Given a function f(int x, int y).
it is supposed to calculate the power x^y.

$$POW(x,y) = \begin{cases} 1 & y = 0 \\ x^{y-1} * x & \text{if } y > 0 \\ \text{undefined} & \text{if } y < 0 \end{cases}$$

this is because f returns an int

Precondition:

```
int f(int x, int y)
  // @ requires y >= 0;
{
}
```

to enable precondition checking:

```
$ colin -d filename.c0
```

Safety of a function call :-

if the user calls $f(x, y)$ with $y < 0$ then it's an unsafe call.

if an unsafe call is made by the function caller,

① the program may abort

② output wrong result.

— On a function call, when all preconditions on a function input are satisfied then it's a safe call.

Output of a function :-

```
int f (int x, int y)
precondition →// @ requires y >=0;
post condition→// @ ensures \result == POW(x,y);
{
    body
}
```

Correctness of a function :-

The program has a bug if all the preconditions are satisfied but the post condition is not satisfied.

When program has a bug we say it is incorrect.

Development code :-

- bugs are possible.
- use com -d command.

Production code :-

- code is working
- compile it without any -d.

A program runs slower with -d flag.

Caller doesn't care about the implementation (body of the function)

Caller should know :-

① header of the function :-

int f (int x, int y)

② precondition :

// requires y >= 0;

