

# Topics and Readings for the Programming Languages Component of the PhD Comprehensive Examination

Department of Computer Science  
University of New Mexico

October 2007

## 1 General Guidelines

*This exam covers various aspects of programming. The exam is given in three sections: the practice of programming, programming language implementation, and the theory of programming languages.*

### 1.1 Preparing for the exam

*There are several things you can do to prepare for the exams:*

- *Talk to professors about the exam.*
- *Talk to students who have already taken the exam.*
- *Take the courses on the material the exams cover.*
- *Read books and articles on the material the exams cover.*
- *Join a study group.*
- *Try to answer sample questions on the material the exams cover.*

## 1.2 General reading

If you can only read one book, read this one: John C. Mitchell, *Concepts in Programming Languages*, Cambridge University Press, 2003. In general, you will need to read more than one book.

## 1.3 On the lists of topics

The following sections list several groups of topics, at the undergraduate and the graduate level, in which proficiency is expected. The listing indicates with each group of topics the UNM courses in which the topics are usually covered or touched upon, and books and articles that should be consulted for in-depth study.

It is by no means intended that you should read *all* the recommended books listed here, nor, indeed, that you should master *all* the topics listed: the lists are here as a beginner's guide for an in-depth study of programming languages. Instead, enroll in the relevant graduate-level courses, **650** and **655**, and perhaps in the undergraduate-level course, **557**, and concentrate on the material and the sources used there.

## 1.4 On the nature of exam questions

Exam questions are at the level of easy-to-moderate final exam questions in classes such as **650** and **557**. Occasionally, questions can be longer or more in-depth than the 2-hour format of a final exam allows (the PhD exam is 8 hours under present rules).

# 2 Undergraduate-level topics

Normally, entering graduate students are expected already to have mastery of these subjects. Students who lack this background must take **557**, and any other needed classes:

## 2.1 Topics in programming languages:

1. Proficient programming in an imperative language (such as C). *Related course(s):* . *Suggested reading:* (too many titles to list)

2. Proficient programming in an industrial-strength object-oriented language (such as Java), and general familiarity with concepts in object-oriented languages (inheritance of interface and implementation, subtyping vs. subclassing, abstract data types). *Related course(s):* . *Suggested reading:* (too many titles to list)
3. Proficient programming in a typed functional language (such as SML and Haskell). Mastery of concepts in functional programming: higher-order functions, parametric polymorphism, and type reconstruction. *Related course(s):* . *Suggested reading:* Lawrence C. Paulson: *ML for the Working Programmer, 2nd edition*. Richard Bird: *Introduction to Functional Programming using Haskell*. Simon Thompson: *Haskell: The Craft of Functional Programming*.
4. Proficient programming in a dynamically-typed functional language, and, specifically, in Scheme. *Related course(s):* . *Suggested reading:* George Springer and Daniel Friedman: *Scheme and the Art of Programming*. Abelson and Sussman: *Structure and Interpretation of Computer Programs*. R. Kent Dybvig: *The Scheme Programming Language, 3rd edition*
5. Basic proficiency in a logic programming language and, specifically, in Prolog. Knowledge of elementary logic concepts: propositional and first order predicate logic, concept of proof, knowledge representation. Elementary knowledge of logic programming concepts: informal declarative and operational semantics, pure logic programming, recursive programs and data types, search, control. *Related course(s):* . *Suggested reading:* Sterling & Shapiro: *The Art of Prolog, 2nd edition* K. Apt: *From Logic Programming to Prolog*. Clocksin & Mellish: *Programming in Prolog, 5th edition*.
6. Formal languages: finite state automata, regular expressions, grammars, context-free grammars, regular grammars. *Exam questions do not test knowledge in this area directly, but the area is a necessary prerequisite for several graduate-level topics. Related course(s):* . *Suggested reading:*

## 2.2 Instrumental topics:

1. Computer organization: basic proficiency with assembly language/machine code as needed to be able to generate correct code. Memory hierarchy, instruction level parallelism, etc. as they apply to language implementation.

