CS-101 Programming

Objective: This course is the first in a sequence of three courses on programming. A succesful student would have learnt to write code for non-trivial problems by learning the appropriate abstractions and paradigm, namely functional. A subsequent step is the transformation of these into the imperative form.

- Standard Constructs: Function and type definition, block structure, Guarded equations, pattern matching, Special syntax for lists, comprehension.
- Standard Data Types: Fluency is to be achieved in the standard data types: numbers, Boolean, character, tuple, list, List programs in an algebraic vein, Lists in the context of general collections sets, bags, lists, and tuples.
- Calculus: A direct way for denoting functions.
- First-Class-ness: All values are uniformly treated and conceptualized.
- Higher Order Functions: Use of first class, higher order functions to capture large classes of computations in a simple way, an understanding of the benefits that accrue modularity, flexibility, brevity, elegance.
- Laziness: The use of infinite data structures to separate control from action.
- Type discipline
- Polymorphism: The use of generic types to model and capture large classes of data structures by factorizing common patterns.
- Inference: The types of expressions may be determined by simple examination of the program text, understanding such rules.
- User defined types: User defined types as a means to model, a means to extend the language, a means to understand the built in types in a uniform framework.
- Concrete types: Types are concrete. i.e. values that are read or written by the system correspond directly to the abstractions that they represent. More specifically, unlike abstract types which are defined in terms of admissible operations, concrete types are defined by directly specifying the set of possible values.
- Recursion: Recursive definitions as: a means of looping indefinitely; a structural counterpart to recursive data type definition; a means to understand induction in a more general framework than just for natural numbers;
- Operational Semantics: Functional programs execute by rewriting, Calculus as a rewriting system, Reduction, confluence, reasons for preferring normal order reduction.
- Type Classes: Values are to types as types are to classes. Only elementary ideas

The imperative paradigm is smoothly introduced as follows:

Worlds	The Timeless Worlds	World of time
Domain	Mathematics	Programming
Syntax	Expressions	Statements
Semantics Explicit	Values Data Structure	Objects Control structure
Thinking with	Input-Output relations	State change
Abstractions	Functions	Procedures
Relation	Denote programs	Implement functions

In the following we spell out some of the points of how FP translates into Imperative P. The examples may be analogized from say how one would teach assembly language to someone who understands structured programming.

- Semantic relations: The central relation is that imperative programming's denotational semantics is FP, FP's operational semantics is imperative programming.
- Operational Thinking: In FP data dependency implicitly determines sequencing whereas in Imp P it is done explicitly. Advantages and disadvantages of operational thinking.
- Environment: In imperative programming there is a single implicit environment memory. In FP there are multiple environments; which could be explicit to the point of first class-ness (the value of variables bound in environments could be other environments). Use of environments to model data abstraction, various object frameworks, module systems.
- Recursion iteration equivalence: General principles
- Type Issues: Monomorphic, polymorphic and latent typing: translating one into another.

Language(s) to convey these two paradigms could be Gofer (or Haskell), Python, Scheme, etc. and can be expected to vary across time as better languages are developed for pedagogic purpose.

References:

- Introduction to Functional Programming, Bird and Wadler.
- Algebra of Programs, Bird
- Structure and Interpretation of Computer Programs, Abelson and Sussman
- Scheme and the Art of Programming, Friedman and Haynes

- Equations Models and Programs, Thomas Myers
- Algorithms + Data Structures = Programs, N Wirth
- Functional Programming, Reade
- Programming from First Principles, Bornat
- Discrete Math with a computer, Hall and Donnell
- Learning Python, Mark Lutz

CS-102 Systems

Objective This course is the first in a sequence which aims at getting an understanding as to what constitutes the various hardware subsystems of a (simple) modern computing device as well as software subsystems that are imperative in order to make effective (and efficient) use of a computing system.

Hardware subsystem

- From a calculator to a stored-program computer: Internal structure of a calculator that leads to this functionality. Machine language and programs writing a sequence of instructions to evaluate arithmetic expressions. Interpreting the computer's behavior when instructions are carried out: the fetch- decode-execute cycle as the basic or atomic unit of a computer's function. Control unit: that performs the fetch-decode-execute cycle.

