Collaboration

You must work on this quiz with a partner. You should have listed a partner on the Google form available on Moodle *at class time*, and received a confirmation message regarding your partner from the professor.

Deliverables

You should deliver a **PDF** file containing the answers to the questions, and a file containing the implementations of the functions in the skeleton code. Submit a single **compressed file** containing both files.

1. [4 pts] Chapter 2, exercise 12.
2. [6 pts] Chapter 3, exercises 8, 9, 10.
3. [10 pts] Implement the missing `reverse()`, `itoa()`, `myputs()`, and the `myprintf()` functions provided in the skeleton code. The output of the tests, in the `main()` function, should be:

   ```
   Number is 132
   Negative number is -132
   Unsigned number is 4096
   Unsigned number in hex is 0x1000
   Long number is 5000000000
   Long negative number is -5000000000
   ```

   Here is the suggested order of implementation:

   (a) Implement the `reverse()` function, which reverses the string in `buffer`. Note that since `buffer` is a string, it is terminated with a null character `'\0'`. You can use `strlen()` to find the length of the string. Your function should only reverse the non-null characters of the string, keeping the null character in its original position.

   (b) Implement the `itoa()` function. This function converts the integer `number` passed as argument into a string stored in `buffer`. The parameter `base` indicates whether the resulting string is a base 10 or 16 (the only supported options) representation of `number`.

i. Feel free to generate the number in reverse into `buffer`, and then use your previous `reverse()` function. That might be convenient if you obtain the individual digits of `number` by doing subsequent divisions and rest-of-division operations.

ii. Negative numbers should have a minus sign preprending the number.

iii. Numbers in hexadecimal should be given in the form `0xdadada` or `-0xdadada` for positive and negative numbers, respectively.

iv. Your function should return a pointer to (the updated) `buffer`.

(c) Implement the `myputs()` function. This is a straightforward function that uses a loop and `putchar()` to output characters into the screen.

(d) Implement the `myprintf()` function. This function is a subset of the `printf()` function, and received an initial string argument (`format`) and an unbounded sequence of extra parameters (indicated in ..., which is valid C). Your function basically iterates over the characters of the string passed in format, and while you don't read a `'%'` character, you call `putchar()` to output them into the terminal.

i. If you see a `'%'` character, check if the the next character is one of $\{d, u, x, l\}$, associated with integers, unsigned integers, hexadecimals, and long integers, respectively.

If you see a `%d`, for instance, you should use *obtain* (see below) the *next* argument in the variable argument list treated as an integer. Then, you can use your `itoa()` function to get the string representation of that integer, and output in the terminal using `myputs()`. The workflow for the other cases ($\{u, x, l\}$) is similar.

Your should use `va_list`, `va_start`, and `va_arg` to obtain the parameters in the "..." unbounded parameter sequence. You can find all information you need by using the manual page:

```
man stdarg
```