Problem 214: Round It Out

Difficulty: Medium

Author: Brett Reynolds, Annapolis Junction, Maryland, United States

Originally Published: Code Quest 2023

Problem Background

Numbers can be extremely precise, but very often we don't really need them to be that precise. For example, when working with money, we only need two decimal places, since no currency uses denominations less than one hundredth of its main unit. If a number is excessively precise, we need to "round" it to a less precise number. However, there are a large number of ways to round numbers, and how you do the rounding makes a very big difference.

You most likely learned the "half up" rounding method in school. In this rounding method, numbers are rounded to the nearest value of the desired precision. For example, if rounding to whole numbers, 0.2 rounds down to 0; 0.8 rounds up to 1. However, 0.5 is the midpoint between 0 and 1; with the "half up" method, the halfway point will round up, to 1. Note that whenever we say "round up" we really mean "round away from 0;" -0.5 rounds to -1, even though this number has a lesser value. Remember to pay attention to more precise values; 0.500000000001 rounds up to 1.

Other rounding methods follow different rules. For this problem, you'll need to appropriately apply those rules to a variety of situations.

Problem Description

Please note that this problem will ask you to round numbers in multiple ways. This problem overrules the general guidelines for rounding numbers provided in the Reference Materials.

You will need to write a program that is able to implement the various rounding rules listed below to round provided numbers to a specific number of decimal places.

- Half Up (HU) - Round numbers to the nearest value of the desired precision. 5 rounds away from zero.
- Half Down (HD) - Round numbers to the nearest value of the desired precision. 5 rounds towards zero.
- Up (U) - Round all numbers to the next value of the desired precision, moving away from zero.
- Down (D) - Round all numbers to the next value of the desired precision, moving towards zero.
- Half Even (HE) - Round numbers to the nearest value of the desired precision. 5 rounds to the value that is evenly divisible by 2.
- Half Odd (HO) - Round numbers to the nearest value of the desired precision. 5 rounds to the value that is not evenly divisible by 2.

For example, if we are rounding the following numbers to one decimal place, the results with each rounding method would be:

Original Number	HU	HD	U	D	HE	HO
1.23	1.2	1.2	1.3	1.2	1.2	1.2
1.28	1.3	1.3	1.3	1.2	1.3	1.3
1.33	1.3	1.3	1.4	1.3	1.3	1.3
1.38	1.4	1.4	1.4	1.3	1.4	1.4
1.25	1.3	1.2	1.3	1.2	1.2	1.3
1.35	1.4	1.3	1.4	1.3	1.4	1.3
-1.25	-1.3	-1.2	-1.3	-1.2	-1.2	-1.3
-1.35	-1.4	-1.3	-1.4	-1.3	-1.4	-1.3

Again, the examples above are for rounding to one (1) decimal place. Rounding to zero (0) decimal places means rounding to an integer. Rounding to -1 decimal places means rounding to a multiple of ten.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include a single line with the following information, separated by spaces:

- A decimal number, X
- A one- or two-character code indicating the rounding method to be used:
  - HU = Half Up
  - HD = Half Down
  - U = Up
  - D = Down
  - HE = Half Even
  - HO = Half Odd
- An integer, P, indicating the number of decimal places to which X should be rounded

```
6
1.23 HU 1
1.23 HO 1
1.23 HD 1
5.0 U -1
5.0 HE -1
-4.205 D 2
```

Sample Output

For each test case, your program must round X to P decimal places using the indicated method and print a single line containing the resulting number. The decimal point should only be printed when needed. Trailing zeroes that occur after the decimal point should not be printed.

```
1.2
1.2
1.2
10
0
-4.2
```