*Related course(s):* . *Suggested reading:* John L. Hennessy and David A. Patterson: *Computer Architecture: A Quantitative Approach, 4th edition.*

2. Operating systems concepts: as they relate to language implementation (file systems, process management, locking, etc. *Related course(s):* . *Suggested reading:*
3. Elementary notions of compilation. Lexical analysis, syntax representation, register allocation, symbol tables, run-time management, code generation. *Related course(s):* . *Suggested reading:* Cooper & Torczon: *Engineering a Compiler.*

### 3 Graduate-level topics

These topics should be covered by taking classes such as **650, 554, 655, 557**, seminars, and/or through individual study.

#### 3.1 Programming language implementation

##### 3.1.1 Program interpretation

1. Understanding the notion of interpretation and the structure of interpreters. *Related course(s):* **357,557,554**. *Suggested reading:* Abelson and Sussman: *Structure and Interpretation of Computer Programs*

##### 3.1.2 Program compilation

1. Parsing and scanning: standard techniques of one-pass parsing for regular and context-free grammars—practical lexical analysis, top-down parsing, recursive-descent parsing, predictive parsing. Attribute grammars and transducers. *Related course(s):* **201, 351, 454/554, 655**. *Suggested reading:* Michael L. Scott: *Programming Language Pragmatics, 2nd edition.* Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman: *Compilers: Principles, Techniques, and Tools, 2nd edition.*
2. Analysis, optimization, and code generation: intermediate representations for optimization and code generation—at an elementary level. Code generation: traditional instruction selection, BURS compiler generation. Elementary dataflow analyses. Register allocation. Instruction scheduling. *Related*

*course(s): 454/554, 655. Suggested reading: Steven Muchnick: Advanced Compiler Design and Implementation.*

### **3.1.3 Implementation of specific programming paradigms**

1. Functional language implementation: definitional interpreters, executable semantic descriptions, compilation of strict and lazy languages. *Related course(s): 655. Suggested reading: Michael J. C. Gordon: Programming Language Theory and its Implementation. Simon Peyton Jones and David R Lester: Implementing Functional Languages: a tutorial. Andrew W. Appel: Compiling with Continuations.*
2. Logic language implementation: the Warren Abstract Machine. Concurrent and Parallel execution models. *Related course(s): 655. Suggested reading: Hassan Aït-Kaci: Warren's Abstract Machine, A Tutorial Reconstruction.*

### **3.1.4 Run-time systems**

1. Garbage collection basics: objectives; copying, mark and sweep, and generational methods. *Related course(s): 554. Suggested reading: Richard Jones and Rafael Lins: Garbage Collection: Algorithms for Automatic Dynamic Memory Management.*
2. Languages with dynamic compilation. The structure of the Java Virtual Machine. *Related course(s): 554. Suggested reading: Daniel P. Friedman, Mitchell Wand, and Christopher T. Haynes: Essentials of Programming Languages.*

## **3.2 Programming language theory**

1. Lambda-calculus: syntax, notion of substitution, notions of conversion ( $\alpha$ ,  $\beta$ ,  $\eta$ ), redexes and reduction order, Church-Rosser theorem, programming in the pure lambda-calculus, simply-typed lambda calculus. *Related course(s): 557, 650, 655. Suggested reading: Benjamin C. Pierce: Types and Programming Languages.*
2. Logic programming and logic programming languages, constraint logic programming, constraint solving, higher-order, semantics (declarative, operational, fixpoint, equivalences), unification and resolution, fundamental theo-

retical results (semi-decidability). *Related course(s): 650. Suggested reading:* Marriott & Stuckey: *Programming with Constraints: An Introduction*.

3. Axiomatic semantics: Floyd-Hoare-Dijkstra Logic, invariants, verification of properties of programs (imperative and functional). *Related course(s): 650. Suggested reading:* J. Loeck and K. Sieber: *The foundations of program verification*.
4. Denotational semantics: direct and continuation, domains (Scott's construction), fixed points, least fixed points, interaction between language design and semantics. *Related course(s): 650. Suggested reading:* Joseph Stoy: *Denotational Semantics*. Michael J. C. Gordon: *The Denotational Description of Programming Languages*. David Schmidt: *Denotational Semantics: A Methodology for Language Development*.
5. Concurrency: semaphores, fork and join, coroutines, tasking, exception handling, message passing (as in Smalltalk and actors), models of concurrency. *Related course(s): 481, 650. Suggested reading:*