**Question 1:** At the end of the interview with Prof. Papadimitriou, the interviewer commented that “the history of computer science -- the discipline -- was actually built by Jewish exiles, by queer men, by women.” Name one of the people described in this comment. 

The 3 people described in this comment are Alan Turing, John von Neumann, and Klara (Dan) von Neumann. We accepted “Turing” and “von Neumann.”

**Question 2:** Give three examples of sensors used in computing and their function.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermometer</td>
<td>Measure temperature</td>
</tr>
<tr>
<td>GPS</td>
<td>Detect physical location on earth</td>
</tr>
<tr>
<td>Microphone</td>
<td>Capture sound</td>
</tr>
<tr>
<td>Camera</td>
<td>Capture images through light</td>
</tr>
<tr>
<td>Mouse</td>
<td>Measure hand motion</td>
</tr>
</tbody>
</table>

We accepted a large variety of answers here. In general: A sensor is a device that measures a physical quantity and returns a numerical value. Examples we accepted include: Thermometer | Measure temperature, GPS | Detect physical location on earth, Microphone | Capture sound, Camera | Capture images through light, Mouse | Measure hand motion. See the Gradescope rubric for a long list of accepted answers. We accepted 2 of 3 lines correctly completed in the table for full score.

**Question 3:** For each of the following statements, fill in the blanks with one of the following terms: procedure, function, input, output, domain, range, data type, side effect

- The difference between a reporter and predicate block is in their range.
- A blank in a block is called a(n) input. The shape of that blank specifies the expected data type.
- The shape of the block tells you whether or not to expect a(n) output.

- A reporter and a predicate can have the same output (if they report true or false), but the real difference is the set of possible outputs, which is the range.
- The domain refers to the set of inputs that a procedure accepts, but since we are just asking about the input blank, the correct answer is data type. You were definitely on the right track if you put domain.
- The shape of the block tells you whether it is a command, reporter, or predicate block. This determines whether there should be an output (reporter or predicate) or not (command). A side effect is determined by the implementation of a block, which you are not privy to thanks to abstraction. While most command blocks you have seen do have side effects (e.g. move, say, pen), a few counter examples include most of the control blocks (e.g. report, run) or mutating a local/script variable (e.g. set, change, add, delete).

**Question 4:**

a. We want to store a binary encoding of the 150 original Pokemon. How many bits do we need to use?

8 bits (a byte) $2^7 = 128 < 150 < 256 = 2^8$

b. What is the encoding for Pikachu (#25)?

$25 = 16 + 8 + 1 = 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 0b11001$. Because we are using an encoding for all 150 Pokemon, the encoding needs to be 8 bits long, so the answer is 0b00011001.

**Question 5:** Provide both a positive and negative consequence of one of the algorithms described in the article “Algorithms Are Great and All, But They Can Also Ruin Lives.”

We were pretty generous with grading this question. The point was to talk about a positive and negative of the SAME algorithm from the article, but we graded them independently. Some of the algorithms from the article were DMV/RMV facial recognition, profiling terrorists, and targeted advertising/recommendations.
e) I’m getting the message “cannot execute binary file”. In one or two sentences, explain what the problem is and how to fix it. (1pt w/explanation)

Assuming not a corrupted file, then the executable was compiled on another machine with a different architecture and can’t be read. **Re-compile on the current machine.**

f) In our 32-bit single-precision floating point representation, we decide to convert one significand bit to an exponent bit. How many **denormalized numbers** do we have relative to before? (Circle one)

More

Fewer

Half as many because lost a significand bit (1 pt)

Rounded to the nearest power of 2, how many denorm numbers are there in our new format? (Answer in IEC format) (1 pt)

22 significand bits + sign bit but not counting ±0, so exactly $2^{23}-2$ denoms

__8 Mebi #s___

**Question 2:** Flippin’ Fo’ Fun (10 points, 14 minutes)

Assume that the most significant bit (MSB) of $x$ is a 0. We store the result of flipping $x$’s bits into $y$.

