Lecture 1

Programming II

Concepts of programming languages

Iztok Savnik, FAMNIT

February, 2024.

Contents

- Course organization
- History of programming languages
- Programming models
- Concepts of programming languages
- Meta-Language ML
- Method of the course

Course organization

- Lectures
  - Iztok Savnik
- Exercises
  - Murat Elhüseyni
- Written exam
  - 4-5 exercises from tutorials and lectures
  - 90 min
- Homework
  - Writing programs in Ocaml
  - 3 homework assignments
- Oral exam
  - Can be approached after successful written exam and homework
  - Three questions from the course material

Grading

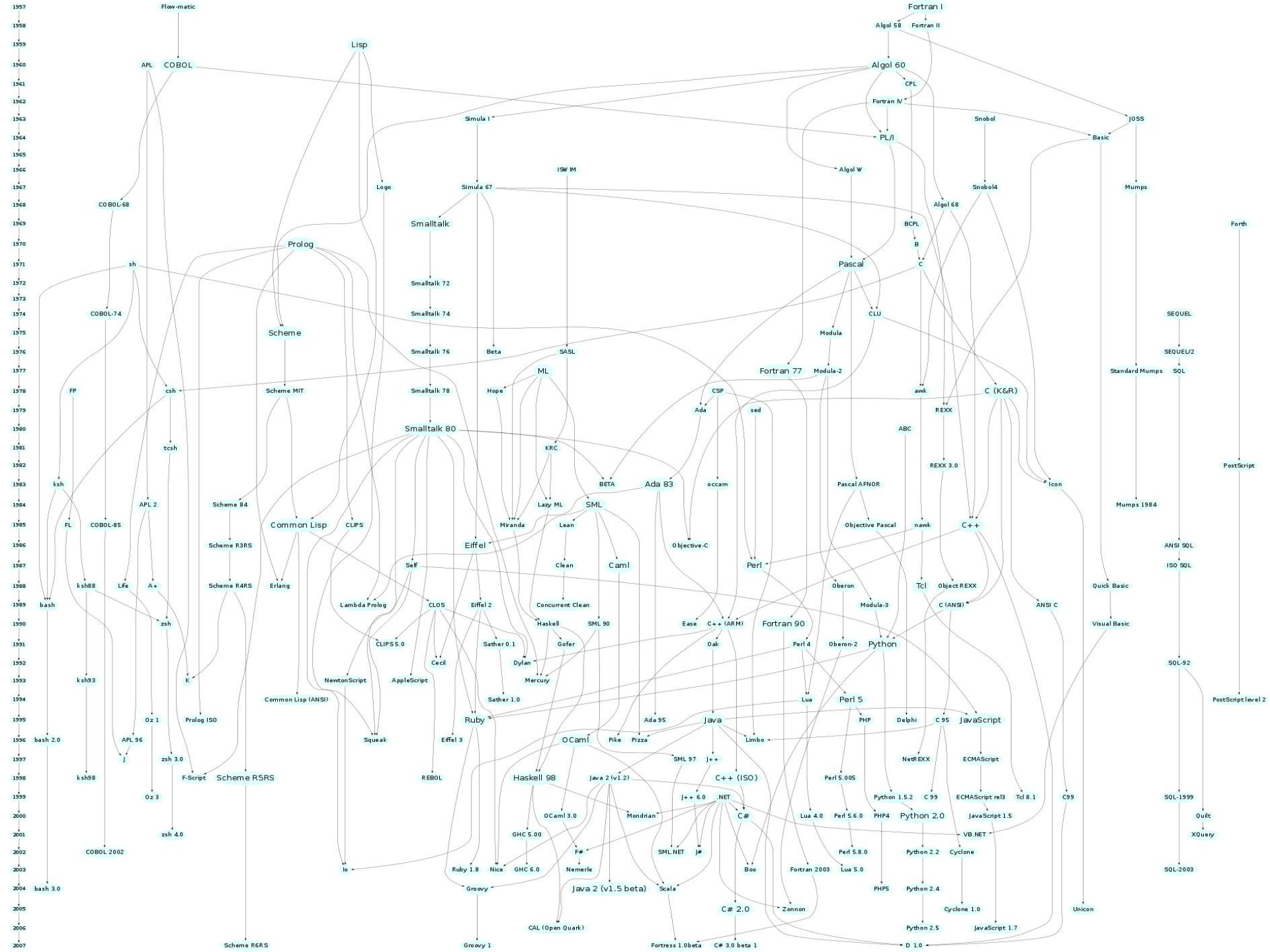
1) Written exam – 40%
2) Homeworks – 40%
3) Oral exam – 20%

All parts must be positive (>50%).

