[Question 3 / C++ problem]

You are tasked with developing a program that extracts the area and name of shapes. A foundational structure and classes are already provided. Complete the code following the requirements below.

Requirements:

1. The `Shape` class serves as the base class for all shapes.

   - `calc_area()` should be a pure virtual function responsible for calculating the area of the shape.

   - `get_name()` should return the string "Shape".

2. The `Rectangle` class should inherit from the `Shape` class.

   - Member variables: `w` (width), `h` (height)

   - The area calculation function should compute the area using width and height.

   - The name-returning function should return "Rectangle".

   - The constructor should accept width and height as arguments.

3. The `Square` class should inherit from the `Rectangle` class.

   - Use the inherited function to calculate area.

   - The name-returning function should return "Square".

   - The constructor should accept only width as an argument.

4. The `Triangle` class should inherit from the `Shape` class.

   - Member variables: `a`, `b`, `c` (lengths of the three sides)

   - The area calculation should use Heron's formula:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$

where $s = \frac{a+b+c}{2}$ (semi-perimeter of the triangle).

- The name-returning function should return "Triangle".

- The constructor should accept the lengths of the three sides as arguments.

5. The `Right Triangle` class should inherit from the `Triangle` class.

   - The area calculation should use the lengths of the two shorter sides to compute the area of the right triangle.

   - The name-returning function should return "Right Triangle".

   - The constructor should accept the lengths of the three sides as arguments.

6. If the `Triangle` class receives side lengths that don't satisfy the triangle inequality, the area calculation function should return -1.

Run the given `main` function to verify if the code works correctly.

Submission:

Submit your completed code. The submitted code will be tested along with the provided `main` function.

The problem is now translated into English and includes Heron's formula for clarity.

[Question 2 / C problem]

Given a string `s`, return the minimum number of characters that need to be deleted to obtain the longest palindromic subsequence and the resulting subsequence itself.

Constraints:

- The length of `s` is between 1 and 10.

- `s` consists only of lowercase English letters.

Example 1:

Input -> s = "unique"

3

unu

Example 2:

Input -> s = "radar"

0

radar

Note:

- In the first example, by deleting three characters "iqe" from "unique", we can form the palindromic subsequence "unu".

- In the second example, "radar" is already a palindrome, so no deletions are necessary.

Approach:

1. Determine the length of the given string.

2. Initialize a 2D dynamic programming table.

3. Compute the length of the longest palindromic subsequence.

4. Determine the number of characters to be deleted as `length of s - length of the longest palindromic subsequence`.

5. Backtrack to retrieve the actual subsequence.

6. Print the results.

Requirements:

- Use `scanf` to get the input string.

- Use `printf` to print the results.

- The entire program should work correctly with the provided `main` function.

The problem now focuses on the number of deletions to achieve the longest palindromic subsequence and provides the desired subsequence.

[Question 1 / Python problem]

Based on the provided code, it seems the problem is related to finding "root vertices" in a graph. A "root vertex" is a vertex from which, when started, we can visit all other vertices in the graph. Here's a reconstructed problem description:

Problem: Finding Root Vertices in a Graph

Given a directed graph G, determine the root vertices. A root vertex in a directed graph is a vertex from which all other vertices are reachable.

The graph G is represented as an adjacency list, where each node has an ID (a string) and each key in the dictionary G has a list of neighboring nodes.

Function Signature:

def find_root_vertices(G: dict) -> list:

Input:

- A dictionary G representing the adjacency list of the graph.

Output:

- A list containing the IDs of all root vertices. Return an empty list if there are no root vertices.

Class Definition:

```
class GNode:
    def __init__(self, id, color="W", d=0, p=None):
        self.id = id # id is a string
        self.color = color # color (status) of node
```

```
        self.distance = d
        self.parent = p
    def __str__(self):
        return self.id
```

Example 1:

```python
A, B, C = GNode('A'), GNode('B'), GNode('C')
D, E, F = GNode('D'), GNode('E'), GNode('F')
G = dict()
G[A], G[B], G[C] = [C, D], [A,E], [B, D]
G[D], G[E], G[F] = [F], [F], []

find_root_vertices(G)   # Output: ['A', 'B', 'C']
```

Example 2:

```python
A, B, C = GNode('A'), GNode('B'), GNode('C')
D, E, F = GNode('D'), GNode('E'), GNode('F')
G = dict()
```

```
G[A], G[B], G[C] = [D], [E], [B, D]

G[D], G[E], G[F] = [F], [F], []


find_root_vertices(G)    # Output: []
```