CoE 164

Computing Platforms

Assessments Week 01

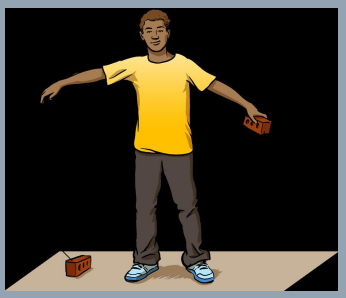
Academic Period: 2nd Semester AY 2022-2023
Workload: 3 hours
Synopsis: Rust programming basics

SE Week 01A

This assessment will probably be your first program in Rust.

This is worth 40% of your grade for this week.

Problem Statement



Graves, in his sixth-year grade school physics class, learned of *potential energy*, wherein objects have energy "stored" in them depending on their distance from some large object. He has recently been obsessed with putting his stuffed toys and other random objects at different heights and computing for their potential energy. For reference, the potential energy of an object on Earth with mass m located at height h from the center of the Earth is given by $E = mgh$. g here refers to the acceleration due to gravity pegged at $9.81\frac{m}{s^2}$.

Please help Graves compute the potential energy of all of his objects so that he can already go back to playing with his toys or the computer.

Input

The input consists of only two decimal numbers on a single line separated each by a space. They are m (in grams) and h (in meters), respectively. These represent the mass and height of an object from the center of the Earth.

Output

The output should consist only of one decimal number depicting the corresponding potential energy in joules. It should be rounded *down* (i.e. floored) to the nearest two decimal places.

Constraints

$m, h \in R$

$0 \leq m, h \leq 10000.00$

You can assume that all of the inputs are well-formed and are always provided within these constraints. You are not required to handle any errors.

Sample Input/Output

Sample Input 1:
`10 5`
Sample Output 1:
`0.49`

Sample Input 2:
`0 0`
Sample Output 2:
`0.00`

Sample Input 3:
`2 12`
Sample Output 3:
`0.23`

Steps

1. Write your program in Rust. Make sure to accept input via standard input and print your output via standard output.
2. Submit a copy of the source code to the Week 01A submission bin. Make sure that you attach one (1) file in the bin containing the Rust source code with a .rs extension or .rs.txt extension (if UVLe doesn't accept .rs files).

SE Week 01B

This assessment will let you be familiar with loops, conditionals, and math computations in Rust.

This is worth 30% of your grade for this week.

Problem Statement

Flappy Bird was a game released in 2013 on mobile phones. It was a viral sensation during that time due to its simple but "annoying" gameplay. A player controls a bird that falls downward by touching on the screen to make it jump a certain height. The game is over when the bird touches either an obstacle or the ground.

For your exercise in a new programming language named Rust, your instructor has given you a problem on recreating the game on the command line. To do that, you first need to know how the bird moves in the game. Flappy (our name for the bird) moves only upward or downward on the screen, while the background and obstacles move from right to left. Without external intervention, Flappy will fall downward at an acceleration of $9.81\frac{m}{s^2}$ due to gravity. Otherwise, they will jump a certain height and velocity. Due to gravity, Flappy's velocity will decrease until they fall downward again.

You start off by programming an algorithm that will check whether Flappy hit the ground given their current height from it and the time it took for the person to touch the screen from that height. You assume that at the current height, Flappy's velocity is zero and is starting to fall towards the ground. If Flappy's height is zero or less *at the time* the person touched the screen, the player will be greeted with a "game over" screen.

Input

The input starts with a number T on a single line denoting the number of test cases. T lines then follow, with each line denoting the current location of the bird. Each line consists of two space-separated decimal numbers h s denoting the current height of Flappy, and the time it took for the player to tap on the screen to make it jump.

Output

The output consists of T lines, with each line denoting the corresponding test case in the input. Each line should be of the format $Case\ \#t:\ RES$ where t is the serial of the test case starting from 1, and RES either a `YES` or a `NO` depending on whether it was game over for the player.

Constraints

$h, s \in R^{+}$
$0 \leq h, s \leq 100.00$

You can assume that all of the inputs are well-formed and are always provided within these constraints. You are not required to handle any errors.

Sample Input/Output

Sample Input 1:

```
5
5 0.3
10 0.1
4.3 1
1.1 0
2 0.64
```

Sample Output 1:

```
Case #1: NO
Case #2: NO
Case #3: YES
Case #4: NO
Case #5: YES
```

Steps

1. Write your program in Rust. Make sure to accept input via standard input and print your output via standard output.
2. Submit a copy of the source code to the Week 01B submission bin. Make sure that you attach one (1) file in the bin containing the Rust source code with a .rs extension or .rs.txt extension (if UVLe doesn't accept .rs files).

SE Week 01C

This assessment will let you be familiar with loops and conditionals in Rust.

This is worth 30% of your grade for this week.

Problem Statement

Documents that state some amount of money often have those amounts written in terms of both numerical and word. This promotes redundancy and prevents misread and easy manipulation of figures. Nowadays, most automated check clearers look at the numerical amount for clearing, but knowing how to write amounts in words is still a required skill especially if you have a checking account or write legal documents.