Oral exam can be approach after (1) and (2) are graded positive.

Literature

1) John Mitchell, Concepts in Programming Languages, Cambridge Univrsity Press, 2003.
2) Michael L. Scott, Programming Language Pragmatics (3rd ed.), Elsevier, 2009.
3) Iztok Savnik, Transparencies of Programming 2 Lectures, 2023.


1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
Flow-matic
Fortran I
Algol 58
Fortran II
Lisp
APL
COBOL
Algol 60
CPL
Fortran IV
Simula I
Snobol
JOSS
PL/I
Basic
Algol W
ISWIM
Logo
Simula 67
Snobol4
Mumps
Forth
COBOL-68
Algol 68
Smalltalk
BCPL
B
Prolog
C
Pascal
sh
Smalltalk 72
Smalltalk 74
COBOL-74
CLU
SEQUEL
Modula
Scheme
Smalltalk 76
Beta
SASL
SEQUEL/2
Fortran 77
Modula-2
Standard Mumps
SQL
FP
csh
Scheme MIT
Smalltalk 78
ML
Hope
CSP
awk
C (K&R)
Ada
sed
REXX
Smalltalk 80
tcsh
ABC
KRC
REXX 3.0
PostScript
ksh
BETA
Ada 83
occam
Pascal AFNOR
Icon
APL 2
Scheme 84
Lazy ML
SML
Mumps 1984
FL
COBOL-85
Common Lisp
CLIPS
Miranda
Lean
Objective Pascal
nawk
C++
Eiffel
ANSI SQL
Scheme R3RS
Objective-C
Self
Clean
Caml
Perl
ISO SQL
ksh88
Life
A+
Scheme R4RS
Erlang
Oberon
Tcl
Object REXX
Quick Basic
bash
Lambda Prolog
CLOS
Eiffel 2
Concurrent Clean
C (ANSI)
ANSI C
zsh
Haskell
SML 90
Ease
C++ (ARM)
Modula-3
Fortran 90
Visual Basic
CLIPS 5.0
Sather 0.1
Gofer
Oak
Oberon-2
Python
ksh93
Cecil
Dylan
Perl 4
SQL-92
K
NewtonScript
AppleScript
Mercury
Common Lisp (ANSI)
Sather 1.0
Lua
Perl 5
PostScript level 2
Ruby
Ada 95
Java
PHP
Delphi
C 95
JavaScript
0z 1
Prolog ISO
Squeak
Pike
Pizza
Limbo
OCaml
bash 2.0
APL 96
J
Eiffel 3
SML 97
J++
NetREXX
ECMAScript
zsh 3.0
ksh98
F-Script
Scheme R5RS
REBOL
Haskell 98
Java 2 (v1.2)
C++ (ISO)
Perl 5.005
Tcl 8.1
C99
0z 3
J++ 6.0
.NET
Python 1.5.2
C 99
ECMAScript rel3
SQL-1999
OCaml 3.0
Mondrian
C#
Lua 4.0
Perl 5.6.0
PHP4
Python 2.0
JavaScript 1.5
Quilt
zsh 4.0
GHC 5.00
VB.NET
XQuery
COBOL 2002
F#
SML.NET
J#
Perl 5.8.0
Python 2.2
Cyclone
Io
Ruby 1.8
Nice
GHC 6.0
Nemerle
Boo
Fortran 2003
Lua 5.0
SQL-2003
bash 3.0
Groovy
Java 2 (v1.5 beta)
Scala
PHP5
Python 2.4
C# 2.0
Zonnon
Cyclone 1.0
Unicon
CAL (Open Quark)
Python 2.5
JavaScript 1.7
Scheme R6RS
Groovy 1
Fortress 1.0beta
C# 3.0 beta 1
D 1.0

Contents

- Course organization
- History of programming languages
- Models of programming languages
- Concepts of programming languages
- Meta-Language ML
- Method of the course

Charles Babbage

- The origins of the digital programmable computer
  - English mathematician, philosopher, inventor and mechanical engineer.
  - In 1837 he has designed Analytical engine, the first mechanical computer that led to the design of more complex electronic computers.
- The analytical engine
  - Decimal system
  - Mechanical computation of arithmetic operations
  - Branching
  - Loops were the results of the previous computation are taken as an input to the following iterations.
- Turing-complete machine

