is generated
- **Linking & loading**: collecting system program units and linking them to a user program

20

20

Compilation
The Process



21

21

Compilation
Von Neumann Bottleneck

- Every piece of **data** and **instruction** has to pass across the **data bus** in order to move from **memory** into the **processor** (and back again)
- The **data bus** is a lot **slower** than the rate at which the processor executes **instructions**
- This is a problem called the **Von Neumann bottleneck**
- If nothing were done, the processor would spend most of its time waiting around for instructions
- It is the primary limiting factor in the speed of computers

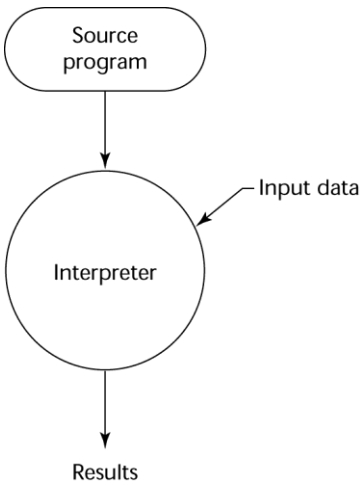
22

22

Pure Interpretation

- No translation
- Easier implementation of programs (run-time errors can easily and immediately be displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Now rare for traditional high-level languages
- Significant comeback with some Web scripting languages (e.g., JavaScript, PHP)

Pure Interpretation
Pure Interpretation Process



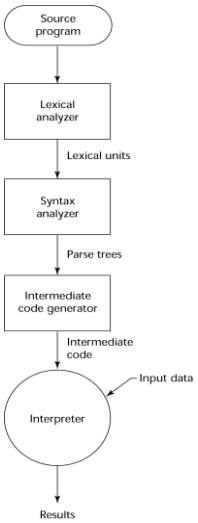
Hybrid Implementation Systems

- A compromise between compilers and pure interpreters
- A high-level language program is translated to an intermediate language that allows easy interpretation
- Faster than pure interpretation
- Examples
  - Perl programs are partially compiled to detect errors before interpretation
  - Initial implementations of Java were hybrid

25

25

Hybrid Implementation Systems
Hybrid Implementation Process



26

26

Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile the intermediate language of the subprograms into machine code when they are called
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system
- In essence, JIT systems are delayed compilers

27

Preprocessors

- A **preprocessor** is a program that processes a program just before the program is compiled
- Preprocessor instructions are embedded in programs & used to specify that the code from another file is to be included
- Example (C preprocessor):
  - `#include "myLib.h"`

  causes the preprocessor to copy the contents of myLib.h into the program at the position of the #include

28

Programming Environments

- **Programming environment** is a collection of tools used in software development such as:
  - **UNIX**
    - An older operating system and tool collection
    - Nowadays often used through a GUI (e.g., CDE, KDE, or GNOME) that runs on top of UNIX
  - **Microsoft Visual Studio.NET**
    - A large, complex visual environment
    - Used to build Web applications and non-Web applications in any .NET language
  - **NetBeans**
    - Related to Visual Studio .NET, except for applications in Java

29