CPSC 499 # Functional Data Structures *UNBC*

Winter 2004

Prerequisites: A C$^-$ in CPSC 281 and CPSC 370 and all of their pre-requisites, or permission of the instructor.

Overview: CPSC 482-3 DATA STRUCTURES II has not yet been taught, and the calendar description needs revision. This special topics course also focusses on data structures, but looks at them in the context of the functional programming paradigm. In pure functional languages explicit modification of *parts* of a data structure is disallowed, and instead one must copy the entire object modified. Trees continue to work well in the functional paradigm, with $O(\log n)$ modification times, but straight-forward adaption of array-based algorithms from imperative languages to functional languages is disasterous as the cost of array modification goes from $O(1)$ to $O(n)$.

This course introduces modern functional programming languages (STANDARD ML, and maybe HASKELL or SCHEME). It then looks at techniques to implement purely functional analogs of classical imperative-language data-structures. Many of these techniques, (such as overeager evaluation and amortized cost analysis) are also applicable to strictly imperative approaches to data structures.

Professor: Dr. David Casperson **Office:** Lib 5-471 **Telephone:** 960-6672
e-mail: `casper@unbc.ca` **web address** `http://web.unbc.ca/~casper`

Text: *Purely Functional Data Structures* by Chris Okasaki.

Grading: (subject to revision)

Homework	:	15%
Exam 1	:	25%
Exam 2	:	25%
(Final) Exam 3	:	35%

Lecture times: M W F 11:30–12:20. **Room** 5-158.

Syllabus: My main goal is to discuss functional programming, using in particular STANDARD ML (and if time permits HASKELL). Although I intend students to acquire a reasonably good grasp of functional programming technique, the major emphasis will be to explain why functional programming requires different data structure and techniques than imperative programming languages. Topics will be chosen from among the following.

An introduction to functional programming. Static versus dynamic typing. Strict versus non-strict evaluation. Some common functional programning languages. "Pure" versus "impure" functional programming.

Set theory review from CPSC 141. Functions and partial functions. Cartesian products. Disjoint unions. The notion of Currying.

Standard ML. Builtin types and literals. Tuples. Lists. Declarations. Function declarations and function values. Product types and function types.

Pattern matching in function declarations and function values. Case statements. `datatype` declarations.

Recursion, tail recursion, and accumulator arguments. Higher order functions for lists.

Space and time complexity for functional programs and data structures.

Persistant and ephemeral data structures.

Amortized analysis. Lazy evaluation. The use of lazy evaluation to achieve amortization in persistent data structures. Imperative implementation of functional data structures. Strategies for re-adjusting time complexity. Overeager evaluation.

Case studies. Leftist heaps, binomial queues, and red-black trees. Functional analogs of imperative data-structures. Streams, queues, and arrays.

Non-strict evaluation and Haskell. Monads and other strategies for dealing with non-functional programming elements.