CS2261 Media Device Architecture – Recitation 3

Overview:

* Recitation Slides posted online
* Due to several days of missed lecture, we reviewed basic concepts of the C programming language
* Basic review of C
  + End file names with .c
  + C has no classes or methods. C has functions.
  + To run a C file in command prompt, type in gcc filename.c (which compiles the file)
    - After compilation, another file is created called a.i
    - To run this file, simply type in “a” and hit enter on the next command prompt line
  + No methods in C, we use functions instead. Difference not important
* Four basic steps
  + Preprocessing
    - Machine independent (no matter the machine you run on, it’ll be the same
    - Creates .i files
    - Some basic code below:
    - No importing in C, instead do “including” which copies over a .h file
      * Ex. #include <filename.h>
    - Printf(“statement”) -to print things onto console
    - #define THIS\_IS\_A\_MACRO “Cornerface”
      * This is basically make a variable called THIS\_IS\_A\_MACRO
  + Compilation
    - Machine dependent
      * We use flags to change defaults
    - Translates into assembly source files (.s)
      * Many different kinds of assembly
      * The GBA uses ARM processor
    - Still stored as ASCII
  + Assembly
    - Machine independent
    - Turns the assemble source into binary machine code (.o files)
    - No longer ASCII representation
    - Not too complicated if you know assembly language
  + Linking
    - Machine dependent
      * Use flags and makefile directive to do it for GBA
    - Organizes the object code and fills in the gaps
* ^ These four steps aren’t all that important
* **Pointers** (actually important)
  + Gives you assembly-like power over variables
  + Pointers don’t exist in java, because they allow you to physically alter the variable, incredibly dangerous if you don’t know what you’re doing
  + C runs on the machine you’re running on, but java will run on JVM (Java Virtual Machine)
    - Address: storage place in memory of code
  + Since C is a strongly typed language, you need to know the type of every variable at all times
  + Declaring a pointer: typename \*varname;
    - Ex. int \*a;
  + You can assign an address directly, but don’t, work backwards instead
* Referencing and dereferencing
  + If you want to know the address of something, you reference it
  + Use the reference operator &
    - Ex. int a = 7;
    - int \*b = &a;
* To find what’s stored at an address, you dereference using the dereference operator \*
  + Use \*
    - Int c = \*b;
* The \* functions as a multiplication operator, pointer declaration operator, and a dereference operator
* The & functions as a reference operator and a bitwise “AND” operator

Examples (this is supposed to be code):



Other general notes:

* A pointer is an address
* Should never directly assign addresses, thus the reference variable
  + But to directly assign, it would look like this:
    - int \*b = (int\*) 0xfff4;
    - 0xfff4 is meant to represent a hexadecimal
* When you create a variable, the info is stored randomly in the address so we use the & to get a hold of the addresses
* Addresses are stored in hexadecimal
* We’ll be using hexadecimal and binary a lot
* **Need to know how to convert between hexadecimal, binary and decimal**
  + <http://www.learning-about-computers.com/tutorials/binary_and_hexidecimal.shtml>
* It is best to create diagrams mimicking storage to keep track of and visualize what’s happening in your code, like:

|  |  |  |
| --- | --- | --- |
|  | Value (randomly assigned, this column reps storage) | Address (hexadecimal) |
| int a | 15 | 0x400 |
|  |  | 0x404 |
| int b | 9 | 0x408 |
|  |  | 0x40C |
| int \*c | 0x404 (address of b) | 0x410 |
|  |  | 0x414 |
|  |  | 0x418 |
|  |  | 0x41C |