PLC 1: Language design, implementation, evaluation

- Language categories / paradigms
- Influences on language design - brief history
- Programming domains
- Language implementation
- Language evaluation criteria

Language categories / paradigms

Imperative languages
- Instr. transform the contents of memory cells step by step
- oriented towards von Neumanns computer arch. (fig 1.1)

Functional languages
- specify and apply functions similar to mathematics
- abstracts away from the computer

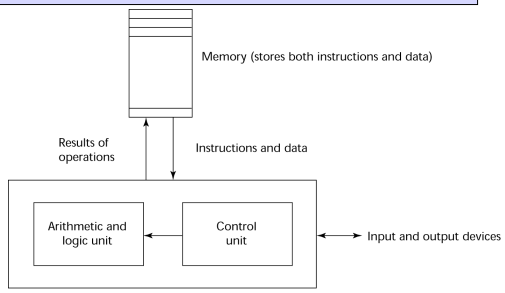
Logic programming languages
- specify ground truths (facts) and their relation
- use logical inference rules to deduct new facts

Object oriented languages
- the fundamental concept is ”objects” which ”communicate” with each other
- often an imperative language, basically

Influences on Language Design

- Computer architecture: Von Neumann
- We use imperative languages, at least in part, because we use von Neumann machines (next slide)
  - Data and programs stored in same 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 piping
    - Iteration is efficient

Von Neumann Architecture



The von Neumann Architecture

- Fetch-execute-cycle (on a von Neumann architecture computer)

```
initialize the program counter
repeat forever
  fetch the instruction pointed by the counter
  increment the counter
  decode the instruction
  execute the instruction
end repeat
```

- “von Neumann Bottleneck”

Influences on Language Design

- Programming methodologies
  - 1950s and early 1960s: Simple applications; worry about machine efficiency
  - Late 1960s: People 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
  - 1980s: Functional programming, Logic programming
  - Middle 1980s: Object-oriented programming
  - Concurrency (“multicore”)

Programming domains

Scientific applications → efficiency, fast floating-point arithmetic (first language: FORTRAN)

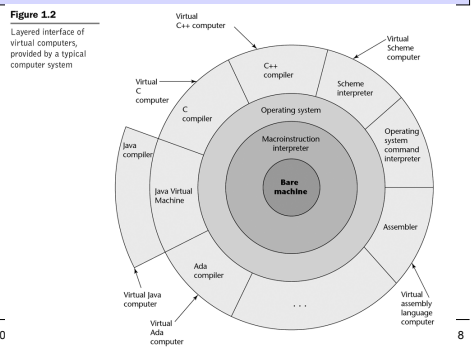
Business applications → decimal arithmetic, producing elaborate reports (first language: COBOL)

Artificial intelligence → symbolic computations, lists of data (first language: LISP, later: PROLOG)

Systems programming → efficiency, direct access to resources of the operating system (early language: PL/I (PL/S), later: C)

Miscellaneous (markup languages, scripting languages, web software etc)

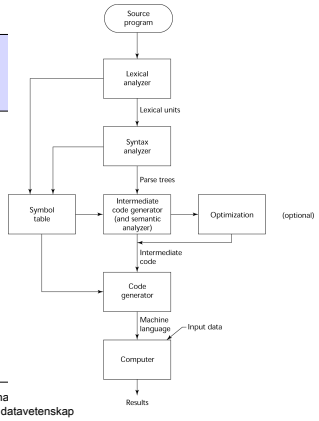
The context of prog. languages



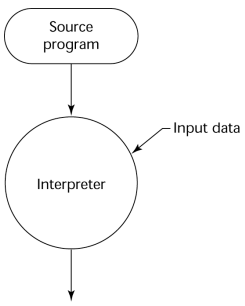
Implementation Methods

- Compilation
  - Translate high-level program to machine code
  - Slow translation
  - Fast execution
- Pure interpretation
  - No translation
  - Slow execution
  - Common => Rare => now gaining ground againg
- Hybrid implementation systems
  - Small translation cost
  - Medium execution speed
  - Intermediate code, virtual machine

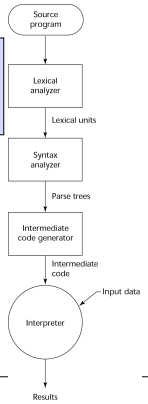
Compilation Process



Pure Interpretation



Hybrid Implementation System



Programming Environments

- The collection of tools used in software development
- UNIX
  - An older operating system and tool collection
- Borland JBuilder
  - An integrated development environment for Java
- Microsoft Visual Studio.NET
  - A large, complex visual environment
  - Used to program in C#, VB.NET, Jscript, J#, F#
  - Used to build web appplications and non-web aaplications in any .NET language
- NetBeans
  - Related to Visual Studio .NET, except for Web applications in Java

Language evaluation criteria

Is a language /language concept good?

Readability
- How easy is it to understand what the program does?
- Decisive factor for software maintenance
- Has influence on all other criteria

Writeability
- How easy is it to write a program? To formulate / express the problem and its solution?
- Writing programs is the basic motivation for developing programming languages
- A language can have different writeability for different application areas
- No writeability without readability

Language evaluation criteria (2)

Reliability
- Are programs doing what they are supposed to do?
- No reliability without readbility and writeability
- The most important criterion for some applications

Costs, which can be caused by
- Training programmers, writing, using and maintaining programs
- Implement the language, compile programs
- Execute programs (efficiency)

Characteristics that have a significant effect on readability

Simplicity
- The number of concepts /features.
- Feature multiplicity. How many ways can the same thing be expressed?
- Sensible use of operator overloading.

Orthogonality
- A relatively small set of primitive constructs can be combined in a relatively small number of ways. All combinations are legal. Increases readability (but not when used in excess)

Control statements
- Good structure is more important than maximum flexibility.

Characteristics that have a significant effect on readability (2)

Data types and data structures
- Adequate facilities for defining (your own) datatypes a must (orthogonality!)
- ”Natural” representation of data

Syntax
- what identifiers are allowed?
- The syntax of the language should be natural an easy to interpret (correctly) (form and meaning harmonize)

Characteristics that have a significant effect on writeability

Simplicity and orthogonality
- Learn a few fundamental concepts and rules for combining them and you know the language!

Support for abstraction
- Concepts that allow you to focus on different levels
- Possible to decompose problems
- Both process and data abstraction are important

Expressivity
- Powerful concepts/features increases writeability (but may decrease readability!)

Features that have a significant effect on reliability

Type checking
- type errors indicate bad or sloppy programming style
- preferably compile-time type checking

Exception handling
- simple and powerful means for detecting and handling errors
- recovery after taking corrective measures

Aliasing, pointers / references
- what are they allowed to refer to?
- aliases - a dangerous feature
- explicit deallocation makes certain errors possible

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 definition

Simple answers are missing!!!

The criteria sometimes conflicts with one another. To find the best compromise you have to weigh different perspectives.

Examples:
- simplicity (readability) vs expressivity (writeability)
- reliability vs efficiency
- flexibility vs security