Principles of Programming Languages

General Comments: for each topic, the students are expected to demonstrated competency in both design issues as well as how features and constructs are implemented. Ability to demonstrate concepts through examples in existing programming languages is also expected.

Reference Books:

General Principles
- Categories of Programming Languages and Programming Paradigms
- Implementation models of programming languages
  - Interpreters
  - Compilers
  - Virtual Machines
  - Just-in-time Compilation
- General Purpose vs. Domain Specific Languages

Syntax and Semantics
- Informal and Formal models for describing syntax
  - Regular Expressions and Finite State Automata
  - Context Free Grammars
  - Attribute Grammars
- Concepts of Lexical Analysis
- Concepts of Parsing
  - Recursive-descent parsing
  - Bottom-up parsing
- Describing semantics
  - Assertions and Axiomatic Semantics
  - States, State transformations and Operational Semantics

Names, Bindings, and Scope
- Bindings and binding times
- Variables
• Scope
• Lifetime
• Referencing Environments

• Constants and Values

Data Types
• Atomic Data Types
  o Numeric Data Types
  o Character Data Types
  o Other atomic primitive data types (e.g., Booleans)
  o User defined Atomic Data Types
• Complex Data Types
  o Arrays
  o Records
  o Unions
  o Pointers
  o Sets and Lists
  o Other complex data types
• Handling Data Types
  o Type conversions
  o Type checking
  o Strong Typing
  o Type equivalence

Control Structures
• Expressions and expression-level control
  o Design
  o Implementation
• Assignments
  o Types of assignments
  o Design Issues
  o Implementation Issues
• Statement-level control structures
  o Sequencing
  o Selection
  o Iteration
  o Jumps
  o Guarded Commands and Concurrent control
  o Implementation of control structures
• Subprograms and subroutines
  o Design issues
  o Referencing and reference environments
  o Parameter passing methods
  o Subroutines as parameters
  o Overloading
  o Generic subroutines
Coroutines
  o Implementation of subroutines

Exceptions and Exception handling
  o Design
  o Implementation

Abstract Data Types
  • Abstraction and Encapsulation; Information Hiding
  • Design issues for ADT
  • Examples of language-level support for ADT
  • Parameterized ADTs

Object Oriented Programming Languages
  • Objects, Classes
  • Forms of Inheritance and issues underlying inheritance
  • Models of OO in different programming languages
  • Implementation of OO languages

Logic Programming
  • Foundations of logic programming languages
  • Prolog and its execution model
    o Syntax
    o Unification
    o Resolution
  • Extensions of Prolog
    o Constraint solving
    o Tabling

Functional Programming
  • Foundations of functional programming
  • Basics of Scheme
  • Other functional programming languages

Concurrency and Concurrent Languages
  • Statement-level and Subprogram-level concurrency
  • Processes, Threads
  • Critical sections and mutual exclusion
    o Semaphores
    o Monitors
    o Rendez-vous and Message Passing
  • Examples from different languages
Design and Analysis of Algorithms and Data Structures

- Asymptotic notations
- Recurrence equations
- Divide-and-conquer strategies
- Dynamic programming
- Greedy algorithms
- Graph algorithms
- Worst case analysis, average case analysis
- Correctness proof
- Basic data structures (e.g., linked lists, heap, stack, tree, balanced tree, binary heaps, hashing)

Discrete Mathematics

- Concepts of set, function, relation, sequence (list), graph, tree.
  Skills:
  (1) Able to read and write using the set language terminologies and notations.
  (2) Able to prove two sets are the same, reasoning about functions and relations.
- Propositional and first-order logic.
  Skills:
  (1) Able to use formal logic to formulate a mathematical statement.
  (2) Able to prove a statement that involves existential and universal quantifiers.
  (3) Able to formulate the concept of limit using quantifiers.
- Mathematical induction.
  Skills:
  (1) Able to write mathematical induction proofs.
  (2) Able to define discrete structures inductively.
  (3) Able to reason about recursive structures and codes using induction.
- Sequences and series (arithmetic and geometric series, summation techniques, basic summation formulas).
  Skills:
  (1) Able to define sequences using recurrences.
- Basic concepts in combinatorics (permutation, combinations).
  Skills:
  (1) Able to devise algorithms for enumerating combinations and permutations.
  (2) Able to use recurrences for solving combinatorics problems.
- Basic concepts of probability theory (random variables, expectation, standard deviation).
• Basic number theory (number systems (binary and decimal numbers), modular arithmetic, Euclid's gcd algorithm).
• Basic concepts of matrices and linear algebra (solving linear system, inverting matrices, matrix rank, linear independence, dimension of vector space, basis vectors, subspace).

Operating Systems

• System calls, operating system debugging; process concept and scheduling; interprocess communications.
• Multithreading models and thread libraries; issues with threading; I/O (blocking vs. nonblocking).
• Scheduling algorithms for CPU, multiprocessors, and threads; native threads vs. user-level threads.
• Critical section; process synchronization using semaphores, monitors, and mutexes; deadlock characterization, prevention, and handling; deadlock detection and recovery.
• Paging and segmentation of memory; page replacement, copy-on-write, demand paging; paging data structures (e.g., page table design).
• File systems (e.g., ext3fs, ext2fs, and ntfs).

Architecture

• Computer system performance evaluation.
• Pipelining and hazards.
• Static/dynamic instruction scheduling and speculation.
• Memory hierarchy including cache and virtual memory.
• Snooping and directory-based cache coherence.
• Synchronization and memory consistency.