Konrad Zuse

- The first electro-mechanical computer
  - German civil engineer and inventor Konrad Zuse
  - Z1, Z2, Z3, 1936-1945, totally independent
  - Floating-point mechanical digital calculator Z1
    - Series of the calculations of the arithmetical operations
  - Z2 = Z1 + Using telephone relays for the memory
  - Z3 = Improved Z2 (22-bit FP arithmetic)
- First programming language Plankalkuel
  - Binary system
  - Loops
  - No branching
    - Despite no branching, shown to be Turing-complete
  - Sequence of arithmetical operations

ENIAC

- First general-purpose electronic computer.
  - 1943 Uni. of Pennsylvania (J. Mauchly and J.P. Eckert)
  - ENIAC (Electronic Numerical Integrator and Computer)
  - 1000x faster than the electro-mechanical computers
  - Programming
    - 10 digit arithmetic, accumulators, general bus
    - Complex functions, branching, loops, procedures
- UNIVAC developed from ENIAC
  - First commercial computer

Electronic computers

- Mark 1, Mark 2, Mark 3
  - ASCC (Automatic Sequence Controlled Calculator)
  - Developed by Harvard University in 1944
  - Original concept presented to IBM by H.Aiken in 1937
  - Following Charles Babbage
  - Design, architecture
    - Decimal, 60 sets of 24 switches for data input, 72 numbers
    - ~2000 counters, 72 adding machines,
    - Punched paper tape, program, data
    - Instructions, execute current, read next
    - Loops, paper tape attached to the beginning
    - Branching, manual, later electronic (1946)
    - 800km wire, 5 tons, 5KW power
-

Electronic computers

- von Neumann architecture
  - Princeton arhitecture, 1945
  - First Draft of a Report on the EDVAC
    - Descendant of ENIAC, Pensilvania University, 1944
    - U.S. Army's Ballistics Research Laboratory
    - Binary, 44-bit words, 5.6KB memory, 5,937 vacuum tubes, 12,000 diodes, power 56 kW, 45.5 m², (+) 0.864 ms, (*) 2.9 ms
    - von Neumann contributed as a consultant; he wrote "First Draft of a Report on the EDVAC", 1945.
  - Principal abstraction, representation of the basic function of computers
    - central processing unit
    - main memory and
    - input/output devices

First compiler

- A-0 programming language
  - Algorithmic Language version 0
  - Developed on UNIVAC I computer
  - Grace Hopper in 1951, 1952
- Program in A-0 language was a sequence of subroutine calls together with the parameters
- The compiler was merely a linker and a loader
- Followed by the A-1, A-2, A-3
  - Commercal names: ARITH-MATIC (A-1), MATH-MATIC (AT-3), FLOW-MATIC (B-0)

Fortran

- First complete compiler
- John W. Backus, 1953, IBM
  - IBM 704 mainframe
- Mathematical **FOR**mula **TRAN**slating System
  - IBM 360
- Constructs of Fortran
  - DO, GOTO, IF, SUBRUTINE, CALL
- Numerical processing
- Followed by the imperative familliy
  - C, Pascal, Modula, Ada, Java
- Fortran II, Fortran 77, Fortran III, Fortran 90, Fortran 95, Fortran 2003, ...

FLOW-MATIC

- Bussiness Language version 0, or B-0.
  - UNIVAC I at Remington Rand
  - Grace Hopper in 1955.
  - Business does not like formulas, therefore based on English language
- Hopper team developed compiler by 1959.
  - First language that made a clear distinction between the data definition section and the code section
- Strong influence on the development of the programming language COBOL.

Information Processing Language

- Information Processing Language (IPL)
  - Allen Newell, Cliff Shaw and Herbert Simon
  - Rand Corporation in 1956.
  - Assembly language - operations manipulate lists
- IPL computer
  - Set of symbols, set of cells, set of primitive functions
  - Functions defined on the cells and lists
  - primitive PL run-time environment
- AI language
  - Abstract and suitable for expressing AI programs
  - Programs are not efficient

Algol

C.A.R. Hoare: "Here is a language so far ahead of its time that it was not only an improvement on its predecessors but also on nearly all its successors."

