1.
How many bytes would the following data structures require?

```
	struct ucla {
		char blue[6];
		union {
			int gold;
			char joe[8];
		} bruin;

	} arr[4];
```

2.
What do I need to do if I want to access a function through buffer overflow (what needs to be done to the stack)?
The function's address is 0x500142. Also remember we're working in little endian!

*The function Gets is similar to the standard library function gets—it reads a string from standard input (terminated by '\n' or end-of-file) and stores it (along with a null terminator) at the specified destination (such as a char array previously declared). Functions Gets() and gets() have no way to determine whether their destination buffers are large enough to store the string they read.*

```
Dump of assembler code for function getbuf:
=> 0x0000000000601748 <+0>:       push   %rax
   0x000000000060174c <+4>:       sub    $0x40,%rsp
   0x000000000060174f <+7>:       mov    %rsp,%rdi
   0x0000000000601754 <+12>:      callq  0x40198a <Gets>
   0x0000000000601759 <+17>:      add    $0x40,%rsp
   0x000000000060175d <+21>:      pop    %rax
   0x000000000060175f <+23>:      retq
```

3.
What is the value of the following 8-bit, tiny floating point number? Note that the exponent field is 4 bits, and the fractional field is 3.
```
	01100000
```

What about the single precision 32-bit floating point number?
```
	01000010000001000000000000000000  or  0x42040000
```

What is -13.1875 in single precision 32-bit floating point format?

4. Designing a (Better?) Floating Point

What do we want to represent with floating point numbers?

Is it possible to represent all of the numbers we would like to represent?

How can we deal with the above?

What qualities do we want from our representation?

So approximation (and hence precision) is going to play a large role in the design of our floating point numbers. How can we build in the idea of precision into our numbers? (We would like our numbers to be as precise as possible. Think about how errors build when we perform arithmetic operations - maybe what you learned about significant figures will help)

What are some problems with the representation you came up with, and how can they be addressed?

Some possible issues to consider:

- Range of numbers?
- Precision of numbers?
- Overflow?
- Underflow?
- Bit efficiency?
- Representation(s) of 0?
- Unique representations?
- Rounding?

5.
What are some optimizations that can be made to the following function?

```
void cs33fun(char* Midterm, char* Grade, int* Final, int n) {

      for (int i = 0; i < (strlen(Midterm)); i++) {
            strcat(Grade, Midterm);

            for (int j = 0; j < n; j++)
                  for (int k = 0; k < i; k++)
                        Final[j] += strlen(Grade);
      }
}
```