Principles of Programming Languages [PLP-2015]
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-15/) 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.

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
   k. From RE to DFA directly
4. From DFAs to regular expressions and backwards
   [Reading material (see course web page): (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. Programming languages and abstraction: names and bindings **[Scott] Chapter 3, [GM] Chapter 4**
   a. Programming language concepts as abstractions of Abstract Machine components
   b. Abstraction by naming, by parametrization and by specification
   c. Names as abstractions, binding times
   d. Object lifetime vs. binding lifetime
   e. Static, stack and heap allocation of objects
   f. Stack allocation: Activation records and stack management
   g. Implicit and explicit heap allocation; heap allocation algorithms
8. Scoping rules **[Scott] Chapter 3, [GM] Chapter 4 and 5**
   a. Static vs. dynamic scoping
   b. Closest nested scope rule

    c. Resolving non-local references with static scoping: static links and displays
    d. Implementation of dynamic scope: binding stack with name-object bindings
    e. Modules as abstraction and encapsulation mechanism
    f. Modules as algebraic data types, modules as classes
    g. Implementation of scopes **[Scott] Section 3.4**
        - Static scoping: LeBlanc & Cook lookup algorithm
        - Dynamic scoping: association lists and central reference tables
9. Denotational semantics: a light introduction
   [Reading material: R.D. Tennent: The denotational semantics of programming languages, Communications of the ACM, Volume 19 Issue 8, Aug. 1976]
    a. Syntactic domains, semantic domains and semantic interpretation functions
    b. Denotational semantics of LOOP programs
    c. Complete Partial Orderings (CPOs) as semantic domains for recursive definitions
    d. Denotational semantics of assignment, blocks and parameterless procedures
10. More on management of bindings **[Scott] Chapter 3**
    a. Aliases and Overloading
    b. Deep vs. shallow binding for procedural parameters, with dynamic or static scoping
    c. Denotational semantics of deep/shalow binding: intuition
    d. Returning subroutines as closures with unlimited extent
    e. Object closures in Object Oriented languages.
11. Control flow in programming languages **[Scott] Chapter 6, [GM] Chapter 6**
    a. Evaluation order of expressions, short-circuit evaluation
    b. Assignment: value and reference memory model
    c. Denotational semantics of Value and Reference Model
    d. Structured and unstructured flow, sequencing and selection
    e. Iteration: enumeration controlled and logically controlled loops
    f. Iterators and collections/containers, iterators in Java
    g. True iterators and iterators based on higher order functions
12. 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
13. 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. Pointers as references in value model memory stores

    j. Preventing dangling pointers: tombstones, locks and keys
    k. Pointers and arrays in C
    l. Recursive data types: lists in various programming languages
14. Control abstraction **[Scott] Section 8.3, [GM] Chapter 7**
    a. Parameter Passing Modes and Mechanisms
    b. Call by name/value/result/reference/sharing/need
    c. Closures
    d. Default parameters, named parameters, varargs
15. Data Abstraction and Object Oriented programming languages **[Scott] Chapter 9, [GM] Chapter 9-10**
    a. Abstraction mechanisms applied to data
    b. Object Oriented: Encapsulation + Inheritance + Dynamic method binding
    c. Visibility rules in Java and C++
    d. Initialization and finalization of objects
    e. Dynamic binding: virtual functions in C++, methods in Java
    f. Multiple inheritance
    g. Mix-in inheritance in Java; Classes, Abstract classes and Interfaces
16. 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
17. Scripting languages **[Scott] Chapter 13**
    a. Origins and common characteristics
    b. Problem domains: shell languages, text processing and report generations, “glue” languages, extension languages, WWW (server and client side)
    c. Innovative features (supported in various ways)
        - Variable declarations not needed, thus typing is dynamic
        - Various original nesting and scoping rules
        - Rich set of string and pattern (regular expressions) manipulation operators
        - Data types: generic numeric types, associative arrays (hash tables)
        - Object Orientation

18. 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
    e. DAG representation of basic blocks, equivalence of basic blocks
    f. Local Transformation Techniques
    g. Next-use and liveness informations
    h. Simple code generation algorithm
    i. Simple register allocation algorithm
    j. Peephole optimization
    k. Global register allocation with graph coloring
    l. Instruction selection using tree rewriting
19. Dataflow analysis **[ALSU] Chapter 9**
    a. Data flow analysis frameworks
    b. Data flow iterative algorithm
    c. Examples: reaching definitions, live variables, constant propagation / folding
    d. Accuracy, Safeness, and Conservative Estimations
    e. Determining loops in flow graphs: dominators
    f. Data flow analysis for dominators