Graphviz Assignment

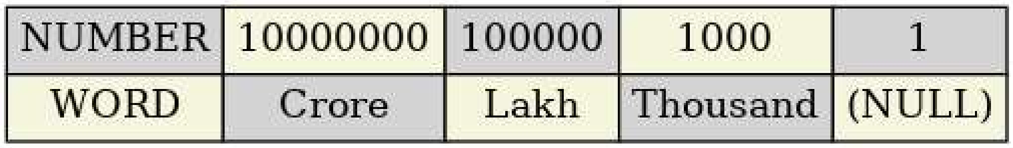
Fig2Words

Given a number convert it into words.

Algorithm

This problem can be solved for both western and Indian system denominations, but here we'll be considering the Indian System. The same procedure is applied to the western system as well.

Start
1. Input the number
2. Construct a dictionary for the denominations as shown below let's call it **DENOM_NAMES** :

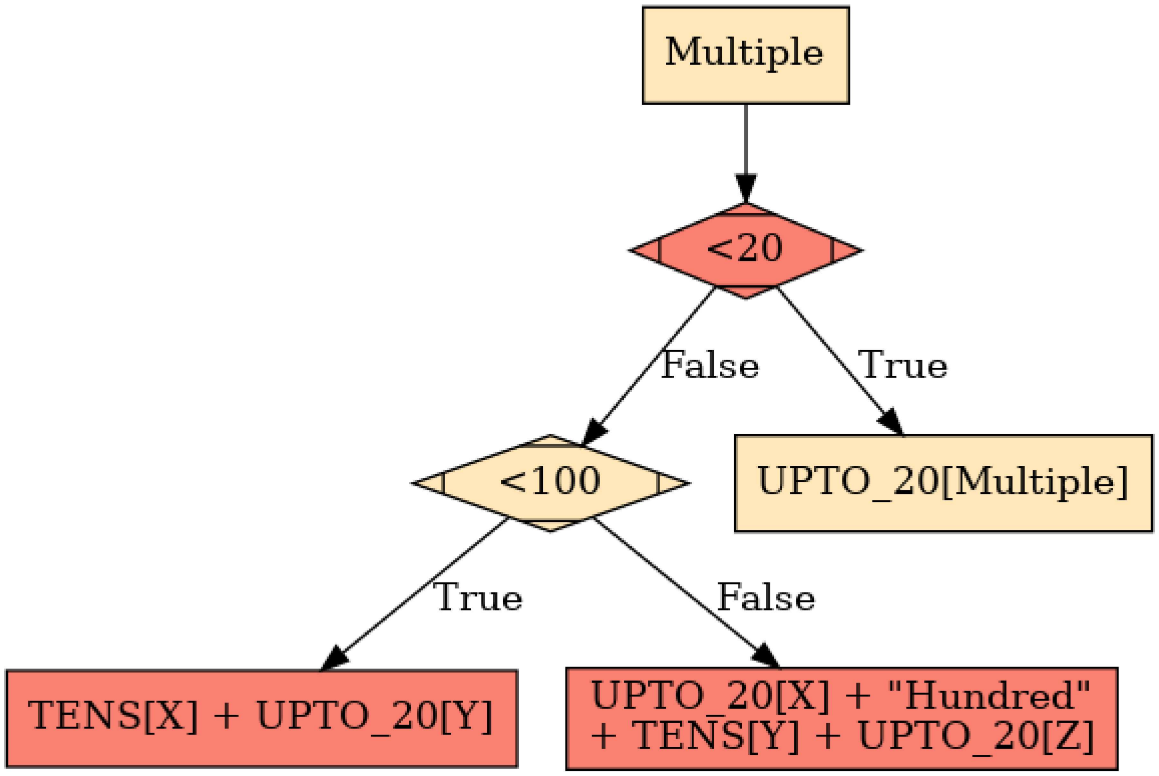


NUMBER	10000000	100000	1000	1
WORD	Crore	Lakh	Thousand	(NULL)

