Answers to Midterm Review Questions from Studio 6

Command line

No, don’t panic. The midterm exam will not expect you to be an expert on using the command line. We will, however, expect you to be able to do a few simple things.

- Understand the concept of a current directory. How would I identify my current directory? I.e., how would I learn my current directory if I don’t know what it is?
  
  `> pwd`

- Understand how to navigate the directory structure. How do I move from one directory to another?

  `> cd directoryname`

- How do I learn what files are present in the current directory?

  `> ls`

- Understand how to edit text files. Can you use an editor available from the command line to make simple changes to a text file?
- Understand how and what it means to check out, update, and commit to an SVN repository.

  `> svn co reponame`

  `> svn update`

  `> svn commit -m "message"`

Information representation

Now we’re getting into the heart of the matter. This is stuff you should definitely expect to be asked about.

In general, don’t be surprised when I ask about how some data type is different in Arduino C vs. Java.

Number bases

Know about conversion back and forth between any of decimal, binary, and hexadecimal.
• Convert from binary to hex: 01011010, 111111100111100, 01100111
  \[ \text{0x5a, 0xff3c, 0x67} \]

• Convert from hex to binary: 0xfe, 0x4321, 0x20
  \[ \begin{array}{c}
  11111110 \\
  0100001100100001 \\
  00100000
  \end{array} \]

• Convert from hex to decimal: 0x132, 0xcd, 0x11
  \[ \begin{array}{c}
  0x123 = 1 \times 256 + 3 \times 16 + 2 = 306 \\
  0xcd = 12 \times 16 + 13 = 205 \\
  0x11 = 1 \times 16 + 1 = 17
  \end{array} \]

• Convert from decimal to hex: 723 = 0x2d3, 100 = 0x64, 50 = 0x32

**Integer and fixed-point representations**

Know about two’s complement representation, as well as excess notation and sign-magnitude notation. You also want to know the basics of fractional numbers.

• How do you represent \(-1\) in the following forms: 8-bit two’s complement, 8-bit sign-magnitude, 8-bit excess-127.
  \[ \begin{array}{c}
  11111111 \\
  10000001 \\
  01111110
  \end{array} \]

• How do you represent \(-2\) in these forms?
  \[ \begin{array}{c}
  11111110 \\
  10000010 \\
  01111101
  \end{array} \]

• What is the weight of the most significant bit in 8-bit two’s complement?
  \[-128\]

• What is the weight of the least significant bit in 8-bit two’s complement?
  \[1\]

• How many fractional bits are in a Q1.15 number?
  \[15\]

**Floating point representation**
I’ll keep the floating point questions relatively simple (that’s code for “I won’t give you a 32-bit binary number and ask what it means as an IEEE floating point number.”). Actually, I gave you enough information in lecture to be able to do that, but it’s icky and I promise not to ask. There are some questions, though, that are fair game, such as:

- How many bits are in the exponent of a float in Java, or in C?
  
  8, both languages

- How many bytes in a double in C, in Java?

  4 in Arduino C, 8 in Java

- What form is used to represent negative numbers for the number as a whole?

  Sign-magnitude

- For the exponent?

  Excess notation (or offset notation)

**Strings**

We have discussed strings in three different formats this semester: null-terminated arrays of single-byte characters in Arduino C, the String class in Java (under the hood using two-byte UTF-16 characters), and UTF-8 strings, which have a two-byte size followed by a sequence of flexible width (single-byte in our case) characters.

- Use an ASCII table to look up the hex encoding for “ASCII table” and author an Arduino sketch using Serial.write() to send the string ASCII table to the Serial Monitor on the PC. Use hex values as arguments to Serial.write() (i.e., not Serial.write('A'): Serial.write('S'); ...).

  0x41 0x53 0x43 0x49 0x49 0x20 0x74 0x61 0x62 0x65

- What else do you need to send (e.g., via Serial.write()) so that further (i.e., subsequent) versions of your string are printed on the next line in Serial Monitor? Try it.

  Need to send a ‘\r’ character.

**C language**

There are a number of things we have investigated where C is somewhat different than Java. Can you name a few? Maybe more important, will you recognize them when you come across them in code? (e.g., declarations of arrays.)
There are many things where Arduino C and Java are quite alike. (e.g., indexing into arrays.)

