Final Exam Review

1 Time and Location

The final exam will be given in EGR 0135 (our normal meeting place) 8:00 a.m.-10:00 a.m., on Saturday, May 18th.

2 Format

The final will be open-book (Kernighan and Ritchie, and Loukides and Oram), open-notes, closed everything else. In other words, you can bring any material that you have accumulated in class and recitation, but nothing else. Also, no laptops or calculators are allowed during the exam.

3 Scope

The final will be cumulative, covering all the material introduced this semester. Here’s an outline of the topics covered since the 2nd midterm. You should append it to the two other outlines you received before midterms 1 and 2 for a complete list of the topics we covered this semester. Disclaimer: this is not meant to be an absolutely water-tight complete list of topics. If there is a topic not present in this list, it may still show up on the final. However, the list is a pretty good first-cut at what we have covered.

I. Abstract Data Types
   A. Overview
      1. Definition: user-defined datatype, often complex, with strong separation between use and implementation
      2. Examples
         a. Complex numbers
         b. Rational numbers
         c. Linked list
         d. Dictionary
         e. Checkers game
      3. Features
         a. Well-defined interface
         b. Implemented through programming convention
   B. Implementation (basic issues)
1. Data portion: typedef, typically a struct
2. Operators portion: provide single function per operation permitted by ADT
   a. Public vs private functions
3. Common public functions
   a. Constructor
   b. Destructor
   c. Print
4. Encapsulation
   a. Interface: <ADT name>.h
   b. Implementation: <ADT name>.c
   c. Information hiding: use of "static"
C. Polymorphism (advanced issues)
   1. Make ADTs as general as possible to support many types of data
   2. Unions: polymorphic variables
      a. Requires declaring a template, much like structs
      b. Instead of different fields, members specify different type for the same variable
      c. Use: use "." syntax to select the type
3. Generic pointers
   a. Use "void *" to permit ADT to point to user-defined data
   b. Doesn't require fixing types of data supported by the ADT (more general than unions)
D. Code examples
   1. rational.h / rational.c

II. Function Pointers
A. Basics
   1. Declaration
      a. Requires declaring type of function pointed to
      b. Ex: void (*func)(int)
   2. Call
      a. Assignment function pointer to point to function
      b. Call function through function pointer
   3. Function pointers are not as general as data pointers; cannot perform pointer arithmetic
B. Uses
   1. Dynamic function selection
      a. Choose between different algorithms
      b. Choose between debug / no-debug code
   2. Supporting polymorphism
      a. Data that "knows" how to process itself
      b. User-defined functions that handle user-defined data
C. Code examples
   1. func-pointer.c
   2. poly-print.c

III. Bit-Level Operations
A. Integer representation
   1. 2's complement
B. Operators
1. Not: ~
2. And: &
3. Or: |
4. Xor: ^
5. Shift: >> and <<
   a. Equivalence with multiplication and division
C. Masking
1. Single bit
2. Bit fields
D. Uses
1. Creating datatypes that can’t be expressed using primitive datatypes
2. Extracting data from external sources (ex: IP packets)
E. Code examples
1. bit-negate.c
2. nibble-array.c
3. ip-header.c

IV. Performance
A. Optimization
1. Algorithmic
2. Coding
3. Compiler
   a. -O0 (default), -O1, -O2, -O3