3. Split the given number into denomination- multiple pairs.
   For example :
      If the given number is 14,594 = [(1000,14), (1,594)]
      Here (1000, 14) is a denomination- multiple pair where 1000 is denomination and 14 is the muliple.
4. Covert the denominations in the list to words using the dictionary we constructed in step 2.
   Following the same example:
      After conversion, we get [(Thousand,14), (, 594)]
5. Now, covert the multiples into words. In order to do that follow the below steps:
   1. Create a dictionary, named **UPTO_20** for all numbers <=20 as we have done in **DENOM_NAMES** storing them in words.
   2. Create a second dictionary that would contain all multiple of 10 (Eg: ten, twenty, thirty, etc.) in a similar manner. Let's call it **TENS**.
   3. Now for each multiple in our list, there can be the following cases :
      a. If multiple is less than 20, return its value (i.e, words) from **UPTO_20** directly.
         In our example, the multiple 14 is < 20 so we can directly replace it with its word in the dictionary we made earlier. So 14 becomes ***"Fourteen"***.
      b. If multiple is less than 100, let **X** be the digit in 10's place and **Y** be the digit in 1's place then return the sum of **TENS[X]** and **UPTO_20[Y]**.

c. Let **X** be the digit in 100's place, **Y** be the digit in 10's place and **Z** be the digit in 100's place. Return **UPTO_20[X]** + ***"Hundred"*** + **TENS[Y]** + **UPTO_20[Z]** after adding ***"And"*** at the appropriate position.
Multiple 594 would become: ***"Five Hundred And Ninety Four"***

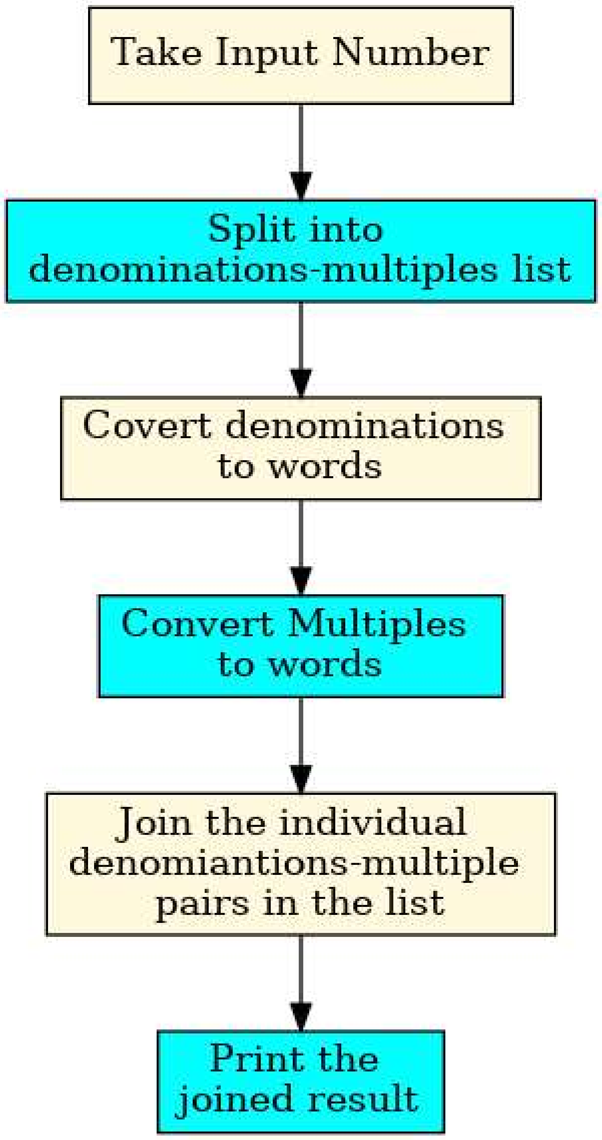
Replace the multiple with these returned words. After this step, our list would look like [("Fourteen", "Thousand"),("Five Hundred And Ninety Four", "")]
The above step can be understood with the below graph.



