CS 381: Final Exam Review

Functional programming

Essential terminology: **type**, **expression**, **value**, **function**

Kinds of questions:

- what is the type of a given Haskell expression?
- what is the result of evaluating a given expression?
- create a value of the indicated type
- implement a function given an informal specification

See: Homework #1, Homework #2, Quiz #1, Midterm, practice problems

Syntax

Essential terminology: **grammar**, **abstract syntax**, **concrete syntax**, **AST**, **syntactic sugar**, **object language**, **metalanguage**

Kinds of questions:

- what sentences are generated by a given grammar? from which nonterminal?
- encode a grammar as a Haskell data type
- encode the AST for an object language term as a Haskell value

See: Homework #3, Quiz #2, Midterm, practice problems

Denotational semantics

Essential terminology: **semantic domain**, **semantic function**

Kinds of questions:

- identify the best semantic domain from a syntax + informal spec
- given a syntax + semantic domain, implement the semantic function

See: Homework #4, Homework #5, Midterm, practice problems

Type systems

Essential terminology: **type**, **static typing**, **dynamic typing**, **typing relation**

Kinds of questions:

- what are some benefits of static typing?
- given a spec + syntax, what is an appropriate representation of types?
- given a syntax + types, implement the typing relation

See: Homework #5, Quiz #3, practice problems

Naming and scope

Essential terminology: **name**, **declaration**, **binding**, **reference**, **shadowing**, **dynamic scope**, **static scope**, **environment**, **closure**

Kinds of questions:

- identify the declarations and references in a C or Haskell snippet
- what declaration does a name reference?
- evaluate an expression with dynamic vs. static scoping

See: Homework #5, Quiz #3, practice problems

Parameter passing

Essential terminology: **call-by-value**, **call-by-name**, **call-by-need (lazy)**, **call-by-reference**

Kinds of questions:

- how many times will an argument be evaluated under each scheme?
- what are the properties and tradeoffs of each scheme?

See: Homework #5, practice problems (for Quiz #3)

Logic programming

Essential terminology: **atom**, **predicate**, **variable**, **goal/query**, **database**, **fact**, **rule**, **rule head**, **rule body**, **goal search**, **unification**, **cut**

Kinds of questions:

- what is the result of a given unification problem?
- given a database, what are all solutions to a given query?
- given a database, write a query that returns a particular result
- given a spec, define a predicate using facts and rules
- given a database, will goal search terminate for a particular query?

See: Homework #6, practice problems (to be posted)