- Developed in ETH Zuerich, 1958.
  - Team of computer scientists from Europe and US
    - Allan J. Perlis, Peter Naur, John McCarthy, and others
  - Originally: Algebraic Language, or, IAL
  - Avoided problems perceived in FORTRAN
- Contributions
  - John Backus developed BNF to specify ALGOL 58
  - Formal semantics, two parameter passing methods: cbv and cbr, code blocks, nested functions, lexical scope
  - Imperative effect of cbr to lambda calculus studied
- Influence on:
  - Pascal, C, Modula, Java, C++

Lisp

- Developed by John McCarthy at MIT, 1958
- Lisp is based on lambda calculus
  - Alonzo Church in 1936, lambda calculus used in paper on undecidability
  - Strongly influenced by IPL
  - Lists used for the representation of programs and data
- New concepts
  - Automatic storage management, dynamic typing, self-hosting compiler, and others.
- AI language
  - Second-oldest programming language that is still in use
- Common Lisp and Scheme.

Cobol

- Designed by CODASYL, 1959
  - Conference/Committee on Data Systems Languages
  - Previous work of Grace Murray Hopper
- Programming language for business applications
  - Imperative and procedural
  - Since 2002, object-oriented
- A compiled English-like programming language
  - Awkward syntax
  - Cobol programs work today!
  - Syntax has been changed

Pascal

- Niklaus Wirth, ETH Zürich, 1970
- Very popular in Europa
- Structured programming language
  - Algol family
  - Data structures, pointers, arrays, variable records, ……
- Many implementations
  - VAX Pascal, Turbo Pascal, …
  - Delphi: a nice programming environment
- Modula-2, Modula-3, Oberon

Programming Language C

- Dennis Ritchie, Bell Laboratories, 1972
  - C designed for Unix
  - Language B, based on BCPL
- Language constructs
  - Algol family
  - Still one of best languages (C++ =? C + object illusion)
  - Close to hardware, system programming
- Arrays and references are closely related
  - E1[E2] = *((E1)+(E2))
  - Adress arithmetic
- Ritchie wrote:
  - “C is quirky, flawed, and a tremendous success.”

Contents

- Course organization
- History of programming languages
- Programming model
- Concepts of programming languages
- Meta-Language ML
- Method and aims of the course

Programming models

- Functional
- Imperative
- Object-oriented
- Modular
- Script
- Logic

Imperative languages

- Rooted in assembly language
  - Instructions, procedures, loops, branching
  - Program is a sequence of instructions
  - The outcome is built in the execution of instructions
  - Instruction changes the contents of the main memory
- Abstract syntax added
  - Variables, IF, LOOP, FOR, PROCEDURE, FUNCTION
- Fortran
  - John Backus, 1956
- Pascal, C, Ada, Modula

Functional languages

- Rooted in logic
  - Lambda calculus, 1930, Alonzo Church
- LISP
  - John McCarthy, 1958
  - Implementation of lambda calculus
- Meta-Language
  - ML, Rob Milner, 1970
- Haskel, Ocaml, Ruby, Scala
- More appropriate for teaching?
  - Strong typing
  - Program is proof
  - Elements of declarative programming

Object-Oriented Languages

- First OO language is Simula
  - 1960, Norwegian Computing Center
  - Simulation programming environment
  - It included everything recent OO languages have
- Smalltalk is one of the most famous OO languages
  - Adele Goldberg, Xerox, Palo Alto, 1970
  - Everything is object!
  - Every object belongs to one class
  - Class is prototype + class is object!
  - Complex inheritance hierarchies
- C++, Java, Objective C, Eiffel, Ruby, Scala...
- Java
  - J.Gosling, B.Joy, G.Steele, G.Bracha
    The Java Language Specification
    https://docs.oracle.com/javase/specs/

Modular Languages

- In imperative programming languages
  - Program is split into multiple program units or modules.
  - A modul gathers code for the implementation of some conceptual entity or physical object
    - Screen, driver, device, data structure, algorithm, etc.
  - Module includes an interface and an implementation
  - Controled module access and use from other modules
- In functional languages a module is an ADT
  - Abstract data type (ADT)
  - Module includes the definition of
    - Common abstract data structure and
    - Set of operations for working with abstract data structures.
  - Interface/implementation ≡ Signature/structure
- Modules are used in most programming languages
  - ML, C, Pascal, Modula 2, Scala, Go, Erlang, Perl, Phyton, itd.

