CS 3120 Final Exam review

Chapters 7,8, 15, Haskell, 9, activation records 10, ch 12, ch 14 just understand the concept of a logic language (prolog)

A good place to start studying is with the homework assignments. I also recommend looking at the Review questions after each chapter. For programming, we will stick with Haskell. (no Python)

Chapter 7 Expressions and Assignment statements

* Precedence rules with expressions. Know how to evaluate with given rules in right left, left right order.
* What is a side effect? (referred to in Ch 7 and ch 9). Be able to give an example of one.

Pointers, access outside of scope

* What is referential transparency – listed here, chapter 9 and chapter 15
* What does it mean if an operator is overloaded? Give an example

object1 + object2 = object.someInt = object2.someInt

* What is data type coercion, narrowing and expanding? Be able to give an example

int to float (expanding) float to int (narrowing)

* What is short circuit evaluation?
* Know how pre and post operators work in C++ (cnt++/++cnt)

Chapter 8 Statement Level Control Structures

* What are the design issues for the various control structures? Be able to name a few for each one.
  + Two way selection statements and nested selections– give and example, what are design issues?

If else. Haskell says you have to have an else.

* + Multiple selection statements- design issues and example/issues

switch statement

* + Loops counter controlled – design issues. What does C++ allow as opposed to other languages? Be able to explain all the parts – loop control variable, update, etc
  + Logic loops, do while and while. Be able to give an example in any language
* What is a user control loop mechanism

breaks, continues

* Why are gotos considered unsafe?

Chapter 9 Subprograms

* Basic definitions concerning subprograms – be able to give an example or point out parts
* What is the difference between a procedure and a pure function?

Pure funcion will return a single value

* Again side effects, give an example of one, why are they to be avoided?
* Parameters - Terminology: **actual, formal, keyword, default**
* Local variables static or dynamic, stack or static
* Parameter passing: know the difference between and be able to give an example of each: pass-by-value, pass-by-result, pass-by-value-result (copied in/out), pass-by-reference (skip pass by name). Be able to explain how side effects might occur with some of these.
* Overloaded subprograms – what does this mean? How can the compiler tell which implementation should be called?
* Generic subprograms – what are these? Why are they useful? How are they implemented in C++?
* User defined overloaded operators – covered this previously – what are they?
* Design issues for functions

Chapter 10 activation records – section 10.1-10.3

pages 450-462

* Support for subroutines – what needs to happen?

Save environmental pointer.

Dynamic link

Copy param values or access link(for pass by reference)

local variables

return value (if not void)

* What is an activation record? What is in it? Be able to draw one given a function and its call. Remember dynamic links.

Chapter 15 Functional Languages

* Haskell
* What is a functional language, how does it differ from an imperative language? Compare and contrast.
* Referential transparency and recursion
* Haskell – be able to write functions that use lists, pattern matching, list comprehension, if then else, see slides.
* What is a function type signature? Be able to write one for a simple function.

Chapter 12 Support for Object Oriented Programming

* What are the advantages of o-o languages over imperative languages

Encapsulation, easier to maintain, polymorphism

* Inheritance – describe it, derived classes differences from parent class
* Type checking and polymorphism, what are these, issues?
* Support for o-o languages – public private
* Dynamic binding – how does it work?
* Abstract classes and pure virtual functions
* Support for o-o programming in C++ - look this section over, do not need to look at the other languages
* Compare and constrast oop and imperative languages