Coding Challenge: Implement a Single-Function Discrete Low-Pass Filter

Problem

You are controlling a robot chassis from a microcontroller and need to smooth abrupt operator commands. Implement a one-pole first-order discrete low-pass filter that can be called once per control-loop tick and keeps its own internal state.

The function must support:

- Initialization/refresh when the caller passes $init = 1$
- Stable behavior for invalid parameters (pass-through)

Function to Implement

```
real low_pass_filter(real x, real alpha, int init)
```

Behavior

- **Update rule (EMA / RC low-pass):**

  $$\boldsymbol{y[n] = (1 - \alpha)y[n - 1] + \alpha x[n]}$$

  and $x[n]$ is a list of discrete-time input samples
- **Initialization / reseed: On the first call or when $\boldsymbol{init == 1}$:**
  - Store the provided alpha.
  - Seed internal output with the current input: $y \leftarrow x$.
  - Return the seeded output for that call.
- **Runtime updates:** Applying the EMA using the latest alpha on regular calls.
- **Alpha clamping:** Accept alpha in (0, 1]. If $\boldsymbol{\alpha \leq 0}$ or $\boldsymbol{\alpha > 1}$, treat as 1.0 (pass-through).

Constraints

- Keep state in static variables inside the function (not re-entrant / not thread-safe).
- Use double (typedef double real; provided).

Input / Output

- **Input each tick:** current sample x, smoothing factor alpha, and $init$ flag.
- **Output:** smoothed value y.

Qualitative Example

With $\alpha = 10$ and a step from 0 to 5 at $n = 10$:

- **Before the step:** output stays near 0 (after the first-sample seed).
- **After the step:** output rises smoothly toward 5 (exponential transient).