CS 201R
Fall 2020
Program 5

For this program you'll write a program that uses a recursive function.

This program plays a simple token-taking game. The rules are simple:
1. You start the game with exactly 13 tokens.
2. On each turn, you may do one of two things:
    a. You may ask for exactly 25 more tokens; or,
    b. IF the number of tokens you have is an even number, you may give back exactly half of the tokens you have.

3. The object of the game is to reach exactly *K* tokens within *N* turns, where *K* and *N* are specified at the start of each game.

Think about this problem recursively; if the goal is to reach, say, 34 tokens within 10 turns, and I begin with 13 tokens and choose to take 25 more, then I have 38 tokens and 9 turns left. *Your program must be recursive to get full credit.*

Your program should ask the user for the goal number of tokens, and the number of turns allowed. It should then search and either report to the user that the total cannot be reached within the desired number of turns, or the exact sequence of moves for doing so, in order from the start state. There may be more than 1 path from the starting state to a winning state; you do not need to find them all, count them all, or find the shortest one, only find a valid path within the specified number of turns if one exists.

Think carefully about how to manage the problem recursively. What are the stopping conditions? (There's more than 1.) And if a solution is found, what should happen? (Hint: Each recursive call to the function will deal with one step of the solution. All it needs to know is that when it makes a recursive call, that call returned true, indicating a solution was found.) What should be recorded? And what if output is produced in a different order than it should be displayed? (Hint: A vector lets you add things in order; but the first thing in the vector may be the last step in what should be done.)

Note: The user can specify how deep the search can go. Don't get carried away with that; the number of possible paths of length N is proportional to 2^N. For $N > 20$ or so, this can start taking quite a bit of time. (79 takes more than 20 steps, but less than 30. Getting to 40 tokens takes more than 50 steps.) But don't tell the program to search to a depth of 100 unless you're prepared to wait a few days in some cases.

Also, program efficiency will be increased if you always begin with a step that tends to reduce the number of remaining tokens.

Submit your GitHub link, or delete the contents of the Debug directory and upload your zipfile, by Sunday night, November 15.