Fall 1995 College of Computing Core Examination

Areas to be Tested

- Algorithms and Data Structures
- Computer Organization
- Formal Languages and Automata Theory
- Operating Systems
- Programming Languages

The required knowledge in each core area will be approximately equivalent to the course material of an upper-level undergraduate course in that area. The core knowledge will be tested by a **4 hour comprehensive written examination** that tests competence in all areas. This examination is expected to be at approximately the same level as the advanced GRE test in computer science. All reading appropriate in preparing for GRE advanced examination in computer science is appropriate for this examination as well.

The Core Examination is designed to be diagnostic. For each area passed, the core requirement in that area is satisfied. For each area failed, the student will be directed to take an appropriate course, either at the graduate or undergraduate level, as directed by the area faculty. Students are expected to make an appropriate grade in the core area course, to be determined by the course instructor and the area advisor.

Students also have the option of not taking the exam, or certain areas on the exam, if they feel their background is inadequate (e.g., students with a non-computer science background). Any area not taken must be satisfied via the course option described above. The courses should be completed by the end of the student's fifth quarter in the Ph.D. program.

Format of the Examination

The examination will be closed book and closed notes. There will be 50 minutes per area and roughly 8-9 questions per area. The examination will consist of three types of questions. An example of each type is provided below.

1. *Multiple choice.*
   One possible method of *preventing* a deadlock is
   (a) follow a hierarchical ordering on mutual-exclusion locks for acquisition and release;
   (b) follow a first-come-first-serve order for granting locks to processes;
   (c) follow a random order for granting locks; or
   (d) none of the above
2. *True/false with a brief justification*
   Dijkstra's shortest-path algorithm can be implemented in time $O(n \log n)$ for a sparse graph (a graph with n vertices is said to be *sparse* if it has $O(n)$ edges).
3. *Short descriptive answers*
   Compare and contrast polling and interrupt as two mechanisms for programmed data transfer in a computer system.

Core Examination **Fall 1995**

Information for Specific Areas

Algorithms and Data Structures

Representative faculty: Howard Karloff (area advisor), John Stasko, Vijay Vazirani, Craig Tovey (in ISyE), Amihood Amir.

Relevant courses: CS 3158 and 6155.

Suggested textbooks:

1. Baase, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley.
2. Brassard and Bratley, *Algorithmics*, Prentice Hall, 1988.
3. Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.
4. Manber, *Introduction to Algorithms*, Addison-Wesley, 1989.

Outline:

This examination will cover basic techniques for designing and analyzing algorithms for sorting and searching problems, graph problems, string-matching, etc. Some introductory material about NP-Completeness is also covered.

Topics:

- Measuring the asymptotic growth of functions.
- The basic techniques for manipulating bounded summations.
- Solving recurrence relations.
- The basic structures of computing: sets, relations, functions, graphs, and trees.
- Heapsort and quicksort.
- Techniques for problem solving:
  - divide-and-conquer;
  - dynamic programming;
  - greedy heuristics.
- Elementary graph algorithms.
- Minimum spanning trees and shortest paths algorithms.
- String-matching algorithms.
- Selecting the k-th largest.
- Introduction to NP-Completeness.

Core Examination Fall 1995

Computer Organization

Representative faculty: Kishore Ramachandran (area advisor), Richard Fujimoto, Ann Chervenak.

Relevant courses: EE 1300, CS 2760, and CS 3760.

Suggested textbooks:

1. Mano, *Computer Engineering Harware Design*, Prentice-Hall, 1988.
2. Tanenbaum, *Structured Computer Organization*, Prentice-Hall, 1984.
3. Sowell, *Programming in Assembly Language VAX-11*, Addison-Wesley, 1987.
4. Hamachar, et al., *Computer Organization*, 3rd Edition, McGraw-Hill, 1990.

Topics:

- Basic concepts in digital design:
  - boolean algebra and logic,
  - combinational and sequential logic systems,
  - logic minimization;
- Number systems;
- Data representation and basic computer arithmetic (integer and floating point);
- Addressing modes and Instruction set design;
- Philosophy of instruction-set design (orthogonality, regularity, RISC and CISC approaches, compiler and OS issues);
- Processor design (datapath and control, hardwired versus microprogrammed implementation approaches);
- Hardware description languages;
- Principles of parallelism (overlapped and pipelined execution);
- I/O organization (DMA, programmed transfer, interrupts);
- Memory hierarchy (main memory, caches, virtual memory).

