Princeton University
COS 217: Introduction to Programming Systems
Fall 2014 Final Exam Preparation

The exam is a three-hour, closed-book, closed-notes, closed-handouts exam. The exam is cumulative, but emphasizes second-half material. No laptops, calculators, or other electronic devices are permitted.

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 unsigned integers
        - Operations on unsigned integers
    - Finite representation of signed integers
        - Signed magnitude, ones' complement, two's complement
        - Operations on signed integers
    - Finite representation of rational numbers

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 and arrays
        - Call-by-reference, arrays as parameters, strings
        - Command-line arguments
    - Input/output functions for standard streams **and files**, and for text **and binary data**
    - Structures
    - Dynamic memory management
        - malloc(), calloc(), realloc(), free()
        - Common errors: dereference of dangling pointer, memory leak, double free
    - Abstract objects
    - Abstract data types; opaque pointers
    - Generic data structures and functions
        - Void pointers
        - Function pointers and function callbacks
    - Parameterized macros and their dangers (see King Section 14.3)

3. Programming-in-the-Large

- Testing
  - External testing taxonomy: statement, path, boundary, stress
  - Internal testing techniques: validate parameters, check invariants, check function return values, change code temporarily, leave testing code intact
  - General testing strategies: automate the tests, test incrementally, let debugging drive testing (fault injection)
- Building
  - Separate independent paths before link
  - Motivation for make, make fundamentals, macros, abbreviations, pattern rules
- Program and programming style
  - Bottom-up design, top-down design, least-risk design
- Debugging
  - General heuristics for debugging: 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 common DMM bugs, diagnose seg faults using gdb, manually inspect malloc(), calls, comment-out free() calls, use Meminfo, use Valgrind
- Data structures and algorithms
  - Linked lists
  - Hash tables: hashing algorithms, defensive copies, key ownership
- Modularity
  - History of modularity: non-modular, structured, abstract object, abstract data type programming
  - Module qualities: encapsulates data, is consistent, has a minimal interface, detects and handles/reports errors, establishes contracts, has strong cohesion, has weak coupling
- Performance Improvement
  - When to improve performance
  - Improving execution (time) efficiency: do timing studies, identify hot spots, use a better algorithm, enable compiler speed optimization, tune the code
  - Improving memory (space) efficiency: use a smaller data type, compute instead of storing, enable compiler space optimization

4. Under the Hood: Language Levels Tour

- **Language levels**
  - **High-level vs. assembly vs. machine language**
- **Computer architecture**
  - **The Von Neumann architecture**
    - **RAM, registers, ALU, control unit, CPU**
  - **Big-endian vs. little-endian byte order**
  - **CISC vs. RISC architectures**
- **The IA-32 computer architecture**
  - **EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EFLAGS, EIP registers**
- **IA-32 assembly language**
  - **Instructions: directives and mnemonics**
  - **Defining data**
  - **Performing arithmetic**
  - **Instruction operands**
    - **Immediate vs. register vs. memory**
  - **Control flow**
    - **Jumps**

- **CC bits in EFLAGS register**
- **Conditional jumps with "signed" data**
- **Conditional jumps with "unsigned" data**

- **Data structures**
  - **Arrays**
  - **Direct, indirect, base+displacement, indexed, scaled-indexed memory addressing**
  - **Structures**
  - **Padding**
- **Function calls and the IA-32 function call conventions**
  - **Passing and accessing arguments**
  - **Storing and accessing local variables**
  - **Returning a value**
  - **Handling registers**
    - **Caller-save and callee-save registers**

- **IA-32 machine language**
  - **Prefix, opcode, ModR/M, SIB, displacement, immediate fields**
- **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: Service Levels Tour

- **Exceptions and Processes**
  - **Exceptions**
    - **Synchronous vs. asynchronous**
    - **interrupts, traps, faults, and aborts**
  - **Traps and system-level functions in IA-32**
  - **The process abstraction**
  - **The illusion of private address space**
    - **Reality: virtual memory via page faults**
  - **The illusion of private control flow**
    - **Reality: context switches during exception handling**
- **Storage Management**
  - **Locality of reference and caching**
  - **Typical storage hierarchy: registers vs. cache vs. memory vs. local secondary storage vs. remote secondary storage**
  - **Virtual memory**
    - **Implementation of virtual memory**
      - **Virtual addresses vs. physical addresses**
      - **Page tables, page faults**
    - **Benefits of virtual memory**
- **Dynamic memory management (DMM)**
  - **The need for DMM**
  - **DMM using the heap section**
    - **The brk() and sbrk() system-level functions**
    - **Internal and external fragmentation**
    - **Free-list, doubly-linked free list, bin implementations**
  - **DMM using virtual memory**

        - **The mmap() and munmap() system-level functions**
- **Process management**
    - **Creating processes**
        - **The getpid() and fork() system-level function**
    - **Waiting for (reaping, harvesting) processes**
        - **The wait() system-level function**
    - **Executing new programs**
        - **The execvp() system-level functions**
    - **The system() function**
- **I/O Management**
    - **The file abstraction**
    - **Standard C I/O**
        - **Buffering**
    - **Unix I/O**
        - **File descriptors, file descriptor tables, file tables**
        - **The creat(), open(), close(), read(), write() system-level functions**
    - **Implementing standard C I/O using Unix I/O**
    - **Redirecting standard files**
        - **The dup() system-level 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 arguments to signal()**
    - **Alarms**
        - **The alarm() function**
    - **Race conditions and critical sections**
    - **Blocking signals**
        - **The sigprocmask() function**

7. Applications

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

8. Tools: The Unix/GNU programming environment

- Unix/Linux, Bash, Emacs, GCC, GDB for C, Make, Gprof, **GDB for assembly language, objdump**

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, 20.1, **22**
- *Computer Systems* (Bryant & O'Hallaron): 1, **3 (OK to skip 3.13 and 3.14), 8.1-5, 9**
- ***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, 6, 7, 10**
- *The Practice of Programming* (Kernighan & Pike): 1, 2, 4, 5, 6, 7, 8
- *Unix Tutorial for Beginners* (website)
- *GNU Emacs Tutorial* (website)
- *GNU GDB Tutorial* (website)
- *GNU Make Tutorial* (website)
- *GNU Gprof Tutorial* (website)
- ***Security as a Class of Interface Guarantee*** **(website)**