Solutions for 1.

a. ANS: The program begins an infinite loop, continually outputting:

```
My guess is 2914745102!
Aw, my guess was too low.  I'll try again.
```

b. ANS:

```
(gdb) p hi
$1 = 1
(gdb) p lo
$2 = 10000
```

c. ANS: No, because even though the variables are reversed (hi should be 10000, lo should be 1), the code within the function treats them as interchangable, adding them together and dividing by 2.

d. ANS:

```
My guess is 4294958494!

(gdb) p guess
$3 = 5000
```

e. ANS: It is AN error (the address of `guess` is being printed), but it isn't the source of the infinite loop error because `guess` is still 5000, as it should be when `hi` and `lo` are averaged.

f. ANS: Eventually, the values of `hi` and `lo` stop changing:

```
Breakpoint 2, takeGuess (lo=1249, hi=1) at q1.c:100
100            uint16_t guess = (hi+lo)/2;
(gdb) c
Continuing.
My guess is 4294958494!
Aw, my guess was too low.  I'll try again.

Breakpoint 2, takeGuess (lo=1249, hi=1) at q1.c:100
```

g. ANS: The students could try to argue that this is the error, but I would argue that it isn't; it's merely a symptom of the error. The actual error lies elsewhere.

h. ANS:

```
(gdb) p guessReturnCode
$3 = 255 '\377'
```

i. ANS: No, the return value should be -1, not 255.

```
(gdb) list 113
108       *   Post: prints a correct/incorrect message and returns
corresponding val
109       *                       returns  1 if guess is too high
110       *                       returns  0 if guess is correct
111       *                       returns -1 if guess is too low
112       */
113      uint8_t handleGuess(uint16_t guess, uint16_t myNum) {
114            if (guess == myNum) {
115                  printf("Hooray!  I guessed correctly!\n\n");
116                  return 0;
117            } else if (guess > myNum) {
```

j. ANS: The `handleGuess()` function returns a `uint8_t` variable, but tries to use -1 as a return code. `main()` sees 255 instead of -1, and can never update `lowPossible` on line 53. This could be fixed by either (1) returning an `int8_t` variable (better answer), changing the return code to a positive number like 2 instead of -1 (also a better answer), or by changing the `else if` command on line 52 to just an `else` command (works, but is sloppy code).

k. ANS: Input will work as long as `lowPossible` never needs to be updated. An input of 1 will work, as will all of the integer divisions along the way to guessing 1 (5000, 2500, 1250, etc.). Most input will not work, since `lowPossible` needs to be updated anytime the computer guesses too low.

Solutions for 2. ANS: I introduced these two bugs into this version:

1. The input validation has an extra 0, accepting input from 1 to 100000 inclusive. But numbers between 10001 and 100000 cannot be guessed.
2. Averaging is done incorrectly, now taking `(hi-lo)/2` instead of `(hi+lo)/2`, and causing a different infinite loop.

Students should submit enough gdb output to show how they have determined these errors. For example, you could detect bug #1 with these snippets:

```
(gdb) list 14
9     uint16_t getUserInput(char*);
10    uint16_t takeGuess(uint16_t, uint16_t);
11    uint8_t handleGuess(uint16_t, uint16_t);
12    void delay(void);
13
14    /**  This program takes an integer between 1 and 10000 inclusive as
15     *   command line input, and then attempts to guess the input number
16     *   in as few guesses as possible.
17     */
18    int main(int argc, char** argv) {

(gdb) b checkInput
Breakpoint 1 at 0x4006c2: file q2.c, line 68.
(gdb) run 12345
Starting program: /home/jwenskovitch/sp18_c02/q2/q2 12345
Breakpoint 1, checkInput (numargs=2, args=0x7fffffffdeb8) at q2.c:68
68          if (numargs != 2) {
(gdb) n
74        uint16_t intInput = getUserInput(args[1]);
(gdb) n
76        if (intInput < 1 || intInput > 100000) {
(gdb) n
81          return 0;
(gdb) p intInput
$1 = 12345
```