Princeton University
COS 217: Introduction to Programming Systems
Spring 2010 Final Exam Preparation

The exam will be a three-hour, open-book, open-notes exam. Electronic devices are not allowed.

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. Number Systems

- The binary, octal, and hexadecimal number systems
- Finite representation of integers
- Representation of negative integers
- Binary arithmetic
- Bitwise operators

2. C Programming

- The program preparation process: preprocess, compile, assemble, link
- Program structure: multi-file programs using header files
- Process memory layout: text, stack, heap, rodata, data, bss sections
- Data types
- Variable declarations and definitions
- Variable scope, linkage, and duration/extent
- Constants: #define, constant variables, enumerations
- Operators and statements
- Function declarations and definitions
- Pointers; call-by-reference
- Arrays: arrays and pointers, arrays as parameters, strings
- Command-line arguments
- Text files
- Structures
- Dynamic memory mgmt.: malloc(), calloc(), realloc(), free()
- Dynamic memory mgmt. errors: dangling pointer, memory leak, double free
- Abstract data types; opaque pointers
- Void pointers
- Function pointers and function callbacks
- Parameterized macros and their dangers (see King Section 14.3)
-

3. Programming-in-the-Large

- Debugging heuristics
    - Understand error messages, think before writing, look for familiar bugs, divide and conquer, add more internal tests, display output, use a debugger, focus on recent changes

  - **Heuristics for debugging dynamic memory management: look for familiar bugs, make the seg fault happen in a debugger, manually inspect each call of malloc(), etc., temporarily hard-code malloc(), etc. to request a large number of bytes, temporarily comment-out each call of free(), use Meminfo**
- Program and programming style
  - Top-down design
- Data structures and algorithms
  - Linked lists, hash tables, memory ownership
- Module qualities:
  - Separates interface and implementation, encapsulates data, manages resources consistently, is consistent, has a minimal interface, reports errors to clients, establishes contracts, has strong cohesion, has weak coupling
- Generics
  - Generic data structures via void pointers
  - Generic algorithms via function pointers
- ~~**Testing (snowed out)**~~
  - ~~**External testing taxonomy: boundary condition, statement, path, stress**~~
  - ~~**Internal testing techniques: testing invariants, verifying conservation properties, checking function return values, changing code temporarily, leaving testing code intact**~~
  - ~~**General testing strategies: testing incrementally, comparing implementations, automation, bug-driven testing, fault injection**~~
- **Building**
  - **Automated builds, dependencies, partial builds**
- **Performance improvement (in precept)**
  - **When to improve performance**
  - **Techniques for improving execution (time) efficiency**
  - **Techniques for improving memory (space) efficiency**
- **Performance improvement revisited (in lecture)**
  - **Optimize only when and where necessary**
  - **Improve asymptotic behavior**
    - **Use better data structures or algorithms**
  - **Improve execution time/space constants**
    - **Coax the compiler to perform optimizations**
    - **Exploit capabilities of the hardware**
    - **Capitalize on knowledge of program execution**
- **Portability**
  - **General heuristics**
  - **Heuristics related to hardware differences**
  - **Heuristics related to operating system differences**
  - **Heuristics related to compiler differences**
  - **Heuristics related to library differences**
  - **Heuristics related to cultural differences**

4. **Under the Hood: Toward the Hardware**

- **Computer architectures and the IA-32 computer architecture**
  - **Computer organization**
  - **Control unit vs. ALU vs. memory**
  - **Little-endian vs. big-endian byte order**
  - **Language levels: high-level vs. assembly vs. machine**
- **Assembly languages and the IA-32 assembly language**
  - **Directives (.section, .asciz, .long, etc.)**
  - **Mnemonics (movl, addl, call, etc.)**

    - **Control transfer: condition codes and jump instructions**
    - **Instruction operands: immediate, register, memory**
    - **Memory operands: direct, indirect, base+displacement, indexed, scaled-indexed**
    - **The stack and local variables**
    - **The stack and function calls: the IA-32 function calling convention**
- **Machine language**
    - **Opcodes**
    - **The ModR/M byte**
    - **The SIB 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. Under the Hood: Toward the Operating System

- **Exceptions and Processes**
    - **Exceptions: interrupts, traps, faults, and aborts**
    - **Traps in Intel processors**
    - **System-level functions (alias "system calls")**
    - **The process abstraction**
    - **The illusion of private control flow**
        - **Reality: context switches**
    - **The illusion of private address space**
        - **Reality: virtual memory**
- **Memory Management**
    - **The memory hierarchy: registers vs. cache vs. memory vs. local secondary storage vs. remote secondary storage**
    - **Locality of reference and caching**
    - **Virtual memory**
    - **Implementation of virtual memory**
        - **Page tables, page faults**
- **Dynamic memory management**
    - **Memory allocation strategies**
    - **Free block management**
    - **Optimizing malloc() and free()**
- **I/O Management**
    - **The stream abstraction**
    - **Implementation of standard C I/O functions using Unix system-level functions**
    - **The open(), creat(), close(), read(), and write() functions**
- **Process management**
    - **Creating and destroying processes**
        - **The getpid(), execvp(), fork(), and wait() functions**
        - **The exit() and system() functions**
    - **Redirection of stdin, stdout, and stderr**
        - **The dup() function**
- **Signals and alarms**
    - **Sending signals via keystrokes, the kill command, and the raise() and kill() functions**

- **Handling signals: the signal() function**
- **The SIG_IGN and SIG_DFL parameters to signal()**
- **Alarms: the alarm() function**
- **Child processes and signals**
- ~~**Interval timers**~~
- **Race conditions and critical sections**
- ~~**Blocking signals: the sigprocmask() function**~~

6. Applications

- De-commenting
- Lexical analysis via finite state automata
- String manipulation
- Symbol tables, linked lists, hash tables
- Dynamically expanding arrays
- **High-precision arithmetic**
- **Buffer overrun attacks**
- **Heap management**
- **Unix shells**

7. Tools: The Unix/GNU programming environment

- ~~Unix, Bash, Emacs, GCC, GDB for C,~~ ~~**Gprof**~~**, Make,** ~~**GDB for assembly language**~~

Readings

As specified by the course "Schedule" Web page. Readings that were assigned after the midterm exam 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, 22
- *Computer Systems* (Bryant & O'Hallaron): 1, **3 (OK to skip 3.14 and 3.15), 8.1-8.5, 10**
- ***Communications of the ACM*** **"Detection and Prevention of Stack Buffer Overflow Attacks"**
- ***The C Programming Language*** **(Kernighan & Ritchie) 8.7**

Recommended:

- *Computer Systems* (Bryant & O'Hallaron): 2, **5.1-5.6, 7, 11**
- *The Practice of Programming* (Kernighan & Pike): 1, 2, 4, 5, 6, **7, 8**
- *Programming with GNU Software* (Loukides & Oram): 1, 2, 3, 4, 6, **7**, 8, **9**

Note: Do not print the IA-32 manuals!!! You will not need them during the exam.