Quiz 0 - Study Tips

CS50 — Fall 2011

Prepared by: Doug Lloyd '09

October 10, 2011

Helpful Hints

Here are a few study tips for preparing for the first quiz.

- Read the quiz instructions, posted on the course website:
  (*https://www.cs50.net/quizzes/2011/fall/0/about0.pdf*)
  Take particular note of the line: "The quiz will be more conceptual than it will be mechanical." The practice problems that you completed over the past few days are useful only inasmuch as they force you to apply the concepts you've learned so far. It's not enough to be able to do these problems. You need to also know why these solutions work!

- Practice tracing through code. Be prepared to do the kind of problems/programs that have appeared in section and lecture.

- Be prepared to identify bugs in a segment of nearly working code. Review common pitfalls and make a list of the errors which caused you the most grief when debugging your last two assignments. Be comfortable with `true`/`false`, one/zero, `if-else`, `switch`, comparison operators (`<`, `>`, `==`) and logical operators (`&&`, `||`). Remember: 0 is `false`, 1 is `true`. Remember also that `=` denotes an assignment, while `==` performs a comparison.

- Read over the part of the lecture notes which discusses compiling, libraries, etc. You do not have to know this in great detail, but you should understand what is going on when you ask the computer to compile your program. Review all the lecture notes, section slides, and cheat sheets. Find and make note of tricky syntax.

- While everything we've covered is fair game for the quiz, the focus will be on conceptual understanding and your ability to code (short programs) in C. You should also review the programmatic constructs supported by Scratch, though you certainly needn't memorize all of Scratch's puzzle pieces.

- The quiz is closed-book, but you're allowed to bring "one two-sided page (8.5" × 11") of notes, typed or written." Use that page to jot down details you're worried you might forget (*e.g.*, the format of a for loop in C). To be clear, it's not terribly important to have such details memorized at this point (after all, you can always look such up in the real world). But why waste time on the quiz trying to remember little things like that. The "cheat sheets" that you have received up to this point should also prove helpful to you when working out "the little things" like this.

- If you need to prioritize your prep, the best guide to the material we've covered thus far is perhaps the scribe notes, available under Lectures on the course's website.

- Try to answer all of the practice quiz questions!

Topics

What follows is a non-exhaustive list of some topics which might show up on the quiz.

Abstract concepts.

- hierarchical decomposition (aka functions!)
- error checking
- basic debugging

Basic Linux Commands. Make sure you know what Linux commands are, and how to use the basic ones such as: `ls`, `cd`, `ssh`, `man`, `mkdir`, `cp`, `mv`, etc.

Compilation. What is the process of compilation; in particular, what does a `Makefile` look like and how does it work?

Binary. Can you convert a number into binary, and read binary numbers?

ASCII. What is the ASCII standard, and why do we have one?

Cryptography. What is cryptography and how does it work in general terms. Also you should be familiar with the *Caesar* and *Vigenere* ciphers and understand how to code them in C.

C Programming Topics

Preprocessor Commands. What is the C preprocessor? You should also be familiar with the directives:

- `#include`
- `#define`

Types. Be familiar with the basic C numeric types, and the CS-50 types:

- `char, int, float, double`
- `bool, string`

What is the difference between an `int` and an `unsigned int`? What different values can they take on?

You should also be familiar with array types: how to declare and initialize them, indexing into them using brackets (`a[i]`), and passing them as arguments to functions.

Pointers. Know what a pointer is (an address), how to obtain a pointer using the addressing operator (`&`), how to make use of a pointer using the dereference operator (`*`), the relationship between pointers and arrays, good pointer style, and what a `void *` pointer is. Know how we use pointers to dynamically allocate memory using `malloc()`. Know from where in memory `malloc()` allocates and know what we must do to all memory we dynamically allocate when we're done with it (`free()`). What is it called if we don't `free()` everything we `malloc()`?

Variables.

- declaration uninitialized `int x;`
- with initialization `int x = 7;` (sometimes called *instantiation*)
- local versus global declaration (and the style implications of each)
- understand variable scoping and be able to say what the scope of a variable is
- know what happens when you reference a variable outside of its scope

Operators.

- What are operators and operator precedence?
- Be familiar with the various operators: arithmetic (`+`, `%`), logical (`||`, `!`), relational (`<`, `==`), assignment (`=`), etc.
- Remember shorthand operators such as `++`, `-=`, etc.

Library Functions. Make sure you know how to use common library functions. In particular: `printf()`:

- `%d %f %c %s`
- field width specification (`%3d`)
- precision specification (`%.3d`)
- justification specification (`%-3d`)

CS-50 functions:

- `GetInt()`
- `GetString()`
- ...

Control Flow Constructs. You should be comfortable with the C control-flow constructs.

- `if`
- `if-else`
- `switch`
- `for`
- `while`
- `do-while`
- `break`
- `continue`
- `?:`

Defining Functions. Make sure you can *declare* and *define* functions with return and argument types. What happens when you call a function: are the arguments copied? How do you use `return`?

Structures. What is a `struct`? How do we define one? How do we access one's fields?