Interpreted in the following number representations, how large is the magnitude of $y$ relative to the magnitude of $x$? Circle ONE choice per row. (2 pts each)

|                     | $|y| < |x|$ | $|y| = |x|$ | $|y| > |x|$ | Can’t Tell |
|---------------------|-----------|-----------|-----------|------------|
| **Unsigned**        |           |           |           |            |
| **One’s Complement**|           |           |           |            |
| **Two’s Complement**|           |           |           |            |
| **Sign and Magnitude**|         |           |           |            |
| **Biased Notation** (e.g. FP exponent) |         |           |           |            |

- In unsigned, a number with the MSB of 1 is always greater than one with a MSB of 0.
- In one’s complement, flipping all of the bits is the negation procedure, so the magnitude will be the same.
- In two’s complement, $y$ is a negative number. Its magnitude can be found by applying the negation procedure, which is flipping the bits and then adding 1, resulting in a larger magnitude than $x$.
- In sign and magnitude, the 2"nd" MSB bit will determine the relative magnitudes of $x$ and $y$, so you can’t tell for certain.
- In biased notation, you read the number the same as unsigned but apply a constant bias to BOTH numbers, so the relation is the same as in unsigned numbers.
**Question 1: Reppin’ Yo Numbas** (10 points, 16 minutes)

For this question, we are using 16-bit numerals. For Floating Point, use 1 sign bit, 5 exponent bits, and 10 mantissa bits. For Biased use a bias of $-2^{15}+1$.

**Scoring:** +1pt for every correctly filled blank, -0.5 for additional incorrect responses in part (a), minimum of 0.

a) Indicate in which representation(s) the numeral is closest to zero: Two’s Complement (T), Floating Point (F), or Biased (B). The first one has been done for you. NaN is not valid in comparisons.

<table>
<thead>
<tr>
<th>Numeral:</th>
<th>Closest to zero:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) 0x0000</td>
<td>TF</td>
</tr>
<tr>
<td>2) 0xFFFF</td>
<td>T</td>
</tr>
<tr>
<td>3) 0x0001</td>
<td>F</td>
</tr>
<tr>
<td>4) 0xFFFFE</td>
<td>T</td>
</tr>
<tr>
<td>5) 0x8000</td>
<td>F</td>
</tr>
<tr>
<td>6) 0x7FFF</td>
<td>B</td>
</tr>
</tbody>
</table>

b) We now wish to add the numerals from **top to bottom (1 to 6)**. However, it is possible that we encounter an error when performing these addition operations. For each number representation, state the FIRST error that is encountered and which numeral causes it; if no error is encountered, answer “no error.”

Possible arithmetic errors are: OVERFLOW, UNDERFLOW, NaN, and Rounding (assume we are rounding using a truncating scheme).

<table>
<thead>
<tr>
<th>Representation:</th>
<th>Arithmetic Error:</th>
<th>Numeral #:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two’s Complement</td>
<td><strong>Overflow</strong></td>
<td>5</td>
</tr>
<tr>
<td>Floating Point</td>
<td><strong>NaN</strong></td>
<td>2</td>
</tr>
<tr>
<td>Biased</td>
<td><strong>No Error</strong></td>
<td>x (this blank was worth zero points)</td>
</tr>
</tbody>
</table>
Question 2: This Problem is Like a Box of Chocolates (27 points, 48 minutes)

a) 4 points. In C, char is in fact a signed variable type. Assume we have 4 chars != 0x00 loaded into a 32-bit int, one in each byte. Complete the function below that will negate the char in byte i of the int “in place”, with byte 0 being the least significant and byte 3 being the most significant.

```c
void negByte(int *data, char i) {
    *((char *)data)+3-i) = -*((char *)data)+3-i);
}
```

Also accepted: *data = *data^(0xFF << (8*i)) + (1 << (8*i))

Parts of the problem:
+1 point for dereferencing data and storing the value back
+1 point for manipulating the memory location of the i-th byte within data
+2 points for correctly negating the byte. This meant both flipping the bits and adding one if students took the masking approach.
Because of the wide range of responses, scoring was done holistically. Deductions of 0.5 points were given for small mistakes (eg. Issues with parentheses).

b) 4 points. You have been given access to a version of C that does not have the sizeof function implemented. However, this version of C still knows how large each type is internally (and a char is still one byte). Implement a constant time sizeof operation by filling in the blank below. Note that your sizeof operation will take in a variable of the type you wish to find the size of, rather than the type itself.

```c
#define sizeof(type) ((char *)&type+1) – (char *)&type)
```

Answers to this problem were widely varied. Points were awarded based on how close it was to the correct answer. In particular, it was important to use pointer arithmetic on the address of the variable.
Answers that dealt with addresses received 1 point, and answers that showed some sort of tangential understanding regarding variable sizes received 0.5 points. Simply using strlen() or len() on type received no credit.

c) 5 points. For the following MIPS code, fill in the branch immediates (in decimal) AFTER pseudo-instruction replacement. Then fill out the relocation and symbol tables. Recall that the assembler will assign addresses starting at 0x0.

