Concepts of Programming Languages

1st Week

Basics

Prof. Dr. Peter Heusch

Why study Programming Languages?

- “The limits of my language mean the limits of my world” – Ludwig Wittgenstein, philosopher
- We can only think about those things that we are able to talk about
- Knowing another programming language can extend the limits of your language and hence the limits of your world
- Knowing other programming paradigms can considerably speed up learning new languages

Further Reasons

- Knowing a broad range of programming languages can help:
  - Selecting the most appropriate language for a certain problem
  - Get a better understanding for implementation issues
- Last not least knowledge of existing languages helps to reengineer existing systems:
  - Web access for COBOL Programs
  - Encapsulate FORTRAN Programs in Java

What is a Programming Language? - From a German Job Offer

Internet-, Intranettechnik,
Content-Management-System (CMS) TYPO3,
Content-Management-Systeme (CMS) einsetzen,
Datenbank MySQL,
Versionsverwaltungsprogramm CVS,
Versionsverwaltungsprogramm Subversion,
EDV-Anwender-Training,
Programmieren,
Software testen,
Programmiersprache CSS,
Programmiersprachen HTML, XML, XHTML,
Programmiersprache Ajax
Programmiersprache PHP,
PHP-Entwicklungsumgebung Zend Studio,
Analyse- und Problemlösefähigkeit,
Sorgfalt/Genauigkeit,
Teamfähigkeit,
Selbständiges Arbeiten,
Motivation/ Leistungsbereitschaft

The Wikipedia Definition

A programming language is an artificial language designed to express computations that can be performed by a machine, particularly a computer. Programming languages can be used to create programs that control the behavior of a machine, to express algorithms precisely, or as a mode of human communication.

Programming Language Domains

- General Purpose:
  Java, C, C++, Smalltalk
- System Programming:
  Assembler, C, C++
- Business: COBOL,
  ABAP, Adabas
- Scientific:
  Fortran, Matlab
- Symbolic Computation
  LISP, Scheme, Prolog

- Scripting:
  Perl, Python, Bash
- Web Programming:
  PHP, JavaScript

- Grey area:
  - Markup Languages:
    TeX, LaTeX, XML
  - Database:
    SQL

Language Selection Criteria

- Readability: Is it possible to easily read programs written in this language
- Writability: Is it possible to easily write programs in this language
- Reliability: How often do programs fail and how hard is it to detect failures
- Cost: What is the cost of writing and running a program:
  - During the initial development
  - During the production (and modification)

Readability

- Simple Syntax:
  - One way to express one concept
  - One possibility to perform a single task
- Orthogonality:
  - Few features, that can be combined in many ways
- Control statements:
  - Complete set of control statements: if, for, while...
- Datatypes:
  - Facilities to aggregate data in meaningful ways

Writability

- Simplicity and Orthogonality:
  - Few Features that can be combined in many ways
- Support for Abstraction:
  - Can complex things be represented in such a way that the „user“ is not concerned with details
- Expressivity:
  - Can specifications be translated into the language without „bending“ them
  - Example: The alternate for loop in Java

Reliability

- Typing:
  - Does the language check types
- Exception handling:
  - Are errors detected and how can I handle them
- Aliasing:
  - Is it possible to reference the same variable via two different references
- Readability and Writability:
  - Does the language support expressing the problem in a „natural“ way

Cost

- How much does it cost to learn the language
- How much does it cost to set up the IDE
- How fast can I transform a given specification into a running program
  - The specification (task) heavily influences cost!
  - Easy to make an expert system in Prolog
  - Hard to make a web app in Prolog
- How easy is it to maintain programs
- How much CPU time do my programs need
- How reliable are my programs? Unreliable programs have high maintenance cost

Other criteria

- Portability:
  - Do I need to run my programs in different environments?
- Generality:
  - Does my language support a lot of different problem domains?
- Unambiguousness:
  - Does the same program always perform the same task, irrespective of the compiler

External Influences

- Computer architecture:
  - Von Neumann Architecture
  - Harvard Architecture
- Programming Methodologies:
  - Procedural Programming
  - Object Oriented Programming
  - Functional Programming
- Fashion:
  - Programming Languages Hype

The Von Neumann Architecture

- Predominant in the area of general computers
- Program and data are stored in the same main memory
  - Program is data (e.g. on disk)
  - Data is program (Java Bytecode)
- Memory is separate from CPU
- Basis of imperative programming languages:
  - Variables can be modified
  - Repeated execution in loops is possible

The Harvard Architecture

- Predominant in the area of embedded computers
- Program and data are stored in different areas of memory:
  - Program is stored readonly e.g. inside Flash-ROM
  - Data is stored modifiable inside RAM
- Memory often integrated into CPU, but RAM is very small, e.g. Atmel ATMega32: 32 KBytes of ROM, but only 2 KBytes of RAM
- Supports imperative programming languages

Programming Methodologies History

- 50s: Simple tasks, main concern is efficiency, expensive machines, cheap programmers
- 60s: Beginning of the software crisis, first attempts to structured programming:
- 70s: Procedural programming was mainstream, but initial development of OOP languages
- 80s: OOP languages become mainstream
- Other methodologies are niche players:
  - Functional programming
  - Logical programming

Language Categories

- Imperative Languages:
  - Central: variable, assignment, iteration
  - Samples: Pascal, C, Fortran
- Functional Languages:
  - Central: Apply functions to parameters
  - Samples: LISP, Scheme
- Logical Languages:
  - Central: Specify rules for solutions
  - Sample: Prolog
- Object Oriented Languages:
  - Central: Combine procedures and data into objects
  - Samples: Java, C++, Smalltalk

More Language (?) Categories

- Markup Languages:
  - Central: Not designed to write programs but to structure information
  - Samples: TeX, XML, HTML
- Query Languages:
  - Central: Not designed to write programs, but to retrieve and manipulate (up to a point) data
  - Samples: SQL

Design Tradeoffs

- There is no optimal language for all purposes
- Language designers must sometimes make decisions:
  - Speed vs. Reliability: Array element access in Java and C/C++
  - Writability vs. Readability: Source code length in APL and OCCAM
  - Flexibility vs. Reliability: Pointers make C and C++ extremely flexible but are also a main source of errors

Implementation Techniques

- Compiled Language:
  - Separation between development and runtime environment: Java, C, C++
  - Compiler output is byte code or machine code
- Interpreted Languages:
  - Single environment for development and runtime: Python, Perl, PHP
  - Intermediate format may be stored for efficiency
- Multi-Tier-Approach:
  - Combination of compilation and interpretation
  - Compile source code to byte code to machine code

Compilation

- Compilation is the transformation of source code to machine(in)dependent code
- Translation is slower, but execution is faster
- Compilation is generally done in phases:
  - Lexical analysis: Convert the character stream into a token stream
  - Syntactic analysis: Convert the token stream into a syntax tree
  - Semantic analysis: Check for cross dependencies
  - Code generation: Generate final code

Pure Interpretation

- The source code is immediately interpreted
- Allows for entering expressions at runtime
- Translation is quite fast, often because typing and variables are not thoroughly checked
- Execution is somewhat slower than for compiled languages
- Often used for „small“ and scripting languages because implementation is easy

Summary

- Defined „Programming Language“
- Named criteria for selecting a specific language for a project
- Discussed external influences on programming language design
- Contrasted the different programming paradigms
- Looked at compilation vs. interpretation