```
############################################################
#           cs315 Week 9
#
#   ->  Floating point multiplication
#
############################################################

floating point multiplication:
    1) to multiply 2 floating point numbers, if the signs are the same the sign of the result will be positive or 0, if they are different the sign of
    the result will be negative or 1. (essentially XOR the signs to get the final sign)

    2) un-bias the two exponents add the true numbers and then bias the result. This is your tentative exponent for your result.
        actual exponent (#1) = bias exponent (#1) - 127
        actual exponent (#2) = bias exponent (#2) - 127

        tentative exponent of the result = actual exponent (#1) + actual exponent (#2) + 127

    3) then multiple the two 8 bit (including the hidden bit) unsigned fractional parts. since the most significant bits of the numbers you are
    multiplying are 1's your result will always be either 15 or 16 bits.

    --> If the 16th bit is a 1 add 1 to the tentative exponent and this gives you the exponent for your result.
    --> If the 16th bit is a 0 the tentative exponent is the exponent of your result.

    * take the eight most significant bits from the multiplication result (starting with the left most bit). This is your fractional part including
       the hidden bit. Assemble the result.

#########################################################################################################
Example 1:

    -8.5 =>
        8.5 = 1000.1 = 1.0001 * 2^3

        1 - 10000010 (1) 0001000
        ^       ^           ^
       sign  exponent   magnitude

        0.5|
        1.0| 1  <-- (1.0 >= 1)


    +3.125 =>
        3.125 = 11.001 = 1.1001 * 2^1

        0 - 10000000 (1) 1001000
        ^       ^           ^
       sign  exponent   magnitude

        0.125|
        0.250| 1
        0.500| 0
        1.000| 1    <-- (1.0 >= 1)


    (-8.5) * (+3.125) = -26.5625 (expected)


    tentative exponent = 3 + 1 = 4
    sign of result = 1 XOR 0 = 1 <-- negative

        (1) 0001000
    *   (1) 1001000
    ----------------
```

```
    8 bit * 8 bit = 16 bits result

    carry:       1
            0000 0000 0000 0000
            0000 0000 0000 0000
            0000 0000 0000 0000
            0000 0100 0100 0000
            0000 0000 0000 0000
            0000 0000 0000 0000
            0010 0010 0000 0000
        *   0100 0100 0000 0000
        ------------------------
            0110 1210 0100 0000

    %2      0110 1010 0100 0000
            ^
    16th bit is 0, which means that the tentative exponent is the final exponent.

    Note:
        * if 16th bit was 1, then final exponent would be tentative exponent + '1'

    thus, final exponent = tentative exponent = 4

    result => 1 - 10000011 (1) 1010100

    check result to make sure it is correct:
        1.1010100 * 2^4 =
        11010.100 * 2^0 =
        11010.100           <== -26.5 (correct, close enough)

##############################################################################################################
Example 2:

        +2.5
    *   +3.5
    --------
        +8.75 (expected)

    2.5 => 10.1 => 1.01 * 2^1 => 0 - 10000000 (1) 0100000
    3.5 => 11.1 => 1.11 * 2^1 => 0 - 10000000 (1) 1100000

    sign of the result => 0 XOR 0 = 0 <-- positive

    tentative exponent => 1 + 1 = 2

        (1) 0100000     -->     10100000
    *   (1) 1100000     --> *   11100000
    ---------------             --------
    --> result should be 16 bits (8+8 = 16)


        111                     <-- carry
        0000 0000 0000 0000
        0000 0000 0000 0000
        0000 0000 0000 0000
        0000 0000 0000 0000
        0000 0000 0000 0000
        0001 0100 0000 0000
        0010 1000 0000 0000
    +   0101 0000 0000 0000
```

```
    -------------------------
        1222 1100 0000 0000

    %2: 1000 1100 0000 0000
        ^
    there is a 1 in 16th bit, hence, add one to tentative exponent

    tentative exponent => 2 + 1 = 3

    result => 1.0001100 * 2^3 = 1000.1100 => 0 - 10000010 (1) 0001100 <= +8.750 (expected)

##########################################################################################################
Example 3:

        +15.33
    *   +7.77
    --------
        +119.1141 (expected)


    15.33 => 1111.0101 = 1.1110101 * 2^3

        .330|
        .660| 0
        .332| 1
        .664| 0
        .338| 1


    7.77 => 111.1100 = 1.1111000 * 2^2

        .770|
        .540| 1
        .080| 1
        .160| 0
        .320| 0
        .640| 0

    15.33   => 0 - 10000010 (1) 1110101
    7.77    => 0 - 10000001 (1) 1111000

    sign of the result => 0 XOR 0 = 0 <-- positive

    tentative exponent => 2 + 3 = 5


        (1) 1110101     -->     11110101
    *   (1) 1111000     -->     11111000
    ---------------             --------
    --> result should be 16 bits (8+8 = 16)

        1233 3322 11
        0000 0000 0000 0000
        0000 0000 0000 0000
        0000 0000 0000 0000
        0000 0111 1010 1000
        0000 1111 0101 0000
        0001 1110 1010 0000
        0011 1101 0100 0000
    +   0111 1010 1000 0000
    -------------------------
        1356 7765 4321 1000
```

```
    %2: 1110 1101 0101 1000
        ^
    there is a 1 in 16th bit, hence, add one to tentative exponent

    tentative exponent => 5 + 1 = 6

    result => 1.1101101 * 2^6 = 1110110.1 => 0 - 10000101 (1) 1101101 <= +118.5 (expected)
```

```
##################################################################################################
```