Section Notes

CS50

Week 5

Contents

1 Quiz 0

1.1 Helpful Hints

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

- Read `http://www.cs50.net/quizzes/2009/0/aboutquiz0.pdf`.
- 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`.
- 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 and section notes. 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.
- 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.

1.2 Topics

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

Abstract concepts.

- hierarchical decomposition
- 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`, etc.

Compilation. What is the process of compilation; in particular, how do you compile a C program using `gcc`?

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.

1.2.1 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 an `unsigned int`?

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.

Variables.

- declaration uninitialized `int x;`
- with initialization `int x = 7;`
- local versus global declaration
- understand variable scoping and be able to say what the scope of variable is

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
- precision specification
- justification specification

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`?

2 Practice!

Below are several questions that are of the sort you will find on the quiz. While this is by no means a comprehensive review of all topics, it will help you study for the quiz. *Of course, you should also attend section, and study the course materials.*

Study Questions.

1. Describe the difference between a `while` loop and a `do-while` loop. Give an example of an instance where a `while` loop would be more useful and another where a `do-while` loop would be more useful and explain why.
2. Write a `for` loop that calculates the sum of the numbers 1 through 10 and stores the result in an integer variable `sum`. Then convert it to a `while` loop.
3. Translate the numbers 12, 33, and 47 into binary. What is the binary number 1010110 in decimal?
4. What is hexadecimal (hex)? How does it relate to binary?
5. What ASCII codes are assigned to the characters 'a' and 'A'?
6. Explain what typecasting is, and why it might be useful.
7. Take your favorite Scratch pieces and determine whether their equivalent lines of C code would be classified as a loop, condition, statement, or boolean expression.
8. If there are 129 students in a class, how many steps would it take to count them all using the most efficient method demonstrated in the first lecture of this class?
9. What is a variable?
10. What is two's complement and why is is convenient for working with binary numbers?
11. How would you compile a program saved as example.c that uses the method GetInt() from the CS50 library such that the resulting file would be named `a.out`? What if we wanted to call it `example1`?
12. What does the line

    ```
    #include <stdio.h>
    ```

    do in a program? Name one function that is declared in stdio.h.
13. Say that at the top of our program we have the following two lines of code:

    ```
    #include <time.h>
    #include "clock.h"
    ```

    Why is `time.h` in angle brackets and `clock.h` in quotes? What does this say about the locations of the header files `time.h` and `clock.h`?

14. Find and correct at least 3 distinct errors in the following piece of code:

```
int
main(int argc, int * argv[])
{
    int x, y;
    // ask user for input
    printf("Give me an integer: ");
    x = GetInt();
    printf("Give me another integer: ");
    y = Getint();

    // do the math
    printf("The remainder when %s is divided by %d is %d!\n", x, y, x / y);
}
```

15. How many times will "`hello, world`" print to the screen when the following code is executed?

```
for(i = 0; i < 10; i++) {
   for(j = 3; j < 303; j += 3) {
       printf("hello, world\n");
   }
}
```

16. David has written a program that takes a student's quiz score and assigns a letter grade to it. The code in the `giveGrade()` function looks like this:

```
char giveGrade(int quizScore) {
   int newScore = round(quizScore);
   char letterGrade;

   switch(newScore) {
      case 100: case 90:
         letterGrade = 'A';
      case 80:
         letterGrade = 'B';
      case 70:
         letterGrade = 'C';
      case 60:
         letterGrade = 'D';
         break;
      default:
         letterGrade = 'F';
   }

   return letterGrade;
}
```

Assuming that the function `round()` takes as its input a quiz score between 0 and 100, and outputs that number rounded down to the nearest 10 (e.g. `round(74)` returns 70 and `round(80)` returns 80), explain why everyone in the class is really upset about their grade.

17. Explain what the following for loop does and how it accomplishes it:

```
for (c = 'A'; c <= 'Z'; c++)
{
  printf("%c", c);
}
```

18. Write the few lines of C code that would print out the multiplication table from 1 to 100 in the following format. (This one's a bit trickier than the others, but good practice nonetheless.)

```
1    2    3    4    5   ....  10
2    4    6    8    10  ....  20
....
10   20   30   40   50  ....  100
```

19. Write a function called `swap` that takes two arguments as integers, and swaps their values locally without returning anything. If this function is called from main as:

```
...
swap(a,b);
...
```

where a and b are integers declared in main, what will happen to the values of a and b? What is one solution to this problem?

20. If main calls the function `domath`, and `domath` calls the function `pow`, what function will be on the **top** of the stack right before `pow` returns: `main, domath, or pow`?

21. Give at least two reasons why `#define` statements are useful.

22. How is `#define` different from `#include`? What, basically, does `#include` do?

23. What does `strlen(a)` return if `a =` "`hello, world!`"? How many bytes does it take to store that string?

24. Write the lines of code that loop through a string, `s`, and print each character out on a line by itself. For example, if `s =` "`hello`", then your program should print:

```
h
e
l
l
o
```

25. If a program, `commandline`, is executed as follows from the prompt:

```
> commandline pick apple cheese
```

what will be contained in the memory location designated by `argv[1][1]`?

26. How do you initialize a 1-D array? A 2-D array? An N-D array?

27. Write code that will print out the contents of the following array:

```
int numbers[5][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12},
{13, 14, 15} };
```

    (a) In the order: `1 2 3 4 5 6 7 8 9 10 11 12 13 14 15`

    (b) In the order: `1 4 7 10 13 2 5 8 11 14 3 6 9 12 15`

28. Write a function that will test to see if an array of integers is in sequential (ascending) order.

29. Write a function that will compare two strings. Why can't you just compare strings using `==`?

30. How many different keys for the Vigenere cipher can be produced with an alphabet of 26 characters? (a) 26! (b) 26n (c) 26^n (d) n^{26}

31. What does this program do?

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  printf("%d ", argc);
  if (argc) main(argc - 1, argv);
  else printf("\n");
  return 0;
}
```

32. What is the printout of this program? `k`, `pk`, `ppk`, `&k`, `&pk`, or `&ppk`?

```
#include <stdio.h>
int main() {
    int k = 3;
    int *pk;
    int **ppk;

    pk = &k;
    ppk = &pk;

    printf("%x", (*(&(*(*ppk)))));
}
```

33. Write one line of code that would dynamically allocate an array of 8 `doubles`.

34. What do we call it if we do not `free` all `malloc`ed memory? What is another major error that we can make involving the use of `free`?

35. Write a function that runs in $O(n^2)$ that takes as parameters (1) an array of unsorted integers and (2) the size of that array, and prints out the sorted array.

36. **Challenge question.** Explain what is happening in this program:

```
#include <stdio.h>
#include <string.h>

void baz(char *qux, int bar);

int main(int argc, char *argv[])
{
    if(argc != 2)
        return 1;

    baz(argv[1], strlen(argv[1]));
}

void baz(char *qux, int bar)
{
    char foo1[bar+1];
    strcpy(foo1, qux);
    char foo2[bar+1], *xyzzy1, *xyzzy2;

    xyzzy1 = foo1 + bar - 1;
    xyzzy2 = foo2;

    while(xyzzy1 >= foo1)
        *xyzzy2++ = *xyzzy1--;

    *xyzzy2 = '\0';

    printf("%s %s\n", foo1, foo2);

    return;
}
```