Core Examination Fall 1995

Formal Languages and Automata Theory

Representative faculty: Leonard Schulman (area advisor), Howard Karloff, Craig Tovey (in ISyE), Amihood Amir.

Relevant course: CS 3156 and 6152.

Suggested textbooks:

1. Hopcroft and Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
2. Lewis and Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, 1981.
3. Linz, *An Introduction to Formal Languages and Automata*, D. C. Heath, 1990.

Outline:

This course covers basic concepts and techniques of theoretical computer science involving finite automata, pushdown automata, Turing machines, and formal languages.

Topics:

- Sets, relations, functions, closures, proof techniques, and languages;
- Deterministic and nondeterministic finite automata and their equivalence;
- Regular expressions; regular sets and finite automata; the pumping theorem for regular sets;
- Context-Free grammars, regular grammars, normal forms; pushdown automata and context-free languages.
- Closure, algorithmic properties of context-free grammars; the pumping theorem for context-free languages;
- Turing machines; creating more powerful Turing machines; nondeterministic Turing machines;
- Church's Thesis; universal Turing machines;
- The Halting Problem; recursively enumerable (Turing-enumerable) sets; recursive (Turing-acceptable) sets; decidability.

Core Examination Fall 1995

Operating Systems

Representative faculty: Mustaque Ahamad (area advisor), Karsten Schwan, Mostafa Ammar, Richard Fujimoto, Scott Hudson, Kishore Ramachandran.

Relevant course: CS 4430.

Suggested textbook: Silberschatz and Peterson, *Operating System Concepts*, 3rd edition, Addison-Wesley.

Outline:

Students should understand the concepts and techniques required for understanding and designing operating systems. There are three main parts. The first part deals with understanding a fundamental concept in an operating system, namely, a process. It includes interprocess communication, synchronization, concurrency and deadlock detection. The second part deals with resource management in general and in particular, with CPU management, memory management, and secondary storage management, including file structures. Some algorithms regarding the efficient utilization of resources should be known as well. The last part deals with issues that arise in distributed and multiprocessor systems.

Topics (chapter references are to Peterson and Silberschatz):

- Concurrent processes (chapter 5);
- Deadlocks (chapter 6);
- CPU scheduling (chapter 4);
- Memory management (chapter 7);
- Virtual memory (chapter 8);
- Secondary storage (chapter 9);
- File systems (chapter 10)
- Protection (chapter 11)
- Distributed Systems (chapter 12)

Core Examination — Fall 1995

Programming Languages

Representative faculty: Bill Appelbe (area advisor), Richard LeBlanc.

Relevant course: CS 3411 (CS 4410 — useful but not required).

Suggested Textbooks: Any language survey textbook, such as:

- Sebesta, *Concepts of programming Languages*, Second Edition.
- MacLennan, *Principles of Programming Languages: Design, Evaluation and Implementation*, Second Edition.
- Pratt, *Programming Languages: Design and Implementation*, Second Edition.

(Please see the area advisor for publisher and date information.)

Reference Textbooks (useful, but more depth than the examination):

Fisher and LeBlanc, *Crafting a Compiler* (Or other compiler textbooks such as Aho, Sethi, and Ullman.) (Please see the area advisor for publisher and date information.)

Outline:

Principles and paradigms of programming languages including Fortran, Algol, Pascal, Ada, Lisp, SmallTalk, C, C++, and Prolog. Familiarity with historically important programming languages such as Snobol, Algol-68, PL/1 (their major contributions). Programming expertise (the ability to write syntactically correct programs) is not examinable. Familiarity with the distinctions between the different classes of programming languages (conventional, functional, logic, object-oriented). Issues in the design and implementation of programming languages are such as data types, parameter passing mechanisms, block structure, information hiding, scoping rules and implementation issues. Structure of compilers and interpreters; grammars (BNF), translation and interpretation. Storage management: stacks and heaps, activation records.

Topics:

- Fortran (pass by reference, activation records);
- Algol (pass by name, pass by value, static versus dynamic scope);
- Pascal (block structured languages, activation records, implementation of procedure call, type equivalence);
- Ada (types, packages, exception handling, in and out parameters);
- Lisp (atoms and lists, functions, property lists, static versus dynamic binding);
- SmallTalk (object oriented programming);
- C++ (types, classes, derived classes, multiple inheritance, references, virtual functions, overloading);
- Prolog (logic, theorems and programs, unification, evaluation order in Prolog);
- Scanners and Parsers (BNF, context-free grammars);
- Runtime Environments (activation records, thunks, heaps).