You are working as an intern in a remittance company. Since the main clients of your company are people from rural Southern Tagalog, they are more adept in conversing in Filipino than English. Hence, the company is requiring people who want to send or receive money write the amount in numerals and in words in Filipino. You are tasked to create an automated system that will clear these forms and file for sending or receiving money based on the amount written in numerals and words. This involves verifying that both amounts are the same.

As a review, Filipino numbers are based off of the following ten words for the numbers 1-10:

English	**Filipino**
one	isa
two	dalawa
three	tatlo
four	apat
five	lima
six	anim
seven	pito
eight	walo
nine	siyam
ten	sampu

Numbers between 11 and 19 inclusive can be said using the prefix “labing-” connected to one of the basic words from above. So, 16 is written as “labinganim”.

Numbers skipping by tens from 20 up to 90 inclusive can be written using the basic words and the suffix “-napu” (if the basic word ends with a consonant) or “-ngpu” (if the basic word ends with a vowel), meaning “and ten”. So, 30 is written as “tatlongpu” and 90 is written as “siyamnapu”. On the other hand, numbers between 21 and 99 inclusive are to be written as, in the case of 67, “sixty and seven”. So, combining the earlier patterns, we write 67 as “animnapu at pito”, which should be shortened to “animnapu’t pito”.

Numbers skipping by hundreds from 100 up to 900 inclusive can be written using the basic words and the phrase “na daan” (if the basic word ends with a consonant) or “-ng daan” (if the basic word ends with a vowel). So, 400 is written as “apat na daan” and 500 is written as “limang daan”. On the other hand, numbers between 101 and 999 inclusive are to be written as, in the case of 371, “three hundred and seventy and one”. So, combining the earlier patterns, we write 371 as “tatlong daan at pitongpu’t isa”.

Numbers skipping by thousands from 1000 up to 9000 inclusive can be written using the basic words and the phrase “na libo” (if the basic word ends with a consonant) or “-ng libo” (if the basic word ends with a vowel). So, 8000 is written as “walong libo”. On the other hand, numbers between 1001 and 9999 inclusive are to be written as, in the case of 5678, “five thousand and six hundred and seventy and eight”. So, combining the earlier patterns, we write 5678 as “limang libo at anim na daan at pitongpu’t walo”, which should be shortened to “limang libo’t anim na daan at pitongpu’t walo”.

Finally, numbers between 10000 and 99999 inclusive are to be written by first writing the two numbers in the thousands place using the same rules for the numbers from 10 to 99, which is then affixed by “libo”. Then the remaining numbers in the hundreds place are written using the same rules for the numbers from 0 to 999. So, 12001 is written as “labingdalawang libo’t isa” and 99999 is written as “siyamnapu’t siyam na libo’t siyam na daan at siyamnapu’t siyam”.

Written amounts in the remittance forms are written with the whole amount in Pesos and Centavos separately. Whole Peso amounts are written using the templates “<#> na piso” (for numbers ending in a consonant) and “<#>ng piso” (for numbers ending in a vowel). Similarly, Centavo amounts are written using the templates “<#> na sentimo” (for numbers ending in a consonant) and “<#>ng sentimo” (for numbers ending in a vowel). Peso amounts come first before the Centavos, and they are connected using the word “at” if both of them are nonzero. So, the amount 10.75 is written as “sampung piso at pitongpu’t limang sentimo”. Otherwise, if either the Pesos or Centavos portion is equal to zero, it is then omitted from the written amount. Finally, the amount always ends in the word “lamang” (“only”).

Using the information above, we can write any number up to 99999, which is the maximum amount that the remittance system can support and send through its network. The OCR (optical character recognition) module of the system has been created by somebody else, so the numerals are already provided. You now aim to convert the numeral amount in the remittance forms into words so that the automated system can cross-check it with the one written on the same form. You thought that this would be easier as converting the other way around requires some more involved string manipulation.

Input

The input starts with a number T on a single line denoting the number of remittance forms. T lines then follow, with each line denoting the numeral amount written on the form. Each line consists of two space-separated nonnegative integers p c denoting the whole Pesos and Centavos, respectively.

Output

The output consists of T lines, with each line denoting the corresponding form in the input. Each line should be of the format $Case\ \#t\colon\ W_{TL}$ where t is the serial of the test case starting from 1 and W_{TL} the amount written in Filipino. The amount should be written in all lowercase letters.

Constraints

$p, c \in Z^{+} \cup \{0\}$
$0 \leq p < 100000$
$0 \leq c < 100$

Both p and c can never be zero.

You can assume that all of the inputs are well-formed and are always provided within these constraints. You are not required to handle any errors.

Sample Input/Output

Sample Input 1:

```
5
50 0
12345 75
1001 10
0 99
74 12
```

Sample Output 1:

```
Case #1: limangpung piso lamang
Case #2: labingdalawang libo't tatlong daan at apatnapu't limang
```

```
piso at pitongpu’t limang sentimo lamang
Case #3: isang libo’t isang piso at sampung sentimo lamang
Case #4: siyamnapu’t siyam na sentimo lamang
Case #5: pitongpu’t apat na piso at labingdalawang sentimo lamang
```

Steps

1. Write your program in Rust. Make sure to accept input via standard input and print your output via standard output.
2. Submit a copy of the source code to the Week 01C submission bin. Make sure that you attach one (1) file in the bin containing the Rust source code with a .rs extension or .rs.txt extension (if UVLe doesn’t accept .rs files).