Princeton University
COS 217: Introduction to Programming Systems
Spring 2007 Final Exam Preparation

Professor Appel's Study Guide for the Final Exam

1. Understand any C program in all 12 weeks of the lecture notes (exception: don't bother with slides 126 and 154).
   a. Convert-to-uppercase (slides 34-45)
   b. Capitalize first letter (49-63)
   c. One-line emacs (91-120, 261-280), complete program at
      http://www.cs.princeton.edu/courses/archive/spr07/cos217/one-line-emacs/
   d. Stack (157-180)
   and smaller program fragments on other slides
2. Review all the programming assignments you did.
3. Understand the alpha-beta algorithm.
4. Understand the mathematical derivation of the Naive Bayes algorithm.
5. Be prepared to read or write an assembly-language program involving local variables, global variables, characters, integers, arrays, structs, pointers, or functions.
6. Understand in general what an operating system does for you.
7. Understand modularity.

Bonus: what's not on the exam!

1. IA-32 instruction encodings (opcode, modR/M, etc.).
2. Segment registers and other "registers you don't care about."
3. Incremental evaluation of heuristic functions.
4. Regular expressions.

Topics

*You are responsible for all material covered in lectures, precepts, assignments, and required readings. This is a non-exhaustive list of topics that were covered. Topics that were covered after the midterm exam are in **boldface**.*

1. C programming
   - The program preparation process
   - Memory layout: text, stack, heap, rodata, data, bss sections
   - Data types
   - Variable declarations and definitions
   - Variable scope, linkage, and duration/extent
   - Variables vs. values
   - Operators
   - Statements
   - Function declarations and definitions

Pointers
Call-by-value and call-by-reference
Arrays
Strings
Command-line arguments
Constants: #define, enumerations, “constant variables”
Input/output functions
Text files
Structures
Dynamic memory management: malloc() and free()
Dynamic memory management errors: dangling ptr., memory leak, multiple free
Void pointers
Function pointers and function callbacks
Macros and their dangers (see King Section 14.3)
The assert() macro
The fwrite() and fread() functions

2. Programming style
Modularity, interfaces, implementations
Programming by contract
Multi-file programs using header files
Protecting header files against accidental multiple inclusion
Opaque pointers
Stateless modules
Abstract data types
Memory "ownership"
Preserving invariants
Testing
Profiling and instrumentation
Performance tuning

3. Number representations
The binary, octal, and hexadecimal number systems
Signed vs. unsigned integers
Binary arithmetic
Signed-magnitude, one's complement, and two's complement representation of negative integers

4. IA-32 architecture and assembly language
General computer architecture
The Von Neumann architecture
Control unit vs. ALU
The memory hierarchy: registers vs. cache vs. memory vs. disk
Little-endian vs. big-endian byte order
CISC vs. RISC
Language levels: high-level vs. assembly vs. machine
Assembly language
Directives (.section, .asciz, .long, etc.)

Mnemonics (movl, addl, call, etc.)
Instruction operands: immediate, register, memory
Memory addressing modes
The stack and local variables
The stack and function calls
The C function call convention

~~Machine language~~
~~Opcodes~~
~~The ModR/M byte~~
~~Immediate, register, memory, displacement operands~~

Assemblers
The forward reference problem
Pass 1: Create symbol table
Pass 2: Use symbol table to generate data section, rodata section, bss section, text section, relocation records

Linkers
Resolution: Fetch library code
Relocation: Use relocation records and symbol table to patch code

5. Operating systems
Services provided
Processes
The process life-cycle
Context switches
Virtual memory
Computer security
Buffer overrun attacks

6. Applications
De-commenting
Lexical analysis via finite state automata
String manipulation
Symbol tables, linked lists, hash tables
Dynamically expanding arrays
Game playing
Minimax search
Alpha-beta search
~~Incremental game state evaluation~~
Spam filters
Naive Bayesian learning
~~Regular expressions~~

7. Tools: The UNIX/GNU programming environment
UNIX, bash, xemacs, gcc, gdb, **gdb for assembly language, make, gprof**

Readings

*As specified by the course "Schedule" Web page. Readings from the second half of the course are in **boldface**.*

Required:

C Programming (King): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19

The Practice of Programming (Kernighan & Pike): 1, 2, 4, **5, 6, 7**

Computer Systems **(Bryant & O'Hallaron): 2, 3**
or
Programming from the Ground Up **(Bartlett) 1, 2, 3, 4, 9, 10, B, E, F**

Othello (http://www.pressmangames.com/instructions/instruct_othello.html)

Kuperman et al. "Detection and Prevention of Stack Buffer Overflow Attacks." ***Communications of the ACM,*** **Volume 48, Number 11. November 2005**

Machine Learning **(Mitchell) 6.9, 6.10**

Goodman et al. "Stopping Spam." ***Scientific American*****. April 2005**

Recommended:

Computer Systems (Bryant & O'Hallaron): 1, **5, 7**

Programming with GNU Software (Loukides & Oram): 1, 2, 3, 4, 6, **7, 9**

Artificial Intelligence **(Rich) 12**

Programming from the Ground Up **(Bartlett) 5, 6, 7, 8, 11, 12, 13, C**