Title: CSci 2041 Advanced Programming Principles

4 credits

Catalog Description:
Principles and techniques for creating correct, robust, and modular programs. Computing with symbolic data, recursion and induction, functional programming, impact of evaluation strategies, parallelism. Organizing data and computations around types. Search-based programming, concurrency, modularity. Practical application. Weekly labs.

3 hours per week and 1 hour lab

Offered Every semester

Component 1: LEC (with final exam)

Prerequisites: (CSci 1913 or CSci 1933) and CSci 2011

Student Learning Outcomes:
* Student in the course:
- Can identify, define, and solve problems

Please explain briefly how this outcome will be addressed in the course. Give brief examples of class work related to the outcome.

The class is focused on principles that underlie the structure and analysis of programs. Students will learn different programming styles, such as those based on functional programming, search-based programming, and concurrent programming, and will learn to program over symbolic structures. Underlying the course will be two or three medium sized programming projects where students will learn about modular development of programs and language principles to support modularity.

How will you assess the students' learning related to this outcome? Give brief examples of how class work related to the outcome will be evaluated.

This SLO will be assessed through labs, homework, and exams. Each of these types of students' work is problem-based; most involve open-ended problems where students need first to identify, define, and/or clarify what the problem is before solving it, and explain the solution they propose.

Sample syllabus

This is a required course for computer science majors that is to be taken at the end of the sophomore year or the beginning of the junior year. The course will use a functional language to introduce a high-level approach to programming over complex data. It will emphasize a view of such data that abstracts away from their representation, using types as a vehicle for organizing them as values and for structuring computations over them.

Advanced programming techniques that use ideas such as recursion, higher-order functions, lazy and eager forms of evaluation and infinite data objects will be explored. Exploitation of parallelism arising from pure forms of expression evaluation will be examined. Other techniques and principles to be studied include search-based programming, modularity and concurrency. Programming projects that focus on symbolic computation will be used in a central way to impart the core ideas in the course; such projects may include writing parsers, type-checkers and interpreters for suitably circumscribed programming languages, and applications of search-based techniques.

Why This Class is Important and its Role in the Curriculum

Proficiency in programming is based on the ability to focus principally on the essential structure of data and computations over them, abstracting away from representational and architectural issues. By using a language that is liberated from the von-Neumann view, students will be exposed to this higher-level view of programming. This setting will also provide an easy way to explore varied programming techniques that can be effective in imperative and object-oriented languages as well but that are harder to introduce directly in such a context. Students will understand how to treat structurally complex objects such as programs as data, a topic that is intrinsic to several theoretical and practical courses later in the curriculum. This is a required course for CSci majors.

Prerequisites and Rationale

CSci 1913 or 1933: Students need to have the degree of programming experience and maturity obtained from completing one of these courses.
CSci 2011: 2041 builds on ideas from this course such as induction, recursion/recurrences, and logic.

Classes Having 2041 as a Prerequisite and Rationale

CSci 3081: 2041 covers topics that are essential to principled and effective programming. 3081 is a natural follow on to this course in that it demonstrates the use of the principles and techniques studied in 2041 in practical, collaborative software development.

CSci 4011: 4011 covers the mathematical aspects of viewing linguistic entities such as programs as objects. Familiarity with the treatment of symbolic data is important to getting started with such a study. The treatment of recursion, induction and types will also build the mathematical needed in 4011.

CSci 4511: Exposure to a functional language and symbolic computation
in 2041 will be used in 4511.

Class Format

4 credits, 3 hour long lectures and 1 hour long lab per week. Lab sessions are used for hands-on learning, experimentation with programming techniques in a guided setting, and discussion of additional examples. Practice in lab sessions is aimed to prepare students for larger projects done outside of class time.

Possible texts

Elements of Functional Programming by Chris Reade; Programming in Standard ML by Bob Harper; Purely Functional Data Structures by Chris Okasaki

Outcomes

By taking this course, students will be able to:

* write and understand functional programs
  - that use different evaluation strategies, understand impact on programming techniques and efficiency
  - that use functions a first-class values
  - that are recursive over complex symbolic data and that especially treat programs as data

* understand the relevance of types to organizing data and to structuring programs over such data

* understand the distinction between data and their representation and learn to focus programming thinking around values rather than their ultimate computer realization

* use inductive principles to reason about recursive programs

* understand how parallelism can exposed by not introducing unnecessary dependencies between different computations especially by avoiding interactions through the use of state

* analyze the complexity of recursive programs over immutable data, and relate this to iterative programming over state

* understand principles of search-based programming, modularity and concurrency

* understand how these varied principles can be translated into practice and especially see their use in programming in non-functional languages and settings

Outline of material covered

The topics to be covered in the course are described below. This is not intended as a week-by-week schedule: material under different topics will be interleaved and reordered in an actual plan for the course.

- Types as an organizing principle for programming. Types as a language, higher-order and polymorphic types, types as means of classifying values, ad hoc versus parametric polymorphism.

- Expressions and computation as effect-free evaluation. Binding of names, scoping, environments, closures; strict and non-strict evaluation, opportunities to exploit parallelism; lazy evaluation as a programming technique and infinite data structures; recursive functions and relation to recursive data; iteration as tail recursion, translating general recursion to tail recursion.

- Recursion and relation to inductive reasoning, invariants over functions, types and invariants, designing functions around invariants.

- Functions as first class objects, higher order functions (map, filter, fold) and applications, parametric polymorphism, functions as parameters, continuation passing style.

- Effects and computation. Type safe references, assignments, other side-effecting constructs, iterative control structures; modelling effectful computation via state transforming functions, effects in lazy languages (monads); object oriented programming as combining environments with state; references and circular data structures.

- Programs and analysis of complexity. Recursive functions and recurrence relations; functional data structures, efficiency and programming techniques; mutable data and efficiency.

- Value-based programming and realization. Mapping data objects to memory; memory usage, copying versus pointing; garbage creation and automatic collection, memory management.

- Search-based computation. Search as a computational paradigm and its applications; programming techniques for realizing search.

- Role of modularity in programming-in-the-large. Interface specifications, abstract data types; language support for modular programming, interface checking as type checking; module composition as function application.

- Concurrency. Asynchronous computation as a paradigm, coordination through communication; language mechanisms for organizing and controlling communication.

- Translation of principles into programming in mainstream, non-functional languages.