Principles of Programming Languages [PLP-2016]
Detailed Syllabus

This document lists the topics presented along the course. The PDF slides published on the course web page (http://www.di.unipi.it/~andrea/Didattica/PLP-16/) provide a detailed outline of the topics to be studied.
The presented topics are based mainly on selected chapters of the following textbooks:

- **[ALSU] Compilers: Principles, Techniques, and Tools** by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, 2nd edition
  Chapters 2 to 6 [excluding sections 4.7.5 and 4.7.6], 8 [till sec. 8.9], 9 [till sec. 9.6]
- **[Scott] Programming Language Pragmatics** by Michael L. Scott, 3rd edition
  Chapters 1, 3, 6, 7, 8 [Section 8.3 only], 9, 10, 13
- **[GM] Programming Languages: Principles and Paradigms** by Maurizio Gabbrielli and Simone Martini
  Chapters 1, 4, 5, 6, 7 [till section 7.2], 8 [till section 8.10], 9, 10, 11
- **[Mitchell] Concepts in Programming Languages** by John C. Mitchell
  Chapters 5, 6 and 7 on Haskell [these are not included in the printed book]

Some additional reading material is indicated below, where relevant. Links to relevant online resources can be found on the course web page.

List of topics

1. Introduction. Abstract machines, interpretation and compilation
   [GM], Chapter 1; [Scott], Chapter 1 (sections 1-4 to 1-6)
   a. Abstract machines
   b. Compilation and interpretation schemes
   c. Cross compilation and bootstrapping
   d. Structure of compilers
2. Overview of a syntax-directed compiler front-end **[ALSU], Chapter 2**
   a. (Context-Free) Grammars, Chomsky hierarchy
   b. Derivations, parse trees, abstract syntax trees
   c. Ambiguity, associativity and precedence
   d. Syntax-directed translation, translation schemes
   e. Predictive recursive descent parsing
   f. Left factoring, elimination of left recursion.
   g. Lexical analysis
   h. Intermediate code generation
   i. Static checking
3. Lexical analysis, Implementing critical parts of a scanner **[ALSU], Chapter 3**
   a. Tokens, lexeme and patterns
   b. Regular expressions and regular definitions
   c. Transition diagrams
   d. Code of a simple lexical analyzer
   e. Lexical errors
   f. Nondeterministic and deterministic finite-state automata (NFA and DFA)
   g. From regular expressions to NFA (Thompson construction)
   h. From NFAs to DFAs (Subset construction algorithm)

i. Minimization (partition-refinement) algorithm for DFAs, Myhill-Nerode theorem
j. The Lex-Flex lexical analyzer generator

4. From DFAs to regular expressions and backwards ***[optional topic]***
   [Reading material (downloadable from the Moodle page of the course):
   (1) Selected pages of of Aiello, Albano, Attardi, Montanari: Teoria della Computabilità, Logica, teoria dei linguaggi formali, Materiali didattici ETS, 1979, in Italian.
   (2) Ginsburg and Rice: Two Families of Languages Related to ALGOL, Journal of the ACM Volume 9 Issue 3, July 1962]
   a. From a DFA to a right-linear grammar
   b. Context-free grammars as continuous transformations on languages
   c. Kleene fixed-point theorem
   d. Generated language as least fixed-point of a grammar
   e. REs as solutions of least-fixed points equations
5. Parsing **[ALSU], Chapter 4.**
   a. Parser as string recognizer (acceptor)
   b. Left-recursion elimination, left-factoring, LL(1) grammars
   c. Recursive-descent parsing, table-driven parsing
   d. Error recovery during top-down parsing.
   e. Bottom-Up, shift-reduce parsing: handles
   f. Stack-implementation of shift-reduce (driver)
   g. Shift/reduce and reduce/reduce conflicts
   h. LR(0) items, LR(0) automaton and LR(0) parsing table, SLR parsing
   i. LR(1) items, automaton and canonical parsing table, LALR parsing tables
   j. LR parsing with ambiguous grammars
   k. Error detection during shift/reduce parsing
   l. Parser generators: Yacc/Bison, dealing with ambiguous grammars in Yacc
6. Syntax-Directed Translation **[ALSU], Chapter 5**
   a. Syntax-directed definitions (attribute grammars)
   b. Synthesized and Inherited attributes, annotated parse trees
   c. S-attributed definitions: evaluation with postorder depth-first traversal
   d. Evaluation order of attributes, dependency graph, topological sort
   e. L-attributed definitions: evaluation with depth-first, left-to-right traversal
   f. Syntax-directed translation schemes
   g. Postfix translation schemes and their implementation with LR parsing
   h. Translation schemes for L-attributed definition schemes: implementation with top-down and bottom-up parsing
7. Intermediate Code Generation **[ASLU] Chapter 6**
   a. Intermediate representations
   b. Syntax-directed translation to three-address code
   c. Handling names in local scopes
   d. Translation of declarations, expressions and statements in scope
   e. Translation of short-circuit boolean expressions
   f. Translation of conditionals and iteration
   g. Use of backpatching lists
8. Code generation **[ALSU] Chapter 8**
   a. Instruction selection, register allocation and assigment, instruction ordering
   b. Target machine architecture and instruction set/addressing modes