6. Now join the denomination in words and the multiple in words to get the final result. [("Fourteen", "Thousand"),("Five Hundred And Ninety Four", "")] = " Fourteen Thousand Five Hundred And Ninety Four".

Stop

The following flow chart lays down the general structure for the program :


Take Input Number
Split into denominations-multiples list
Covert denominations to words
Convert Multiples to words
Join the individual denomiantions-multiple pairs in the list
Print the joined result

Odometer

This odometer can only show readings from 1-9 and only the readings that are in ascending order are valid. The first valid 4 digits are 1234 and the last valid is 6789. 2467 is valid, while 2314 is not. The odometer rolls over to the first valid reading after the last.
The odometer can be of any size.
Write a set of functions that will allow us to carry out the functionalities of the odometer.

Functionalities of an Odometer

1. Get the next reading

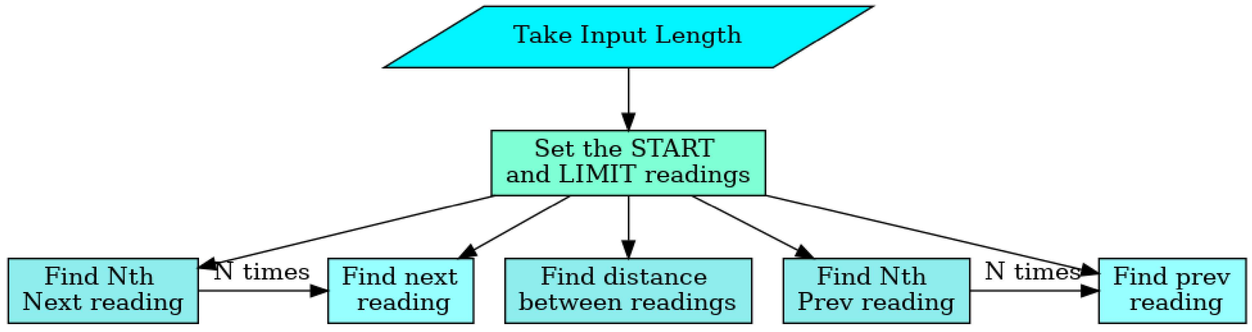
2. Get the previous reading
3. Get the distance between two readings
4. Get nth next reading
5. Get nth prev reading

Algorithm

Start
1. Input the length of the odometer lets call it len
2. Check if len is valid (i.e, >0) if not display an error message.
3. Set the **START** reading and **LIMIT** reading based on the length of the odometer.
    a. **START** reading will be the first len digits “123456789”
    b. **LIMIT** reading will be the last len digits of the same.
4. Set the **CURRENT** reading to the **START** reading.
5. The user can perform the above-mentioned functionalities.
6. ***Next reading***: Increment the value of the **CURRENT** reading until we get a valid reading. A valid reading is one that contains all digits in the reading in ascending order. Here if we arrive at the **LIMIT** while incrementing we need to set it back to **START**.
7. ***Prev reading***: Decrement the value of the **CURRENT** reading until we get a valid reading. if we arrive at the **START** while decrementing we need to set it back to **LIMIT**.
8. ***Next nth reading***: Perform the next reading operation N times for a given value of N
9. ***Prev nth reading***: Perform the prev reading operation N times for a given value of N.
10. ***Distance Between Two readings***: This can be calculated by incrementing one of the reading till we get the other reading while taking care that when we reach the LIMIT reading we need to go to START reading.

Stop

The flow chart below lays down the general structure of the program


Take Input Length
Set the START and LIMIT readings
Find Nth Next reading
N times
Find next reading
Find distance between readings
Find Nth Prev reading
N times
Find prev reading