- Basic Electronics: combinational functions and their implementation with gates and with ROM's; edge-triggered D-flip-flops and sequential circuits; Implementation of data-path and control, using the basic ideas developed so far.

- Parts of a computer: Processor (CPU), memory subsystem, and it's interfaces, peripheral subsystem. and it's interfaces. Parts of these interfaces integrated with the processor, and the remainder contained in the chip-set that supplements the processor. Two main parts of the processor apart from these interfaces: data-path and control (which supervises the data-path) .

- Introductory Machine: Modern computer design, dating back to the 1980's, marks a radical shift from the traditional variety. The new style has given rise to reduced instruction set computers (RISC), as opposed to the older complex instruction set computers (CISC). The ISA of one of the ARM family of processors will be examined. Assembly Language structure, syntax, macros, assembling and disassembling, clock cycle counting

- Pipelining: Improving the performance of a computer and increasing the usage of its subsystems by executing several instructions simultaneously. Analogy to assembly line manufacture of cars. Influence of instruction set design on ease of pipelining. Difficulties with pipelining: structural, data and branch hazards. Branch prediction

- Memory hierarchy: Performance tradeoffs: fast, small, expensive memories (static RAM); slower, larger, inexpensive memories (DRAM); very slow, very large and very cheap memories (magnetic and optical disks). Ideal memory: fast, inexpensive, unbounded size. Ways of creating illusions or approximations of ideal memory. On-chip and off-chip cache memories

Software subsystem

Abstraction of the physical computing device with better properties in order to use it more effectively.

Multiplex multiple programs giving an illusion that each having view of their own processor, memory, etc.

- Memory abstraction: virtual memory, required hardware support and programming necessary to make it happen
- Exceptions, traps and interrupts, ISRs. Minimization and handling of non determinism
- Software subsystem API (system call) and ABI

- Start up scripts, Basics of protected mode and device drivers

References

- Computer Organization and Design, Patterson and Hennessey
- Computer Structures, Ward and Halstead
- Digital Design: Principles and Practices, Wakerley
- Modern Assembly language Programming with the ARM processor, Larry Pyeatt
- Guide to Assembly Language Programming, S P Dandamudi, Springer
- Art of Assembly, Randy Hyde
- Modern Operating systems, Andrew Tanenbaum

CS-103 Mathematics of Computation

Objective: Build the mathematical foundation towards construction and formal reasoning about programs, inferring program behaviour, models of computation and various models which appear natural in the world of programing.

- Logic: Propositional Calculus: Alternative styles: Boolean Algebra, truth tables, equational, deduction, Formal systems, Syntax and semantics, Proof theory and Model theory, consistency and Completeness of different systems.
- Binding Constructs: Abstraction of lambda, for all, program function etc. Free and bound variables, substitution. Common laws.
- Set Theory: Definitions, proofs, notations, building models
- Well formed formulae: Ordinary definition, refinement to types, necessity and limitation of computable type checking.
- Relations: 3 alternative views of foundations of relations: as Cartesian products, as Boolean function(predicates), as power set functions 3 basic types - equivalences, orders, functions
- Graphs: Definition and examples of graphs, Incidence and degree, Handshaking lemma, Isomorphism, Sub-graphs, Weighted Graphs, Eulerian Graphs, Hamiltonian Graphs, Walks, Paths and Circuits, Connectedness algorithm, Chinese Postman problem, Traveling Salesman problem
- Trees: Definition and properties of trees, Pendent vertices, centre of a tree, Rooted and binary tree, spanning trees, minimum spanning tree algorithms, Fundamental circuits, cutsets and cut vertices, fundamental cutsets, connectivity and separativity, max-flow min-cut theorem
- Planar Graphs: Combinational and geometric duals, Kuratowski's graphs Detection of planarity, Thickness and crossings
- Matrix Representation of Graphs: Incidence, Adjacency Matrices and their properties
- Coloring: Chromatic Number, Chromatic Polynomial, the six and five color theorems, the four color theorem

References:

