Here are a couple of function documentation examples from previous teams:

```
// ****************_getKey******************
//  Gets a key from the user
//  Stores in a variable to determine which key was pressed
// then performs a jump to ht_addBuf, ht_delBuf, ht_clearBuf,
// or none (Jumps back to start) depending on the detected key.
//***************key variables***************
// key - variable used to determine which key was pressed
// by comparing it to the numerical values assigned
// to the 0, 1, backspace, c, q, and Enter keys
// q - flag variable which determined whether
// the q key was pressed by the user or not
// if it is positive, jumps to END infinite loop
// otherwise, loops back to start
// clearPostProcessing
// - flag variable used to determine if c or Enter
// was pressed after the 16 bit binary input was
// converted to decimal. If positive, clears buffer
// and screen. Otherwise, processes binary input.
```

****

```
// C++ pseudocode implementation

// while (true)
// {
//     while (key != 0)
//     {
//         if (key == 0)
//         {
//             // pass
//         }
//     }
// }

// Data Table
//
// returns:
// zz_key - the key which was pressed and released
//
// variables:
// zz_loopptr - pointer to start of one of the two while loops
//
```

```
// labels:
// zz_GETKEY - function entrance and start of main while loop
// zz_PRESSEDLOOP - the start of the while loop for when a key
is being held
// zz_BREAK - restart the current while loop
```

For the key actions documentation, use of pseudo code to make the actions clear is appropriate so the flow of the code is clear. My suggestion in class is to start your implementation with pseudo code in comments and then write HACK code underneath them (a common CS approach). Note that this is not the same as simply using a comment to explain what a Hack instruction is doing.

Here are a couple of function documentation examples from previous teams:

```
//SPECIAL CASE -32768 must be hard coded
@zz_PROCESSBUF_decimal
D=M
@zz_OVERFLOW
D;JLT // if the number is less than zero jump

///Check if positive or negative here.
/// If neg, print a - and make the number positive

@zz_PROCESSBUF_negtoken
D=M
@zz_DISPLAY_NEGATIVE
D+1;JEQ// if greater than 0 (-1 means neg) jump


//else display the positive
@zz_ge_next_+
D=A
@ge_output_return
M=D


**************************

// START OF SPECIAL CASES
// IF TOTAL == 0 OR IF TOTAL == -0 AKA -32678
@qq_SUM_TOTAL
D=M // GET TOTAL FOR SPECIAL CASES
```

```
@qq_GET_10000_DIGIT
D;JNE // JUMPS IF TOTAL==0

    @qq_1_DIGIT
    M=0 // ASSUMES TOTAL IS 0 AND STORES ACCORDINGLY

    @R0 // BUT THEN CHECKS SIGN
    D=M // GETS SIGN INTO D FOR COMPARISON

    @qq_DecToDigs_END
    D;JEQ // IF THE SIGN IS 0 (POSITIVE) IT JUMPS TO END

    @3
    D=A // OTHERWISE THE NUMBER MUST BE -32768
    @qq_10000_DIGIT // AND SETS ALL DIGITS ACCORDINGLY
    M=D
```