15-122: Principles of Imperative Computation

Recitation 24 — Matt Dee

The Last ~~Lecture~~ Recitation

Below is a (non-exhaustive) list of testable topics covered this semester for the final exam. You can use this, in conjunction with the old exams that we have posted online at `http://symbolaris.com/course/pic14-resources.html#exams` to prepare for the final. I tried to match some topics with questions from old exams, but just because a topic does not have a link to it

THe final will be from 8:30 to 11:30 on Monday, May 5th 2014. Recitations A-G should go to McConomy, while recitations H-I should go to MM 103 in Margaret Morrison.

Before Midterm 1

- Contracts and Program Reasoning
  - The three basic contracts: requires, ensures, and loop invariants, and when they are checked.
  - How do we use these to prove correctness/safety of code? (4 steps in a proof)
  - See **Problem 4, f12 final solutions**
- Ints
  - Two's complement representation of ints; converting decimal to binary to hex for ints of any size.
  - Bitwise operations `&`, `|`, `~`, `<<`, `>>`, masking, and sign extension.
  - Arithmetic operations, when are they defined/cause errors in C and C0? When does overflow happen, and what does it do? Which operations respect modular arithmetic?
  - Using ints to represent other things, such as pixels.
  - See **Problem 1, s12 final solutions**
- Arrays/Safe Access
  - How do we allocate a C0/C array? What are the values initialized to?
  - Aliasing (ie, having two varaibles refer to the same array)
  - What contracts do we need to assert the safety of an array access?
- Linear and Binary Search
  - How do they work? Why would we use one over the other?
  - What loop invariants and pre/post conditions do we need for each?
  - How do we calculate the mid index in binary search to avoid overflow?
- Big-O
  - Be able to determine a tight bound for a given mathematical function or code
  - Know the formal definition of Big-O and how to use it to prove bounds on functions (finding the c, n_0 values)
  - See **Problem 2, s12 final solutions**

- Sorting
  - Insertion sort : What is its runtime, how does it work?
  - Merge sort : What is its runtime, how does it work?
  - Quick sort : What is its runtime (worst-case and average case), how does it work?
  - See **Problem 3, s12 final solutions**
  - See **Problem 1, f12 final solutions**
- Pointers
  - Syntax for allocating space and getting a pointer, using & to get a pointer, and derefernce the pointer (both with * and ->)
  - When is it illegal to dereference a pointer?
  - See **Problem 4, f13 midterm 1 solutions**
- Stacks and Queues/Linked Lists
  - How do linked lists work? What are advantages/disadvantages compared to arrays?
  - What are the interfaces for stacks/queues? Remember to respect the interface!
  - In what order do things get popped/dequeued?
  - How can we implement Stacks/Queues using Linked Lists?
  - What data structure contracts do we have for each of these?
  - See **Problem 1, f13 midterm 2 solutions**

Before Midterm 2

- Amortized Analysis
  - How can we apply amortized analysis to show better average bounds?
  - What do tokens represents? How does the token method work?
  - How do UBAs work and how can we use amortized analysis to show constant-time insertions.
  - See **Problem 2, f12 final solutions**
- Hash Tables
  - What goes in the client interface and user interface? What should the asymptotic runtimes be?
  - What makes a good hash function?
  - What invariants do hash tables have, and how do they work?
  - See **Problem 4, s13 midterm 2 solutions**
- Priority Queues/Heaps
  - What are the lookup/insertion times?
  - Know the difference between a low priority number and a low priority

  - What are the invariants and partial invariants for heaps?
  - What does a heap insertion or delmin look like?
  - See **Problem 2, f12 midterm 2 solutions**

- Memory management in C
  - When do you need to free memory? (Hint: **ONLY IF IT IS X{M,C}ALLOC'ed**)
  - Common pitfalls with stack-allocated variables (why would `return &x;` be bad)
  - Why do we often need special freeing functions, such as `stack_free()`

- BSTs and AVL trees
  - Understand the invariants for BSTs and AVL trees (what is different between their invariants, and what is the advantage of an AVL tree?)
  - Be able to determine which rotations are needed to satisfy AVL tree invariants.
  - Given an ordered set of inputs, give the resulting BST and AVL tree.
  - See **Problem 4, s12 final solutions**
  - See **Problem 3, f12 final solutions**

- Types and casting in C
  - What happens if you case a signed type to an unsigned type of the same size (and vice versa)
  - What happens if you cast a larger type to a smaller type?
  - What happens if you cast a small signed type to a large signed type? How does this differ if both are unsigned?
  - Why is it considered a bad idea to cast from a small signed type to a large unsigned type, or from small unsigned to large signed?
  - What does casting between different pointer types do?

- Generic Data Structures
  - What is the benefit of generic data structures?
  - Why do we generally use void*s for these? Why does the client need to supply functions?
  - Why does the client pass the functions in the constructor (the ***_new()) function?
  - What is the syntax for typedef'ing, declaring, and calling function pointers?
  - Understand how the generic hashtable works.

Before Final Exam

- Tries
  - Be able to look at a picture of a Ternary Search Trie and determine what words are in it.
  - When are tries better than Hash Tables?
  - What are the invariants on tries, and how do insertions and lookups work? (especially for TSTs)

  - See **Problem 5, f12 final solutions**

- Graph Search
  - What are the different ways to represent a graph? What are the pros/cons of each?
  - What does a graph search algorithm do?
  - What is the difference between a BFS and DFS? Which can be implemented recursively? Which can be implemented with a stack? With a queue? Know how to implement it in all three ways

- Kruskal's Algorithm/Union Find
  - What is a minimum spanning tree? Why are they useful?
  - Understand how Kruskal's algorithm works, and the major steps in it.
  - Which part of the algorithm is a union-find used in?
  - How does a union find keep track of connected components? Be able to track a union find as we add edges to it.
  - See **Problem 5, s12 final solutions**
  - See **Problem 6, f12 final solutions**

- Virtual Machines
  - Understand the basics of how a VM like the c0vm works. You should be able to unsderstand what simple bytecode is doing (NOTE: do not waste time memorizing the individual opcodes)