- Logic for CS by Gallier
- Discrete Math by Tremblay Manohar
- Discrete Math by Stanat
- Laws of Logical Calculi by Morgan
- Computer modelling of mathematical reasoning by Bundy
- Predicate Calculus and Program Semantics by Dijkstra
- A Logical Approach to Discrete Math by Gries and Schneider
- Practical Foundations of Mathematics by Paul Taylor
- Conceptual Mathematics by Lawvere
- Logic in Computer Science: Modelling and Reasoning about Systems, by Michael Huth , Mark Ryan Cambridge Univeristy Press
- Introduction to Graph Theory, Douglas West
- Graph Theory, Robin Wilson
- Graph Theory with Applications, Bondy, J. A. & U. S. R. Murty [1976], MacMillan
- Graph, Networks and Algorithms, Swamy, M. N. S. & K. Tulsiraman [1981], John Willey

CS-104 Mathematics of Computation

Objective: Build the mathematical foundation towards construction of programs whose output has accuracy issues or issues of uncertainty. These outputs are a common consequence of the models used in engineering, physical and biological sciences as well as inaccurate representation of a class of numbers on a finite machine and it's attendant consequences.

- Matrix notation, matrix algebra, matrix operations and their geometric significance, inverse, transpose.
- Vector Spaces and subspaces, linear independence, basis, dimension, linear transformations, four fundamental subspaces
- Orthogonality: orthogonal vectors and subspaces, projections, orthogonal bases, Householder transform
- Eigenvalues and eigenvectors: Their significance, geometric interpretation, similarity transformation and eigenvalues
- Positive definite matrices: It's connection to optima and optimization, recognition of positive definite matrix
- Computing and floating point arithmetic, truncation error, round-off error and it's propagation
- (Numerical) solution of differential equation(s) (initial value problem and boundary value problem) leading to
- solution of difference equations towards error analysis of solution techniques.
- Numerical solution of linear equations using direct and iterative methods, computation of eigenvalues and eigenvectors
- Interpolation and approximation
- solution of non-linear equation(s), fixed point methods, order of convergence

References

- Unified introduction to Linear Algebra, Alan Tucker
- Linear Algebra, Serge Lang
- Elementary Linear Algebra, Howard Anton and Chris Rorres
- Numerical Methods for Scientists and Engineers, Chapra, TMH
- Elements of Numerical Analysis, Peter Henrici, John Wiley & Sons.
- Numerical Linear Algebra, Leslie Fox, Oxford University Press.

CS-105 Database Management System

Objective: Enable the student to appreciate the theoretical underpinnings of the relational model and hence enable them to excel in database design along with the basis which leads to the effective use of SQL.

- DBMS objectives and architectures: Data and models of data, concept of database and information.
- Relational model for data base design: Relational data model, Schema design, Normalization theory, functional dependencies, higher normal forms, integrity rules, Relational operators
- Query language: Relational calculus, relational algebra and SQL
- Database design issues: Query processing, indexing and performance of database sysetm, B-tree index structure, transactions and transaction processing.
- Challenges in database design: conflicting design forces, trade-offs and resolution thereof.

References:

- An introduction to database systems, C. J. Date
- Database Management Systems, Raghu Ramakrishnan, Johannes Gehrke
- Principles of Database Systems Vol. I & Vol II, J. D. Ullman
- Relational Database Index Design and the Optimizers by Tapio Lahdenm, Michael Leach

CS-201 Programming

Objective: This is the second course in the sequence on programming with the idea of constructing of programs by effective use of data organization while building on ideas picked up in the first course in this sequence. The course is woven with the idea of the dual worlds: algebraic and algorithmic, which leads to a smooth and powerful way to develop programs.

- ADTs and Views: (Algebraic) Formulation as recursive data types, data structure invariants, principles of interface design and it's reflection (algorithmic) storage representations, addressing semantics, maximizing abstraction using language features like macros, etc.
- Code: (Algebraic) Pattern matching based recursive definitions wherein exhaustive set of disjoint patterns correspond to total functions leading to runtime bug free programs, recursive code structures follow recursive data structures and it's reflection (Algorithmic) refinement of recursive definitions into iterative algorithms, techniques for improving algorithms like sentinel, double pointers, etc.
- [Control as data, loops], [Co-routines vs. subroutines, functions], [General framework for error handling, escape procedures, stack based software architecture]
- the case studies/examples for the above include but need not be limited to the following: Lists: Various types of representations. Applications: symbol tables, polynomials, OS task queues etc., Trees: Search, Balanced, Red Black, Expression, and Hash Tables Applications: Parsers and Parser generators, interpreters, syntax extenders, Disciplines: Stack, queue etc and uses Polymorphic structures: Implementations

