Getting *You* ready for CS 136

ISG Peeps

Winter 2009

What you need to know: In 30 Seconds

- Scheme should be no stranger to you, including common idioms like `local` and `lambda`.
- ADTs like stacks and tables.
- Functions for performing I/O in Scheme.
- Parsing S-Expressions, parsing in general using a FSM.
- Mutation in Scheme with `set!` and `begin`.
- Memoization.
- Mutable structures.
- Simulating objects in Scheme.
- The simple Memory Model for computers
- C basics.
- Types: Static vs Dynamic Typing
- Routine calls in C, the routine call stack.
- Iterative loop constructs.
- Processor Model and basic machine instructions.
- Efficiency in the sense of runtime and complexity: Order Notation.
- Recurrence Relations.
- More efficient lists in Braun Trees.
- Arrays in C.
- Sorting and searching.
- HashTables, hash functions.
- Reachability with graphs: depth-first search.
- Understanding pointers in C.
- Structs: linked lists, trees, queues

- Memory management with `malloc()` and `free()` and common problems you can run into when managing memory.
- Basics of file I/O in C.
- Enumerations.
- Safety offered by OOP over procedural programming.
- Pitfalls of O-notation, proving and disproving statments in order notation.

Scheme

You should know everything from the *Quick Introduction to Scheme* document. You should also be very comfortable with the idea of tail-recursion, be able to recognize when a function is tail-recursive, and write tail-recursive code. Know what the stack and table ADTs are, as well as the idea of an ADT and the seperation of an ADT from its implementation. For more Scheme functions: know the basics of I/O, and using it to parse input. Know S-Expressions, and how parsing of S-Expressions can be done, especially when using a FSM. You should be familiar with mutation in Scheme, and the concept of memoization, and why it is useful in functional programming languages. You should be able to write and use mutable data structures. Similarly, you should be able to simulate objects in Scheme.

Memory Model

You should know the simple "memory model" for computers i(4-byte words, etc), and how this applies to the basic usage of C. The idea of types, and the differences between static typing and dynamic typing should be familiar to you. You should understand the semantics of routine calls in C, including the call-stack and its relationship with the memory model. You should know how C commonly uses iterative looping constructs rather than recursion. You should also understand how arrays in C and vectors in Scheme relate to the memory model, and their pros and cons compared to lists.

Processor Model

You should be comfortable with the basic "processor model", and have an idea of what simple machine language instructions could be. You should be able to apply these in relation to the effeciency of programs that you write, both in terms of runtime speed and overall complexity. You should be familiar with using order notation to express the complexity of computer programs in both Scheme and C. This will lead to solving recurrence relations in order to find the complexity of a program, you should be able to do this. You should be able to reason about data structures that have lower complexity than the ones you are used to. For example, how Braun Trees are more efficient than lists for some tasks. You should know the common sorting and searching algorithms, and how to analyze their efficiency, both in Scheme and C. You should be familiar with Hash Tables and basic hash functions. You should understand why Hash Tables are useful in graph reachability using depth-first search. Lastly, you should be familiar with the pitfalls of using order notation when talking about the efficiency of programs.

C

You should be able to read and write C code, and be familiar with the common C routines like `malloc()`, `free()`, `realloc()`, and `printf()` on top of ideas like `struct`, `static`, `enum`, `typedef`. You should be able to build structs in C just like in Scheme. You should be somewhat familiar with file I/O in C. Knowing about seperate compilation and header files is also helpful.

Object-Oriented Programming

You should be familiar with the basic concepts of OOP. You should understand the safety features that object-orientation give the programming language.