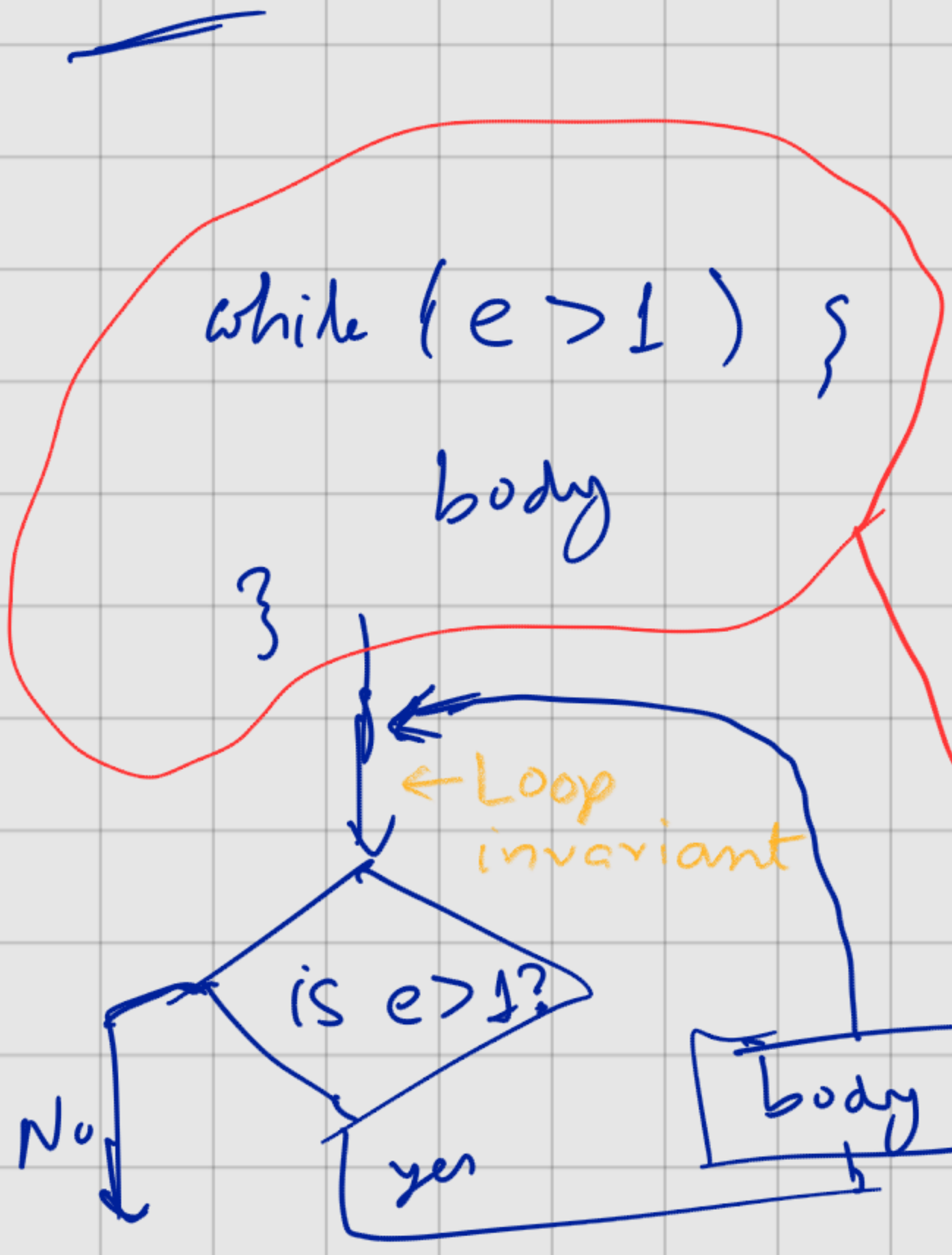
③ Postcondition :-

// ensures \result == pow(x,y);

The order of Writing good code :-

first ① . write the contracts .

second ② body of function.




```c
int POW(int x, int y)
//@requires y >= 0;
{
        if (y == 0) {
                return 1;
        }
        else {
                return POW(x, y-1) * x;
        }
}

int f(int x, int y)
//@requires y >= 0;
//@ensures \result == POW(x,y);
{
        int b = x;
        int e = y;
        int r = 1;

        while (e > 1) {
                if (e % 2 == 1) {
                        r = b * r;
                }
        b = b * b;
        e = e / 2;
        }
        return r * b;
}
```

loop guard

LI

```
while (e > 1)
//loop_invariant condition;
{
  body
}
```

Loop invariants :-

Any quantity that remains constant at each iteration of the loop.

Finding loop invariant by tracing the code :

```
int POW(int x, int y)
//@requires y >= 0;
{
        if (y == 0) {
                return 1;
        }
        else {
                return POW(x, y-1) * x;
        }
}

int f(int x, int y)
//@requires y >= 0;
//@ensures \result == POW(x,y);
{
        int b = x;
        int e = y;
        int r = 1;

        while (e > 1) {
                if (e % 2 == 1) {
                        r = b * r;
                }
        b = b * b;
        e = e / 2;
        }
        return r * b;
}
```

f(2, 8)

x = 2 ; y = 8 ;

b = 2 ; e = 8

r = 1

b	e	r	b^e
2	8	1	28
4	4	1	4^4
16	2	1	16^2
256	1	1	256

b^e

Se appears to be a loop invariant

f(2,7)

b	e	r	
2	7	1	128
4	3	2	64
16	1	8	16

```
int POW(int x, int y)
//@requires y >= 0;
{
        if (y == 0) {
                return 1;
        }
        else {
                return POW(x, y-1) * x;
        }
}

int f(int x, int y)
//@requires y >= 0;
//@ensures \result == POW(x,y);
{
        int b = x;
        int e = y;
        int r = 1;

        while (e > 1) {
                if (e % 2 == 1) {
                        r = b * r;
                }
        b = b * b;
        e = e / 2;
        }
        return r * b;
}
```

Candidate L.I. $L^{e.r.}$

Put L^{eq} as a loop invariant.

//loop_invariant POW(b,e)*r == POW(x,y);

```
while ( e > 1 )
//loop_invariant  POW(b,e)*r == POW(x,y);
{
  body
}
```

call 2 call 1

call 1 to POW is safe because of precondition on f.

Is call 2 to POW safe ??