Modular Languages

- In imperative programming languages
  - Program is split into multiple program units or modules.
  - A modul gathers code for the implementation of some conceptual entity or physical object
    - Screen, driver, device, data structure, algorithm, etc.
  - Module includes an interface and an implementation
    - Controled module access and use from other modules
- In functional languages a module is an ADT
  - Abstract data type (ADT)
    - Common abstract data structure and
    - Set of operations for working with abstract data structures.
  - Interface/implementation ≡ Signature/structure
- Modules are used in most PLs
  - ML, C, Pascal, Modula 2, Scala, Go, Erlang, Perl, Phyton, itd.

Script languages

- Programming languages closely coupled with an existing system.
  - Environments of script languages: editor, Web server, operation system, games, application programs, etc.
- Very high-level programming languages
  - Well-defined and implemeted abstract data structures
    - Collections, Bags, Sets, Dictionarys, Associative arrays, etc.
  - Often serve as control languages of the host system
  - Very slow languages; few 100-1000 times slower than C
  - Some develop into general purpose PLs (Python)
- Examples of script languages:
  - csh, bash, sed, AWK, Python, Perl, Tcl, Lua, JavaScript, PhP, JSP, itd.
  - First script languages were developed around 1960 (shells)

Logic programming languages

- Programming in Logic, Robert Kowalski
- Predicate calculus
  - Horn clauses
  - Unification, resolution
- Strong, abstract and simple language
- Back-tracking
  - Declarative and procedural semantics of Prolog
  - Operator CUT (!)
- Database programming languages
  - Datalog
- Sicstus, SB Prolog, SWI Prolog,…

Contents

- Course organization
- History of programming languages
- Programming models
- Concepts of programming languages
- Meta-Language ML
- Method of the course

Concepts in programming languages

- Programming language is a tool, defined in the form of a language, that is used for design and implementation of computer programs.
  - A tool is some form of a language
  - Text, graphical, 2D, 3D, etc.
- Concepts of programming languages are abstractions used for the representation of the structure and behavior of modeled systems.

Abstraction

- In Philosophy
  - A specific operation of the intellect consisting in detaching some properties and retaining some other properties of a thing
- Aristotel
  - Constructive mental process associated with the relation between forms (ideas) and concrete objects
  - Human mind actively abstracts or extracts forms from the objects in which they are realized
- St. Thomas Aquinas
  - Two kinds of abstractions
  - Mind joins properties that are separate from each other
  - Mind separates properties that are one

Concepts in programming languages

- Abstractions integrated into the language
  - Values, variables, branch, iteration, function, procedure, parameter passing, function composition, recursion, higher-order function, polymorphism, object, method, class, abstract class, specialization, aggregation, classification, module, functor, etc.
- Abstractions define language model
  - Imperative languages: variables, loops, procedures
  - Functional languages: functions, composition, recursion
  - Object languages: objects, methods, classes

Concepts in programming languages

- Algorithmic and data abstractions
  - Algorithmic abstractions
    - Imperative: sequence, loop, branch, blocks, etc.
    - Functional: function, recursion, polymorphism, etc.
  - Data abstractions
    - Simple: integer, bool, real
    - Structured: n-tuple, list, array, dictionary, set, recursive data structures, etc.
  - Algorithmic and data abstractions
    - Objects & classes: encapsulation, classification, inheritance, etc.
    - Modules: abstract data types, functions, abstract values

Concepts in programming languages

- Implementation of the language concepts
  - Function
    - Function call, parameter passing, activation records
  - Recursion
    - Stack of activation records
  - Variables
    - Symbol tables, value/reference models, namespaces
  - Memory representation of data structures
    - Lists, n-tuples, records, unions, objects, classes, modules
  - Objects and classes
    - Static/dynamic binding, inheritance
  - Memory management
    - Stack, heap

Properties of abstractions in PL

- Closer to hardware, simpler abstractions
  - Basic types: integer, real, string, etc.
  - Imperative language: loops, functions, procedures, etc.
  - Usually very efficient languages
- Higher-level abstractions
  - Objects, classes, modules, functors.
    - Not so efficient (inheritance, dynamic binding)
    - General purpose but also for specific areas (simulation, parallel systems, information systems, distributed systems, switching systems, protocols, etc.)
  - High-level abstractions, more directed language
    - Language designed for specific problems
    - Languages for programming with abstract data structures, e.g. tables (PL/SQL), Data-flow processing in distributed environemnt (Spark), Processing and analysing big data (Map-Reduce)

