Wednesday, March 5, 2014

CMSC 104 - Exam 1 Topics

Machine Architecture and Number Systems

- Major components
  - Central Processing Unit (CPU) - the "brain" of the computer, controls all other functions.
  - Main Memory (RAM) - working memory. Contains running programs and information currently in use. Main Memory is volatile in that data disappears when the power is turned-off. Organized into **bits** and **bytes**. Each byte has an **address**. Other computer components can read or write data at a specified address.
  - Secondary Storage - long-term memory such as hard drives, SSDs, thumb drives, CDs, and DVDs. Contains computer programs, data that has been saved from an executing program. Secondary Storage is **non-volatile**.
  - Input/Output (I/O) Devices - all devices that let the user communicated with the computer or the computer communicate with another computer. Monitor, keyboard, mouse, disk drive, CD or DVD drive, printer, scanner, camera, speakers, etc.
  - Bus - the data "highway" that connects the CPU, memory, storage, and I/O.
- The program execution process - e.g. the steps when starting MS Word
  - Mouse click on the Word icon tells the Operating System to run MS Word
  - MS Word is loaded from Secondary Storage to Main Memory
  - The CPU reads the program instructions from Main Memory and executes them one at a time
  - MS Word appears on your monitor
- Bits, Bytes, Words
  - A **bit** is a single 0 or 1
  - A **byte** consists of 8 bits
  - A **word** is 32 bits or 4 bytes. On more modern computers, a word may be 64 bits.
  - Sometimes you will here references to a **nibble**. This is just half a byte, or 4 bits.
- Number Systems - decimal, binary, and hex

- Decimal numbers are base 10. We think of the columns (right to left) as the powers of 10 starting with 1 = 10^0.
- Binary numbers are base 2. We think of the columns (right to left) as the powers of 2 starting with 1 = 2^0.
- Hexadecimal is a short hand for writing binary.
  - Each hexadecimal character corresponds to a nibble (0 = 0000, 1 = 0001, 2 = 0010, 3 = 0011, 4 = 0100, 5 = 0101, 6 = 0110, 7 = 0111, 8 = 1000, 9 = 1001, A = 1010, B = 1011, C = 1100, D = 1101, E = 1110, F = 1111).
  - A byte is written as two hexadecimal characters, e.g. 3F = 0011 1111.
- **You must know how to convert among the three number systems.**

Operating Systems and Linux

- The Operating System coordinates the CPU, memory, and I/O devices; it allows the user to communicate with the computer, controls access, and keeps track of running processes. Often just called an **OS**.
- Examples of an OS: Linux, Windows, Android, Mac OS, iOS.
- Linux provides both a Graphical User Interface (GUI) and a command-line interface (CLI).
- The Linux CLI prompt is something like "`linux1[1]%`". It can vary depending on the configuration of the computer.
- Linux CLI Overview
  - Files. A file is a sequence of bytes. Can be created in many different ways, for example using emacs. File names should not include spaces; stick with upper and lower case letters, numbers, underscore (_), and hyphen (-).
  - Directories. Directories are just special files that contain other files. They are organized in a hierarchy, like an upside-down tree. **You need to understand how to navigate the directory hierarchy**.
  - Special Directories. **Home directory** is where the OS looks for files when you first login. The **current directory** is where the OS is currently looking for files; the current directory is referenced by "." (dot). The **parent directory** is the directory

one level up from the current directory; it is referenced by ".." (dot-dot). **Root** (/) is the top of the directory hierarchy.

