

Introduction

Dr. Nguyen Hua Phung

HCMC University of Technology, Viet Nam

08, 2016

- 1 Introduction
- 2 Reasons to study Concepts of Programming Languages
- 3 Language Evaluation Criteria
- 4 Language Design
- 5 Implementation Methods

- **My name:** Nguyen Hua Phung
- **Email:** nhphung@hcmut.edu.vn
- **Office hour:** 10:00-11:30 Wednesday at CS1
- **Office hour:** 16:00-17:30 Thursday at Room 703-CS2
- **Sakai:** <https://elearning-cse.hcmut.edu.vn>

- Programming Languages: Principles And Paradigms, Maurizio Gabbrielli and Simone Martini, Springer, 2006.
- Programming Languages: Principles and Practices, Kenneth C. Loudon, Thomson Brooks/Cole, 2003.
- Ngon Ngu Lap Trinh: Cac nguyen ly va mo hinh, Cao Hoang Tru, 2004.

- Tutorial/Lab/Online: 10%
- Assignment: 30%
- Midterm: 20%
- Final: 40%

Note: Assignment is calculated by the following formula:

$$\text{Assignment} = 2 * \frac{A * B}{A + B}$$

where A is from some given project and B is from some questions in midterm or final

After complete this subject, students are able to:

- describe formally lexicon and grammar of a programming language
- describe and explain some mechanism of a programming language
- implement a interpreter/compiler for a simple programming language

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

- Scientific Applications
Fortran, ALGOL 60
- Business Applications
COBOL
- Artificial Intelligence
LISP, Prolog
- Systems Programming
PL/S, BLISS, Extended ALGOL, and C
- Web Software
XHTML, JavaScript, PHP

- Simplicity
- Orthogonality
- Support of abstraction (Control, Data)
- Safety
- ...

- Readability
- Writability
- Reliability
- Cost

- Computer Architecture
 - Von Neumann
- Programming Methodologies
 - Imperative
 - Machine-based
 - Procedural
 - Declarative
 - Logic
 - Functional
 - Constraint
 - Query-based
 - Object-Oriented
 - ...

- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
 - Data and programs stored in memory
 - Memory is separate from CPU
 - Instructions and data are piped from memory to CPU
 - Basis for imperative languages
 - Variables model memory cells
 - Assignment statements model writing to memory cell
 - Iteration is efficient

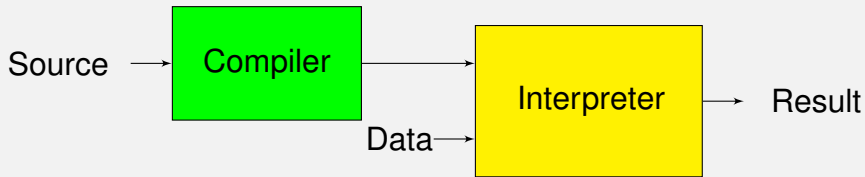
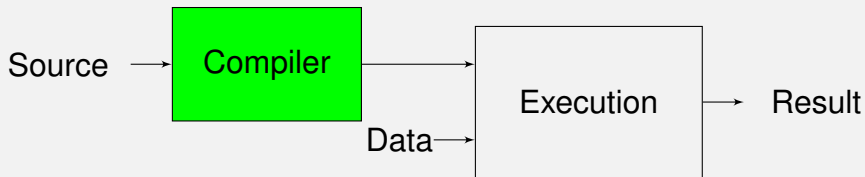
- 1950s and early 1960s: Simple applications; worry about machine efficiency
- Late 1960s: Efficiency became important; readability, better control structures
 - Structured programming
 - Top-down design and step-wise refinement
- Late 1970s: Process-oriented to data-oriented
 - data abstraction
- Middle 1980s: Object-oriented programming
 - Data abstraction + inheritance + polymorphism

- Imperative (C, Pascal)
 - Central features are variables, assignment statements, and iteration
- Functional (LISP, Scheme, Haskell, Ocaml, Scala)
 - Main means of making computations is by applying functions to given parameters
- Logic (Prolog)
 - Rule-based (rules are specified in no particular order)
- Object-oriented (Java, C++, Scala)
 - Data abstraction, inheritance, late binding
- Markup (XHTML, XML)
 - New; not a programming per se, but used to specify the layout of information in Web documents

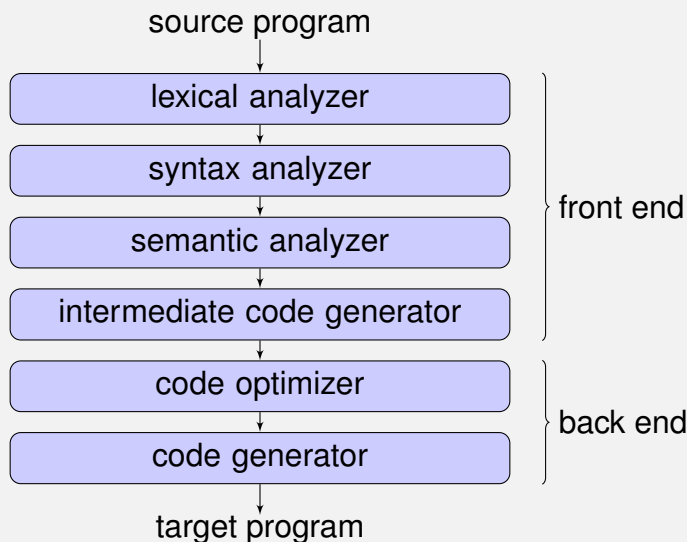
- Reliability vs. cost of execution
 - Conflicting criteria
 - Example: Java demands all references to array elements be checked for proper indexing but that leads to increased execution costs
- Readability vs. writability
 - Another conflicting criteria
 - Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability
- Writability (flexibility) vs. reliability
 - Another conflicting criteria
 - Example: C++ pointers are powerful and very flexible but not reliably used

- **Compilation**
Programs are entirely translated into machine language and then executed
- **Pure Interpretation**
Programs are translated and executed line-by-line
- **Hybrid Implementation Systems**
A compromise between compilers and pure interpreters
- **Just-in-time Compiler**
A compiler inside an interpreter compiles just hot methods

Implementation Methods



Compilation Phases



- Preprocessor
- Assembler
- Linker
- Loader
- Debugger
- Editor

What are still in your mind?