Properties of abstractions in PL

- General high-level programming languages
  - Include multiple levels of abstractions
    - Usually not the most efficient PLs
  - They are among more complex languages
    - C++, C#, F#, Ocaml, Java
  - General applicability
    - Financial analyses, numerical calculations (Fortran, Ocaml)
    - System programming (C++, Lua, Rust, Swift, Python, Ocaml)
    - Information systems (C#, F#, Java)

Abstractions

- Study of abstractions can improve learning process
  - COMMUNICATIONS OF THE ACM April 2007/Vol. 50, No. 4
    IS ABSTRACTION THE KEY TO COMPUTING?

IS ABSTRACTION THE KEY TO COMPUTING?

Why is it that some software engineers and computer scientists are able to produce clear, elegant designs and programs, while others cannot? Is it possible to improve these skills through education and training? Critical to these questions is the notion of abstraction.

By JEFF KRAMER

For over 30 years, I have been involved in teaching and research in computer science and software engineering. My teaching experience ranges from courses in programming, to distributed systems, distributed algorithms, concurrency, and software design. All these courses require that students are able to perform problem solving, conceptualization, modeling, and analysis. My experience is that the better students are clearly able to handle complexity and to produce elegant models and designs. The same students are also able to cope with the complexities of distributed algorithms, the applicability of various modeling notations, and other subtle issues.

Why study concepts in PL?

- We learn about possible ways of expressing ideas
  - Concepts of PL are tools for implementing an idea
- Every PL concept defined with abstraction and implementation
  - Every abstraction has a price
  - Recursion is expensive; iteration can be very complex
  - Working with objects is more expensive than working with common data structures
- We learn how to use given PL abstractions correctly
  - You have to know the tool to be used efficiently
  - Examples of using a PL construct are useful
  - Formal background of programming language
  - Experiences with tools and study of example systems

Contents

- Course organization
- History of programming languages
- Programming models
- Concepts of programming languages
- Meta-Language ML
- Method of the course

Lambda calculus

- Lambda calculus is the foundamental formalism for studying programming languages
- Theory of programming languages
  - Operational and denotational semantics
  - Using rule-based reasoning for:
    - deriving expression types, and
    - evaluation of expressions.
  - Static and dynamic semantics of PL
  - Prooving the properties of languages (determinism, strict typing, termination, etc.).
- Lambda calculus is a topic of the first lecture

Meta-Language

- Developed unintentionally while working on system for automatic theorem proving
- Logic for computable functions (LCF)
  - Robin Milner, Dana Scott
  - Stanford 1970-71, Edinburgh 1972-1995
  - LCF is a version of lambda calculus
  - Automatization of logic reasoning
- Meta-language for LCF system
  - ML used for:
    - Language for expressing proofs (step-by-step)
    - Expressing decisions (tactics) in proving
    - Implementation of theorem prover
  - Program execution is a search for proof

Meta-Language

- Success story from functional branch of programming languages
  - Standard ML, SML/NJ, Alice ML, Ocaml, Moscow ML, Mlton, MLKit, SML.NET, etc.
- Research related to ML
  - Hindley-Milner type-checking algorithm
    - Algorithm is included in a lecture on type checking.
  - Polimorfic lambda calculus (System F, Girard)
    - Implemented in Ocaml (System F=LCF)
  - Type systems, meta-programming
    - COQ, Twelf, CoffeeScript and TypeScript
  - ML had significant influence on development of PLs

Objective Caml

- Objective Caml is reference language
  - Caml core is pure lambda calculus.
  - Theoretically well-studied language.
  - Caml has strong typing.
  - Imperative constructs.
  - Parametric polymorphism
  - Nice object model
  - Modules and functors
- Ocaml offers more than one programming models
  - Imperative + functional + object-oriented + modular programming model
  - Also other PLs: C#, F#, Rust, Scala

communications of the acm
| november 2 0 1 1 | vo l . 5 4 | n o. 1 1

DOI:10.1145/2018396.2018413

Article development led by acmqueue
queue.acm.org

Why the next language you learn should be functional.

BY YARON MINSKY

OCaml for the Masses

Functional programming is an old idea with a distinguished history. Lisp, a functional language inspired by Alonzo Church's lambda calculus, was one of the first programming languages developed at the dawn of the computing age. Statically typed functional languages such as OCaml and Haskell are newer, but their roots go deep—ML, from which they descend, dates back to work by Robin Milner in the early 1970s relating to the pioneering Logic for Computable Functions (LCF) theorem prover.