- Path names. Path names tell the OS where to find a file. **Absolute path names** indicate the location of a file starting at the root; they always start with /. **Relative path names** indicate the location of a file relative to the current directory.
- Directory commands. `cd` is used to change directory; `pwd` prints the current directory; `mkdir` creates a directory; `rmdir` removes a directory; `ls` lists the contents of a directory. **Know how to use these**.
- File commands. `rm` removes a file; `cp` copies a file; `mv` moves a file; `cat` displays the content of a file; `more` pages through the content of a file. **Know how to use these**.
- Miscellaneous: `man` is for manual (help), `emacs` is a text editor.
- Wildcards: in a file name, "?" matches any single character and "*" matches any number of characters. E.g. `hw?.txt` matches `hw1.txt`, `hw2.txt`, but not `hw10.txt`; `hw*.txt` matches `hw314159blargh.txt`.
- I/O Redirection: You can use ">" to save the output of a command to a file. E.g. `ls > files.txt` creates a file called `files.txt` containing a listing of the current direcory.
- **General Pointer:** Review the material from Lab 1. These are things you should understand.

Algorithms

- An **algorithm** is a finite set of unambiguous executable instructions that directs a terminating activity.
- Examples of algorithms: washing machine, Euclid's algorithm.
- Syntax vs. semantics
- Pseudocode. Be able to write pseudocode for a simple algorithm using syntax as described in the lectures.
- Sequence, selection, and repetition. What are they and give examples.

C Programming

- General C Stuff
  - The relationships among machine code, assembly language, and high-level languages such as C.
  - Structure of a program: program header comment, preprocessor directives, main()
  - The **header comment** provides general documentation for the program - file name, date, author, description. Comments begin with /* and end with */.
  - **Preprocessor directives** begin with a # in column one. They help to connect your program with external programs (libraries). E.g. the `printf()` function is part of the stdio library, so code that uses `printf()` should have `#include <stdio.h>` in the preprocessor directives.
  - Every program has a **main function** called `main()` in the program. The main function is typically the first thing after the preprocessor directives. The function body is delimited by curly brackets ({ and }). The main function returns an integer, so it is declared in the function as `int main()` and the last statement in the function should be `return 0;`
  - Study the Hello, World! example. Be sure you understand the the structure of the program.
- Compiling a C program
  - The three phases:
    - Preprocessing - helps to interface your code to external code; also used to make code more portable.
    - Compilation - transforms your code into **object code**, which is machine code, but without all the parts needed for a runnable program.
    - Linking - the output of compilation is linked to external programs and the code needed to make it runnable. Produces a file called a.out (unless you specify a different name)
  - We are using the gcc compiler. To compile the program prog.c, creating an executable called prog, use the command
    - `gcc -Wall prog.c -o prog`
  - The "-Wall" option generates warning messages, which are helpful for debugging.

  - The resulting program (`prog`) can be run using `./prog`
- Variables
  - Know the naming rules: letters, digits, underscore; can't begin with a number; may not be a reserved word. Be able to identify illegal variable names.
  - Variable names are case sensitive.
  - Variable types: `int`, `float`, `double`, `char`
  - How variables are declared, e.g. `float pi;`
  - How variables are initialized (as part of declaration or later).
  - Using variables in simple formulas.
- Arithmetic Operators
  - +, -, *, /, % - especially keep in mind how these work for int vs. float.
  - Types and promotion - how are mixed-type expressions evaluated (int < float, short < long).
  - Operator Precedence
    - ()
    - * / % (left to right)
    - + - (left to right)
    - =
  - Using parentheses to affect order of computation or for clarity.
- Relational and Logical Operators
  - In C, false is 0, true is anything other than zero.
  - Relational operators: < , >, <=, >=, ==, !=
  - Arithmetic expressions are false or true as they are 0 or non-zero, respectively.
  - Be able to evaluate relational and arithmetic expressions.
  - The if, if/else, and nested if/else statements.
  - Watch out for = vs. ==

- Logical operators: && (AND), II (OR), ! (NOT)
- Know the uses and truth tables for logical operators.
- Order of precedence
  - ()
  - * / % (left to right)
  - + - (left to right)
  - < <=   > >= (left to right)
  - == != (left to right)
  - &&
  - II
  - =
- Be able to evaluate logical expressions.