Course Information

Course structure for Spring 2001

The course is an informal survey of programming languages, focusing on programming languages concepts such as binding, evaluation, and types, exhibited on representative languages from the functional and logic programming paradigms.

The goal is to establish a working knowledge of several languages and to practise the craft of programming in different paradigms. The course will be fast-paced, and will involve individual reading of language specification documents, and extensive programming assignments. Previous acquaintance with the procedural and functional paradigm is assumed, as described below.

Requirements

Official class announcements will be sent by email. You must therefore have an email address. Papers and software will be provided within the UNM CS file system. You must therefore have a UNM CS account.

Assignments

Four midterm exams, final exam (covering the entire course), about 10 written homework assignments, 2 or 3 programming projects.

Prerequisites in detail

Experience with developing substantial applications in functional and imperative (especially object-oriented) programming languages. UNM CS courses CS 257L - *Nonimperative Programming* and CS 351L - *Design of Large Programs* provide appropriate background in Scheme and C++, respectively.

Lectures

Mondays & Wednesdays, 4:00 - 5:15, in Tapy Hall 218

Instructor

Darko Stefanovic, office FEC 345C, phone 2776561, email darko@cs.unm.edu — office hours Mondays 5:30-6:30, Tuesdays 11-12, Wednesdays 8:15-9, or by appointment

Textbooks

(The bookstore has ordered the titles marked with *.)

Required reading

*Jeffrey D. Ullman: *Elements of ML Programming, ML97 Edition*, Prentice Hall, 2000, ISBN 0-13-790387-1.

*Richard Bird: *Introduction to Functional Programming using Haskell*, Prentice Hall, 1998, ISBN 0-13-484436-0.

*Matthias Felleisen and Daniel P. Friedman: *A Little Java, a Few Patterns*, MIT Press, 1998, ISBN 0262561158

Optional reading

Laurence C. Paulson: *ML for the Working Programmer, 2nd edition*, Cambridge University Press, 1996, ISBN 0-521-56543-X.

Paul Hudak: *The Haskell School of Expression*, Cambridge University Press, 2000, ISBN 0-521-64408-9.

W. F. Clocksin and C. S. Mellish: *Programming in Prolog, 4th edition*, Springer Verlag, 1994, ISBN 3540583505.

Jon Bentley: *Programming Pearls, 2nd edition*, Addison-Wesley, 2000, ISBN 0-201-65788-0.

To probe further

Krzysztof R. Apt: *From Logic Programming to Prolog*, Prentice Hall, 1997, ISBN 0-13-230368-X.

Richard Bird and Oege de Moor: *Algebra of Programming*, Prentice Hall, 1997, ISBN 0-13-507245-X.

H. P. Barendregt: *The Lambda Calculus: Its Syntax and Semantics, revised edition*, Elsevier North-Holland, 1984, ISBN 0-444-87508-5.

David A. Watt: *Programming Language Concepts and Paradigms*, Prentice Hall, 1990, ISBN 0-13-728874-3.

Michael J. C. Gordon: *Programming Language Theory and its Implementation*, Prentice Hall, 1988, ISBN 0-13-7304170-X.

Anthony J. Field and Peter G. Harrison: *Functional Programming*, Addison-Wesley, 1989, ISBN 0-201-19249-7.

Shriram Krishnamurti, Matthias Felleisen, and Daniel P. Friedman: "Synthesizing Object-Oriented and Functional Designs to Promote Re-use", in *ECOOP'98*, Springer-Verlag LNCS 1445, pp. 91-113, 1998.

Other reading material (required reading, available for free)

At http://www.haskell.org/definition/ you will find the following:

Simon Peyton Jones and John Hughes (Eds.): *Report on the Programming Language Haskell 98*.

Simon Peyton Jones and John Hughes (Eds.): *Standard Libraries for the Haskell 98 Programming Language*.

Mark P. Jones and John C. Peterson: *Hugs 98, A functional programming system based on Haskell 98, User Manual*.

Grading

You are expected to attend class regularly, read the assigned reading before class, and participate in class discussion. The grade will be determined as follows:

Homeworks 30%

Programming projects 20%

Exams 56% (9% each of 4 midterm exams, 20% final)

The excess 6% is intentional: you can do a little worse on any one grading component and still receive the top grade.

Homework and programming assignment hand-in policy

Homework assignments are due on the date assigned, no extensions will be granted, and no credit will be given for late homework. Hard copy solutions must be handed in either in class, or during office hours on the due date. Late programming projects will be penalized $3n^3$%, where n is the number of days late.

Programming questions on homework assignments (also applies to programming projects)

When an assignment asks you to write a program, that means that you must design a program, type it into a computer, compile (depending on the language at hand), and run, and submit a listing of the program and its output. The textual layout of the program must be logically sound and aesthetically pleasing. The names used must be descriptive. Code comments and additional documentation must accompany non-trivial programs, and must provide informal argumentation that the program satisfies the specification. (Occasionally you will be asked to provide a formal correctness proof.)