Who uses OCaml

- Facebook
- Jane Street
- Bloomberg
- Microsoft
- Docker
- Citrix
- See: https://ocaml.org/learn/companies.html

F#

- F# is like C#
  - .NET language
  - F# is based on OCaml
  - C# is C-like language, similar to Java and C++ …
- Features
  - Lightweight syntax
  - Immutable by default
  - Type inference and automatic generalization
  - First-class functions
  - Powerful data types
  - Pattern matching
  - Async programming

Courses taught in OCaml

https://ocaml.org/learn/teaching-ocaml.html

North America

- Boston College - Computer Science I (CS 1101)
- Brown University - An Integrated introducion (CS 17/18) (along with Racket, Scala and Java)
- Caltech - Fundamentals of Computer Programming
- Columbia University - Programming Languages and Translators
- Cornell University - Data Structures and Functional Programming (CS 3110)
- Harvard University - Principles of Programming Language Compilation (CS153)
- Harvard University - Introduction to Computer Science II: Abstraction & Design (CS51)
- McGill University - Programming Languages and Paradigms (COMP 302)
- Princeton University - Functional Programming (COS 326)
- Rice University - Principles of Programming Languages (COMP 311)
- University of California, Los Angeles - Programming Languages (along with Python, Java) (CS 131)
- University of California, San Diego - Programming Languages: Principles and Paradigms (CSE130-a) (with Python, Prolog)
- University of Illinois at Urbana-Champaign - Programming Languages and Compilers (CS 421)
- University of Maryland (along with Ruby, Prolog, Java) - Organization of Programming Languages (CMSC 330)
- University of Massachusetts Amherst - Programming Languages (CMPSCI 631)
- University of Massachusetts - Programming Languages (CS691F)
- University of Minnesota Twin Cities — Advanced Programming Principles (CSCI 2041)
- University of Pennsylvania - Compilers (CIS341)
- University of Pennsylvania - Programming Languages and Techniques I (CIS120)
- University of Virginia - Programming Languages (CS 4610)

Courses taught in OCaml

https://ocaml.org/learn/teaching-ocaml.html

Europe

- Aarhus University - The compilation course (along with Java)
- Aix-Marseille University - Functional Programming
- Epita - Introduction to Algorithms (Year 1 & 2)
- ISAE/Supaéro - Functional programming and introduction to type systems
- Universidade da Beira Interior - Programming Languages and Compilers Design
- University Pierre & Marie Curie - Types and static analysis (5I555)
- University Pierre & Marie Curie - Models of programming and languages interoperability (LI332)
- University of Birmingham - Foundations of Computer Science (FOCS1112)
- University of Cambridge - Advanced Functional Programming (L28)
- University of Innsbruck - Programming in OCAML (SS 06)
- University of Rennes 1 - Programming 2 (PRG2)
- University of Wrocław - Functional Programming
- Université Paris-Diderot - Advanced Functional Programming (PFAV)
- Université Paris-Diderot - Functional Programming (PF5)

Asia

- Indian Institute of Technology, Delhi - Introduction to Computers and Programming (CSL 101) (along with Pascal and Java)

Contents

- Course organization
- History of programmiing languages
- Programming models
- Concepts of programming languages
- Meta-Language ML
- Method of course

Methods of the course

- Concepts of programming languages
  - Language is »conceptual universe” (Perlis)
- Learning more than one PL model
  - Functional, Imperative, Object-oriented, Modular, Logical
- Learning more than one programming language
  - Lambda calculus, ML, Python, Java, C
- Comparing the concepts of PL
  - Which construct is more appropriate in the given context?
  - Implementations of concepts can be different
- Implementations of PL concepts
  - Knowledge about implementation of PL concepts enables writing efficient programs

Languages used in the course

- Lambda calculus
  - Formal foundation
  - Basic principles of functional PLs
  - Lisp: first programming language based on LC
- Meta-Language
  - Mathematical foundations, strong typing, mainly functional
  - Ocaml contains four PL models
- PLs for comarison (with Ocaml): C, Java, Python
  - Python is currently the most popular language.
  - C is still among the most popular languages.
  - Java uses clean (simplified) object-oriented model.
- Additonal PLs: C++, Fortran, Scala, Go
  - Presenting specific concepts of PLs