

CMSI 585

PROGRAMMING LANGUAGES

Fall 2004

Crib Sheet 2

This document summarizes the overall coverage of Quiz 2. It's actually more of a study guide than crib sheet, since the test will be open-paper-anything anyway.

As a general guideline, be familiar with how the constructs discussed look (and vary) in each of our major languages: Java, C/C++, Perl, and ML. It is fair game to ask you to write little snippets of code in any (or all) of these languages, or to ask you to read such code and interpret them.

Also, be familiar with any particularly interesting variations from other languages, as they came up in the discussion. Interesting outliers have emerged in such languages as Algol 68, Fortran (77/90), Ada, Common Lisp, and Smalltalk.

One final general guideline: think about issues of orthogonality throughout the topics: what it means in the context of control flow, types, and subroutines, and how orthogonality manifests in various language designs and implementations.

Chapter 6: Control Flow

Expressions and 99 ways to leave your "goto." Make sure you know...

- All about expressions. Notation (prefix, infix, postfix), operations, precedence, associativity, evaluation order. *Especially evaluation order outside of precedence and associativity (with or without parentheses)*. Did I say evaluation order?
- Variables: value vs. reference models, l-values and r-values, assignments.
- Boolean expressions: short-circuit evaluation.
- The origin and basis of *goto*, and the contexts in which it was used. Know about what constructs have since replaced it.
- Selection: if and case. Be familiar with the variations on each, and have an idea about how these are ultimately implemented.
- Loops: enumerated generation 1, logically-controlled, enumerated generation 2. Be familiar with the variations on each type of loop. Know about semantic issues behind each loop type — restrictions, options for exiting/terminating/iterating, short-circuit evaluation.
- Recursion: how they are implemented (ties in to Chapter 8), comparisons to iteration, optimizations through tail recursion.
- Non-determinacy: guarded commands, applications for non-deterministic code, issues relating to the selection of what option to execute.

Chapter 7: Types

Types provide a framework for defining, interpreting, and operating on information in a program.

While, in the end, "everything is just bits," we as human beings have infinite ways for interpreting these bits, and we like it that way. Make sure you know...

- Basic definitions for types: type system, type equivalence, type compatibility, type inference, type checking, type clashes, strongly typed languages, statically typed languages, dynamic type checking, type declaration vs. definition
- Perspectives on types: denotational, constructive, and abstraction-based

- The different “types of types,” what they are, what they mean, what implementation issues they carry with them; have a general idea of which languages support which types, and if there are any specific twists with how they handle those types
- Simple types: booleans, characters, numbers, enumerations, subranges, alias types
- Type checking in detail: type equivalence (structural vs. name), type compatibility, conversion/casting, coercion, and type inference
- Implementation issues arising from the assorted type checking approaches and mechanisms
- ML’s type inference system — how it works, what it can/can’t do
- Composite types: records/variants, arrays, strings, sets, lists, pointers and recursive types, streams — know what they are, know any variations on these types across languages (or even whether a language supports that type), and be aware of any implementation issues that accompany these types

Chapter 8: Subroutines and Control Abstraction Basics

While we didn’t get very far with subroutines, we all work with them everyday, so it’s fair to require some practical working knowledge on what they are, how to use them, and what they mean. Make sure you know...

- How a subroutine is implemented. Have some knowledge of the overall *calling sequence* — what happens when a subroutine is called. Have an idea of how the calling sequence looks when entering and exiting a subroutine.
- Memory-related issues with subroutines: stack allocation, local variables
- That’s all for subroutines, for now...