References:

- Data Structures and Algorithms, Aho, Hopcroft and Ullman
- Data Structures, Kruse
- Structure and Interpretation of Computer Programs, Abelson Sussmann
- Functional Programming Henderson
- The Art of Programming Vol. 1. & Vol. 3, D. E. Knuth

CS-202 Programming

Objective: This is the third course of a sequence on programming where a student is exposed to the notion that a \fIcorrectly running program \fP is not the \fIbe all and end all\fP for an effective programmer and computer scientist. At the end of the course, a successful student should be able to design, analyze and prove termination and correctness of (efficient) algorithms to previously unseen/unknown problem specifications using the general principles covered including being able to model/view a new problem as one of the previously solved problems. In addition, the student should be able to appreciate the value/power of randomness/uncertainty in achieving effective/efficient solutions.

- Probability as a model of mathematical uncertainty: Sample space, events, probabilities on events, conditional probabilty, independent events, Bayes' theorem
- Random variable, or function defined on a sample space: Expectation of a r.v., expectation of a function of a r.v., variance, notion of a probability distribution, (cumulative) distribution function, some standard discrete and continuous r.vs.
- Jointly distributed r.vs, Conditional probability and conditional expectation
- Notion of efficiency, Big-Oh notation, it's use in expressing the efficiency of an algorithm, it's calculation for a given algorithm
- Divide and Conquer as an effective paradigm to decompose a given problem into problems of smaller size and then obtaining the solution to the original one as a composition of the solutions of the subproblems
- Sorting, Searching, Selection
- String processing: Knuth-Morris-Pratt Algorithm, Boyer-Moore Algorithm, pattern Matching.
- Graph Algorithms: DFS, BFS, Biconnectivity, all pairs shortest paths, strongly connected components, network flow: Edge Saturation and Node Saturation Algorithms, Maximum Matching on Graphs
- Backtracking, Dynamic Programming, Branch & Bound, Greedy: Use of three paradigms for the solution of problems which invlove optimization or exploration of a search space with some specific property of the desired solution
- Computation steps being decided by the toss of a coin, or randomized algorithms:An introduction with a few examples and analysis
- Introduction to the theory of NP-Completeness: Non-Deterministic Algorithms, Cook's Theorem, clique decision Problem, Node cover decision problem, chromatic number, directed Hamiltonian cycle, travelling salesman problem, scheduling problems.

References:

- Introduction to Probability, John Freund
- Introduction to probability theory and it's Applications, William Feller
- A first course in Probability, Sheldon Ross
- Introduction to Algorithms, Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein,
- Algorithms, Robert Sedgwick
- The Design and Analysis of Computer Algorithms, A. V. Aho, J. E. Hopcroft, J. D. Ullman
- Algorithm Design: Foundations, Analysis, and Internet Examples, Michael T. Goodrich, Roberto Tamassia
- Algorithm Design, Kleinberg and Tardos
- Combinatorial Algorithms (Theory and Practice) , F. M. Reingold, J. Nivergelt and N. Deo

CS-203 Systems

Objective This course is the sequel to course CS-102 and hence it's goal is dovetailed in the same direction as CS-101.

- program in execution --> process abstraction
- lighter verision of a unit of computation --> thread,
- multiple such units --> concurrency, sharing and synchronization,
- software support and requirements to achieve these effectively
- Requirements and mechanisms to reach the stage \fIprogram in execution\fP

- The four dimensions of a programming activity as the basis for systems programming: concept, program generators (humans or other programs), sources and deliverables. For a variety of concepts, a set of program generators generate a set of (possibly overlapping) sources and produce a set of deliverables (executables, libraries, documentation).