Cooperation and cheating

Feel free to *discuss* homework and projects with classmates, and the instructor. However, *do not look at or copy another student's solution to a homework or project*. If a problem appears too difficult, or you lack the background to solve it, you are expected to talk to the instructor promptly. Once you have the background necessary to solve a problem, you must provide your own solution. Exchanging homework or project solutions is cheating and will be reported to the University administration; students involved will not be permitted to continue in the class. Occasionally we will reuse problems that have been used in previous courses. Searching on the web for verbatim solutions is not the best application of your time, and you will receive no credit for it.

Lecture Plan Overview

- Background material review (1 lecture, at first class meeting)
- Prolog; logic programming (2 lectures): elementary examples in logic programming and purely declarative Prolog, informal account of unification
- Untyped functional programming: Scheme review (1 lecture): salient features of Scheme, elementary examples in untyped functional programming, recursive function definition, higher-order functions, currying
- Untyped functional programming: elementary lambda calculus (1 lecture): informal introduction to lambda calculus, purely syntactic interpretation, notions of conversion and reduction, elementary programing examples
- Mid-term 1 (1 class period)
- Typed functional programming: ML: types (5 lectures): ML syntax, SML/NJ usage, type discipline in general, structure of recursive data types, type inference, elementary programing examples, informal treatment of the module system
- Mid-term 2 (1 class period)
- Typed functional programming: Haskell: type classes; program synthesis (6 lectures): Haskell syntax, Hugs usage, the type system of Haskell, algebraic structure of programs, program development by systematic transformation
- Mid-term 3 (1 class period)
- Typed functional programming: Haskell: lazy evaluation; monadic programming (4 lectures): exploiting lazy evaluation to program streams and other infinitary computations, monadic treatment of input-output
- Mid-term 4 (1 class period)
- Object-oriented programming with patterns: Java: (4 lectures): connections between object-oriented and functional programming, data types and class hierarchies, type classes and interfaces, syntax and usage of Java, syntax and usage of Pizza

- Review (1 class period)
- Final exam

Detailed Lecture Plan

Lecture 1: Background material review. Mathematical background. Propositional logic. First-order predicate logic. Proving with predicates. Mathematical induction. Program verification. Formal languages. Language classes. Grammars. Automata. Computability. Decidability. Translation. Parsing and scanning. Compilation. Interpretation. Abstract machines (virtual machines). Hardware architecture.

Lecture 2: Logic programming. Relation between logic and programming in Prolog. Syntax of Prolog; facts, rules, and queries. Prolog inferencing.

Lecture 3: Prolog examples with list constructors and recursion. Append.

Lecture 4: Scheme review. Examples of code in Scheme. Drawn from CS257.

Lecture 5: Elementary lambda calculus. Informal introduction. Syntactic manipulation of lambda expressions. Notions of conversion, reduction, normal form, reduction order. Church numerals.

Lecture 6: ML. Usage. Syntax. Types. Built-in type constructors. (Ullman Ch. 1–5)

Lecture 7: ML. Parametric polymorphism. Type inference.

Lecture 8: ML. Datatypes. Pattern matching. (Ullman Ch. 6; 7.1)

Lecture 9: ML. Various examples in the ML core language.

Lecture 10: ML. Module system. (Ullman Ch. 8)

Lecture 11: Haskell. Usage. Syntax. Simple functions. Types. Lazy evaluation. (Haskell 98 Report, Hugs 98 Manual, Bird Ch 1–2)

Lecture 12: Haskell. Type system. Type classes. Recursive datatypes. Recursive function definitions. Proofs by structural induction. (Bird Ch. 3)

Lecture 13: Haskell. List comprehensions. Programming with lists. (Bird Ch. 4)

Lecture 14: Haskell. More programming with lists. (Bird Ch. 5)

Lecture 15: Haskell. Programming with trees. (Bird Ch. 6)

Lecture 16: Haskell. Abstract datatypes and modules. (Bird Ch. 8)

Lecture 17: Haskell. Lazy evaluation, streams, and infinitary structures. (Bird Ch. 9)

Lecture 18: Haskell. More lazy evaluation, streams, and infinitary structures.

Lecture 19: Haskell. Monadic programming style. (Bird Ch. 10)

Lecture 20: Haskell. More on monadic programming style.

Lecture 21: Object-oriented programming and patterns. (Felleisen and Friedman Ch. 1–4).

Lecture 22: Object-oriented programming and patterns. (Felleisen and Friedman Ch. 5–6).

Lecture 23: Object-oriented programming and patterns. (Felleisen and Friedman Ch. 7–8).

Lecture 24: Object-oriented programming and patterns. (Felleisen and Friedman Ch. 9).

Date and location TBD, during finals week: FINAL EXAM.

Note: additional class meetings may be called during the semester, if the need arises.