c. Flow graphs: basic blocks, control flow graphs, partition algorithm
d. Loops in Control Flow Graphs
e. Local vs. Global Optimization
f. DAG Based Optimization
g. Peephole Optimization and other techniques
h. Next-use and liveness informations
i. Simple code generation algorithm
j. Simple register allocation algorithm
k. Global register allocation with graph coloring
l. Instruction selection using tree translation schemes
m. Optimal register allocation for expressions using Ershov numbers

9. Data-Flow analysis **[ALSU] Chapter 9**
    a. Global, Machine Independent Optimization
    b. Liveness, Available Expressions, Very Busy Expressions and Reachable Definitions Analysis
    c. The Data-Flow analysis frameworks
    d. Data-Flow iterative algorithm
    e. Map semilattices and Constant Propagation Analysis
    f. Accuracy, Safeness, and Conservative Estimations
    g. Determining loops in flow graphs: dominators
    h. Data-Flow analysis for dominators
    i. Region Based Analysis, Symbolic Analysis ***[optional topic]***
10. Programming languages and abstraction: names and bindings **[Scott] Chapter 3, [GM] Chapters 4 and 5**
    a. Programming language and abstractions
    b. Runtime environment
    c. Names and abstraction
    d. Bindings and binding time
    e. Static, stack and heap allocation of memory
    f. Scope of a binding
    g. Static scoping and Closest Nested Scope Rule
    h. Static Links and Displays for supporting Static Scoping
    i. Declarations and Definitions, Modules
    j. Local symbol tables during compilation
    k. Syntax-Directed Translation of three-address code in scope **[ALSU] Section 2.7**
    l. Implementation of scopes **[Scott] Section 3.4**
        i. Static scoping: LeBlanc & Cook lookup algorithm
        ii. Dynamic scoping: association lists and central reference tables
    m. Shallow and deep binding
    n. Functions returning procedures and retention; Object Closures
    a. Type Systems
    Type Errors Modules as abstraction and encapsulation mechanism
    b. Modules as algebraic data types, modules as classes
    c. Implementation of scopes **[Scott] Section 3.4**
        - Static scoping: LeBlanc & Cook lookup algorithm
        - Dynamic scoping: association lists and central reference tables
    d. Returning subroutines as closures with unlimited extent
    e. Object closures in Object Oriented languages.

11. Type systems **[Scott] Chapter 7, [GM] Chapter 8, [ALSU] Chapter 6**
    a. Data types, type errors, type safety
    b. Static vs. dynamic typing, conservativity of static typing
    c. Type equivalence: structural vs. name equivalence
    d. Type compatibility and coercion
    e. Discrete types, scalar types, composite types
    f. Tuples, records and arrays
    g. Generating intermediate code for array declaration and access
    h. Disjoint unions types: algebraic data types, discriminated records, variants, objects, active patterns in F#
    i. Value Model and Reference Model of variables
    j. Preventing dangling pointers: tombstones, locks and keys
    k. Pointers and arrays in C
12. Functional programming languages **[Scott] Chapter 10, [GM] Chapter 11, [Mitchell] Chapter 5**
    a. Historical origins and main concepts
    b. Functional languages: the LISP family, the ML family, Haskell
    c. Applicative and Normal Order evaluation of lambda-terms
    d. Overview of Haskell
        - Primitive types, Algebraic Data Types, Lists and List Constructors
        - Patterns and declarations, functions and pattern matching
        - List comprehension
        - Higher-order functions
        - Lazy evaluation
    e. Implementation of Overloading through Type Classes and Constructor Classes in Haskell **[Mitchell] Chapter 7**
    f. Monads in Haskell; Monads as containers and as computations, the IO Monad
    g. Type Inference: the Hindley-Milner algorithm
       [Mitchell] Chapter 6: pages 118-136
    h. Type Inference with Overloading: generating type constraints
    i. Recursion vs. iteration, tail recursion **[Scott] Section 6.6**
    j. Continuation passing style (CPS)
        - Making argument evaluation order explicit
        - Tail recursion and CPS
13. Java 8 extensions
    a. Lambda expressions in Java 8
    b. The stream API in Java 8