Write addresses in hexadecimal and do not show leading zeros. The tables are part of an object file and only recognize TAL.
Question 1: Number Representation (15 pts)

a) Complete the tables below: (6 pts)

<table>
<thead>
<tr>
<th>Base 8</th>
<th>Hexadecimal</th>
<th>Convert unsigned integers: (4 pts)</th>
<th>Convert to and from IEC prefixes: (2 pts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>115&lt;sub&gt;8&lt;/sub&gt;</td>
<td>0x4D</td>
<td>115 = 2&lt;sup&gt;4&lt;/sup&gt; bits</td>
<td>16 Pebi-bits</td>
</tr>
<tr>
<td>32&lt;sub&gt;8&lt;/sub&gt;</td>
<td>0x1A</td>
<td>2048 students</td>
<td>2 Kibi students</td>
</tr>
</tbody>
</table>

b) Due to limitations in storage space, we are using only 4 bits to represent integers.

1) What is the most negative 2’s complement signed integer (decimal) we can represent? (1 pt)

\[-2^{n-1} = -2^3 = -8\]

2) What is the value (decimal) of the 2’s complement number 0b1010? (1 pt)

-6

3) Write a number (binary) that, when added to 0b0100, will cause signed overflow. (2 pts)

0b0100 > ANS > 0b0111

C) An amino acid is defined by a set of 3 consecutive nucleotides (A, C, G, or T). For example, ATG is Methionine. All combinations are unique (e.g. ATG ≠ AGT ≠ GTA).

1) How many total possible amino acids are there? (1 pt)

\[4^3 = 64\text{ amino acids}\]

2) In reality, there are 21 amino acids found in the human body. How many bits would it take to encode these amino acids in binary? (1 pt)

\[\text{ceil(log}_2(21)) = 5\text{ bits}\]

3) Scientists also use single-digit encodings for amino acids (e.g. ‘A’ for Alanine). In a single sentence, explain why it is okay that we use A for the amino acid Alanine, the nucleotide adanine, and the hex representation of the decimal number 10. (1 pt)

They are different representation systems. We need to know the appropriate interpretation to distinguish the values for our application.

4) We wish to encode the 21 amino acids in base 2, 3, or 5. Which of these choices allows for the MOST new amino acids discoveries before needing to increase the number of digits and how many new discoveries are allowed in this choice? (2 pts)

| Base: 2 | Possible New Discoveries: \(32 - 21 = 11\) |
**Question 2: C Potpourri (12 pts)**

a) Given the library function `rand()` that returns a random number between 0 and \((2^{32})-1\) when called, write a valid C expression that uses *bit operations* \((^, ~, |, \&)\) to initialize the variable `r` with a random integer between 0 and `n`, which is some power of 2 less than \((2^{32})-1\). *(3 pts)*

```c
int n = 8; // In this case, we want r to contain one of {0,1,2,3,4,5,6,7}.
int r = rand() & (n - 1);
```

b) Sally Stanfurd tells you that using a random number generator function is silly in C. She claims that since local variables are not automatically initialized we can use the garbage contained in them as random values. Does Sally's function produce truly random values (circle one)? **Yes** // **No** *(2 pts)*

*Briefly explain why or why not.*

The garbage contained in local variables is not truly random and is somewhat predictable given knowledge of prior functions and their associated stack frames.

c) Complete the implementation of the integer array shuffle function below, which randomizes the ordering of the first `n` entries of a given integer array. Each of the first `n` entries in the array should be swapped with an earlier entry exactly once, and the rest of the array should be left unchanged. Assume you have access to a function `random(int r)` that returns a number between 0 and `r-1`, inclusive. *(5 pts)*

```c
void shuffle(int* array, int n) {
    for (int i = n - 1; i > 0; i--) {
        int newIndex = random(i);
        int temp = array[newIndex];
        *(array + newIndex) = array[i];
        array[i] = temp;
    }
}
```

d) Assume integers are 32 bits. Instead of passing in an integer array as expected, we decide to pass in a string to our shuffle function as shown: *(2 pts)*

```c
char str[] = "fee fie foh fum ";
shuffle((int*) str, strlen(str)/sizeof(int));
printf("shuffle result: %s", str);
```

From the following choices, circle ONE of the following if it is a **possible** result of the `printf` statement:

- **Runtime Error** (e.g. seg fault) "fum foh fee fie ", "ioe h ef uf fmef"
- **Compiler Error** (e.g. incompatible pointer types)