- In C, suppose $x$ and $y$ are declared as `char` and have values 0x54 and 0xab, respectively. What are the values of the following C expressions:
  
  $\begin{align*}
  x & \text{ & } y & \quad 0x54 \text{ & } 0xab = 01010100 \text{ & } 10101111 = 00000000 = 0x00 \\
  x & \text{ | } y & \quad 0x54 \text{ | } 0xab = 01010100 \text{ | } 10101111 = 11111111 = 0xff \\
  ~x & \text{ | } ~y & \quad ~0x54 \text{ | } ~0xab = 10101011 \text{ | } 01010100 = 11111111 = 0xff \\
  ~x & \text{ & } !y & \quad ~0x54 \text{ & } !0xab = 10101011 \text{ & } 00000000 = 00000000 = 0x00 \\
  x & \& \& y & \quad 0x54 \text{ & } 0xab = 00000001 = 0x01 \\
  x \text{ || } y & \quad 0x54 \text{ || } 0xab = 00000001 = 0x01 \\
  !x \text{ || } !y & \quad !0x54 \text{ || } !0xab = 00000000 \text{ || } 00000000 = 00000000 = 0x00 \\
  ~x & \& \& ~y & \quad ~0x54 \text{ & } ~0xab = 10101011 \text{ & } ~01010100 = 00000001 = 0x01
  \end{align*}$

Timing

The focus of our timing investigations have clearly been on delta-timing techniques.

- When using `delay()` based timing, what are the things that can cause the iteration time to be inaccurate?

  Pretty much anything else in the loop will cause delay-based timing to be slow.

- When using delta-time techniques, what can still cause the iteration time to be inaccurate (i.e., delta-time techniques don’t handle everything, what do they still leave for the programmer to ensure is correct)?

  If the computation takes longer than the loop-time, delta-time techniques cannot keep up.

Analog inputs

Given a spec from some arbitrary input device, can you write code that reads from an analog input channel and converts the A/D counts into the appropriate engineering units?

- Hook up the 3.3V pin on the Arduino to a free analog input pin. Author a sketch to read the analog value and print it to the LCD or the Serial Monitor. Approximately what value do you expect to see? Why?

  $1023 \times 3.3/5 = 675$
• Hook up the 3.3V pin on the Arduino to the AREF analog input reference pin (they are right next to one another). Using an external reference (see `analogReference()`), alter your temperature conversion code to again properly read degrees C. Try it out. Is it reasonable? (Don’t worry if it’s a few degrees off, the reference voltages aren’t that precise.)

**Analog outputs**

Have you noticed yet that all of the linear transformations we’ve been doing on analog inputs work pretty much exactly the same way on analog outputs? Given the spec of an analog output device, can you write code that, given a desired output level in engineering units, converts that into a value to pass to the PWM output?

• What is the range of values that can be passed to `analogWrite()`?

  0 to 255

• Draw the waveform that results when `analogWrite()` is invoked with a value of 64.

  Approximately ¼ of the time high, with a period of 2 ms.

**Digital outputs**

We’ve not only generated software that controls digital outputs, we’ve also built the circuits that those digital outputs control.

• With an LED, to get it to light, should the anode have a higher voltage than the cathode, or the other way around?

  Anode needs to have higher voltage than the cathode

• What is a reasonable resistor value to put in series with an LED to both physically protect the circuitry and also have a reasonable brightness?

  Anything between 200 and 500 ohms will work just fine.

• Does it matter what is the orientation (i.e., direction) of the resistor?

  No

• Does it matter whether the LED is first and the resistor is second, or if the resistor is first and the LED is second?

  No
• What resistor value corresponds to the color code red-red-brown-gold?
   
   \[ 22 \times 10 = 220 \text{ ohms} \]

• yellow-orange-red-gold? (The gold indicates tolerance, you can ignore that part. I just included it for completeness.)
   
   \[ 43 \times 100 = 4300 \text{ ohms} \]

**Streams and communication**

• What are the basic operations on a stream (receive/send a byte)? In Java, how does this change if you have a `DataInputSteam` object?
• What is a magic number and what is it good for?
• This is a thought exercise, don’t really write any of this code. Consider how you would design a protocol and write the corresponding code to transmit and receive Java objects (we are only interested in the instance variables appropriate for these Java objects): (1) a rectangle, (2) a pencil (first design the object, then design the protocol). What would your Java objects look like in Arudino C?