- Interpretation as the fundamental activity in Software. Interpreters and interpretation. Program layout strategies on a Von Neumann machine (e.g. Pentium). Division of the final interpretation goal into subtasks and establishing interface export by producer tool and import by consumer tool. Compiler and Assembler translation phases
- Linkers and Loaders Linker as a layout specifying producer and loader as a layout specification consumer. Layout specification strategies: fixed and variable (relocatable and self-relocatable). Layout calculations. Dynamic linking and overlays. Executable format definitions. Object file format as the interface between the compiler and the linker. Few Object file formats like MSDOS, Windows and ELF. Object file manipulation utilities. Editors, version controllers. Version control on object and executable files (e.g. Version support for modules in the Linux kernel)
- Support tools: source browsers, documentation generators, make, GNU auto-conf, git, bug reporting systems, build systems. Debuggers for analysis. Package builders, installers, package managers for deployment

- Persistent storage: Storing and retrieving data (logically coherent from the viewpoint of the producer of the data) with an assocaited label/name --> file

- collections of files --> file system

- Organizing the collection of names/labels --> directory structure

- File layout
- Managing free space
- Process and file interaction
- Issues in robustness of filesystem
- Designing and organizing metadata for performance, adaptive to encompass newer technology like SSD

References

- Modern Operating Systems, Andrew Tanenbaum
- Design of the Unix Operating System, M, J, Bach
- Compilers: Principles, Techniques and Tools, Aho, Lam Sethi
- John Levine, Linkers and Loaders, John Levine

- System Software: An Introduction to Systems Programming, Leland L. Beck

CS-204 Systems

Objective: This is the third course in the collection on systems. At the end of the course a successful student should be able to appreciate the issues involved in developing a software subsystem in which the primary activity is communication of data across widely differing geographical scales and with very little hold on the variety of software/hardware combinations of which this subsystem is a part. All this has to be achieved in the context that the entire underlying communication setup cannot be assumed to be reliable.

- Understand the nature of layering and it's naturalness in the design
- Examine the assumptions of the underlying communication setup (hardware/technology)
- Physical Layer, Transmission media, digital transmission, transmission & switching
- Data link layer and it's interface with the physical layer below and the network layer above, a look at Ethernet and IEEE 802.11
- Network layer: the service it is expected to provide and how it is achieved in the context of IP, SDN: an introduction
- Transport layer: the service it is expected to provide and how it is achieved in the context of TCP
- Application layer: Sitting at the top of the network stack, the service it is expected to provide and some sample protocols under the TCP/IP protocol suite
- The Session and presentation layers which have utility in certain contexts, primarily in the telecom sector.
- Network Security and how it should be integrated into the system
- the place and interaction of this subsytem in the larger context of the software/hardawre system of which this a part and whose objective is to make effective and efficient use of the computing device
- Communication protocols at the small scale: communication at the processor level

Reference:

- Data and communications, W. Stallings
- Computer networks: A systems approach, Peterson and Davie
- Computer Networks, A. S. Tanenbaum
- UNIX Network Programming, Stevens
- TCP/IP Illustrated Volume 1, 2, 3,Richard Stevens

CS-205 Theory of Computing

Objective: Enable the student to get a fair understanding and appreciation of the power as well as the limits of computation.

- Low Power Formalisms Combinational Machines inadequacy
- FSM as acceptor, generator, regular expressions and equivalence
- PDA brief idea, relation between CFG's and programming languages (informal)
- Full Power Mechanisms: (i) Recursive functions, (ii) Turing machines cost models for the RAM, (iii)Post systems/Lambda Calculus/Markov algorithms, (iv) (any one) Use must be stressed along with mutual equivalences. Any of the (iii) should be done so as to give a theoretical backing to the practical notion of 'nonVon-Neumann' language.
- Recursive, Enumerable sets, generators and recognizers
- Universal Algorithm; Diagonal Results: Universal Turing Machine, Uncomputability; Goedel Theorem
- Complexity Basic ideas measuring time usage, time hierarchies Deterministic and Nondeterministic computations.
- Randomness in Computing: Randomness and Complexity, Pseudo-randomness, Cryptography.

References:

- Introduction to the Theory of Computation, Michael Sipser
- Introduction to Computer Theory, Daniel Cohen
- Computability And Complexity From a Programming Perspective, Neil Deaton Jones
- Introduction to Recursive Function theory, Nigel Cutland
- Foundations of Cryptography, Oded Goldreich