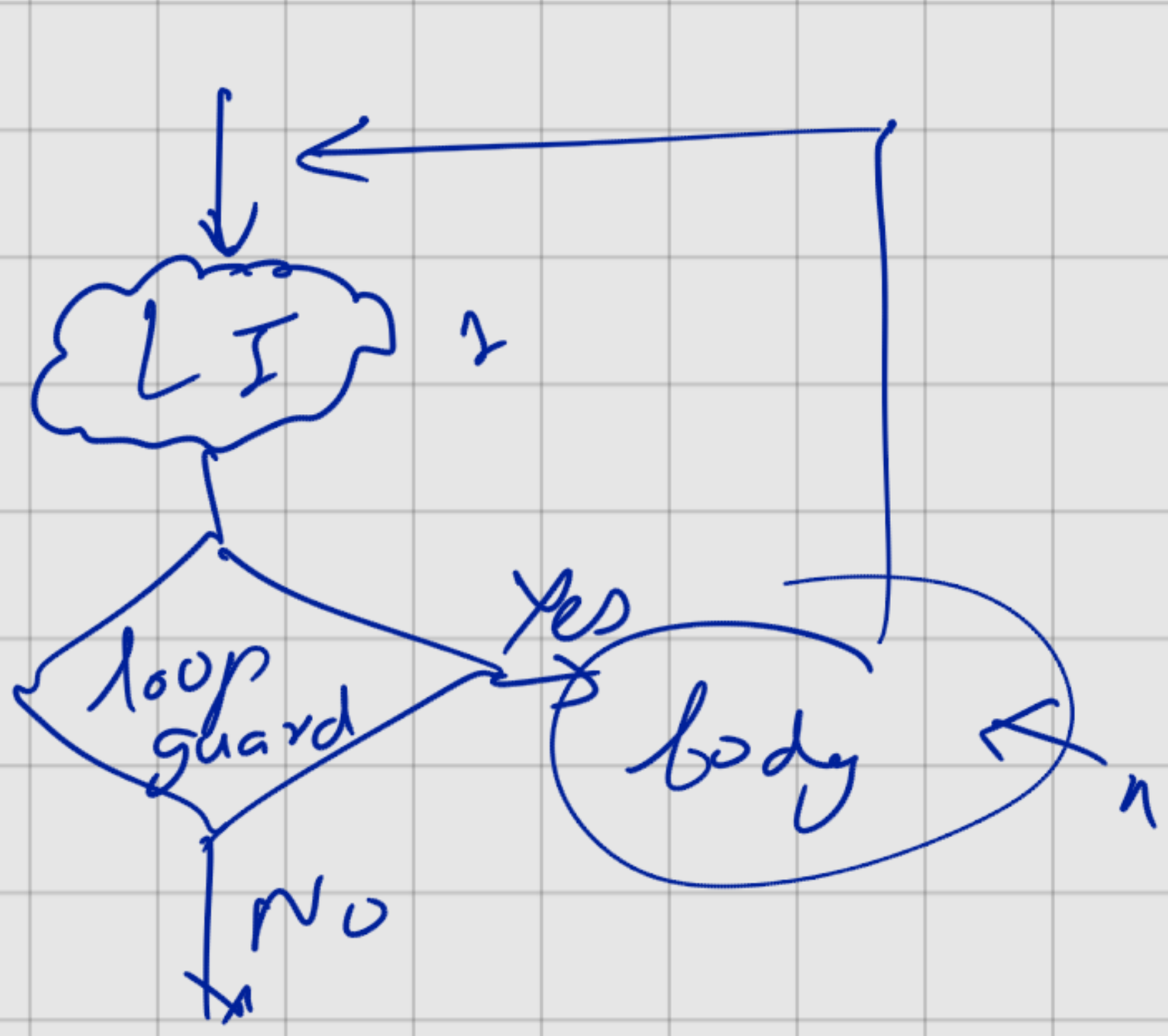
↓

for call 2 to be safe we would need $e \geq 0$, Lets put that as another loop invariant and try to prove it.

```
int POW(int x, int y)
//@requires y >= 0;
{
        if (y == 0) {
                return 1;
        }
        else {
                return POW(x, y-1) * x;
        }
}

int f(int x, int y)
//@requires y >= 0;
//@ensures \result == POW(x,y);
{
        int b = x;
        int e = y;
        int r = 1;

        while (e > 1) {
                if (e % 2 == 1) {
                        r = b * r;
                }
        b = b * b;
        e = e / 2;
        }
        return r * b;
}
```

```
while ( e > 1 )
    //loop_invariant   e >= 0 ;
    //loop_invariant pow(b,e)*x == pow(x,y);
{

}
```

Note that
if loop body is executed n times.
then LI & (loop guard
are checked
n+1 times!



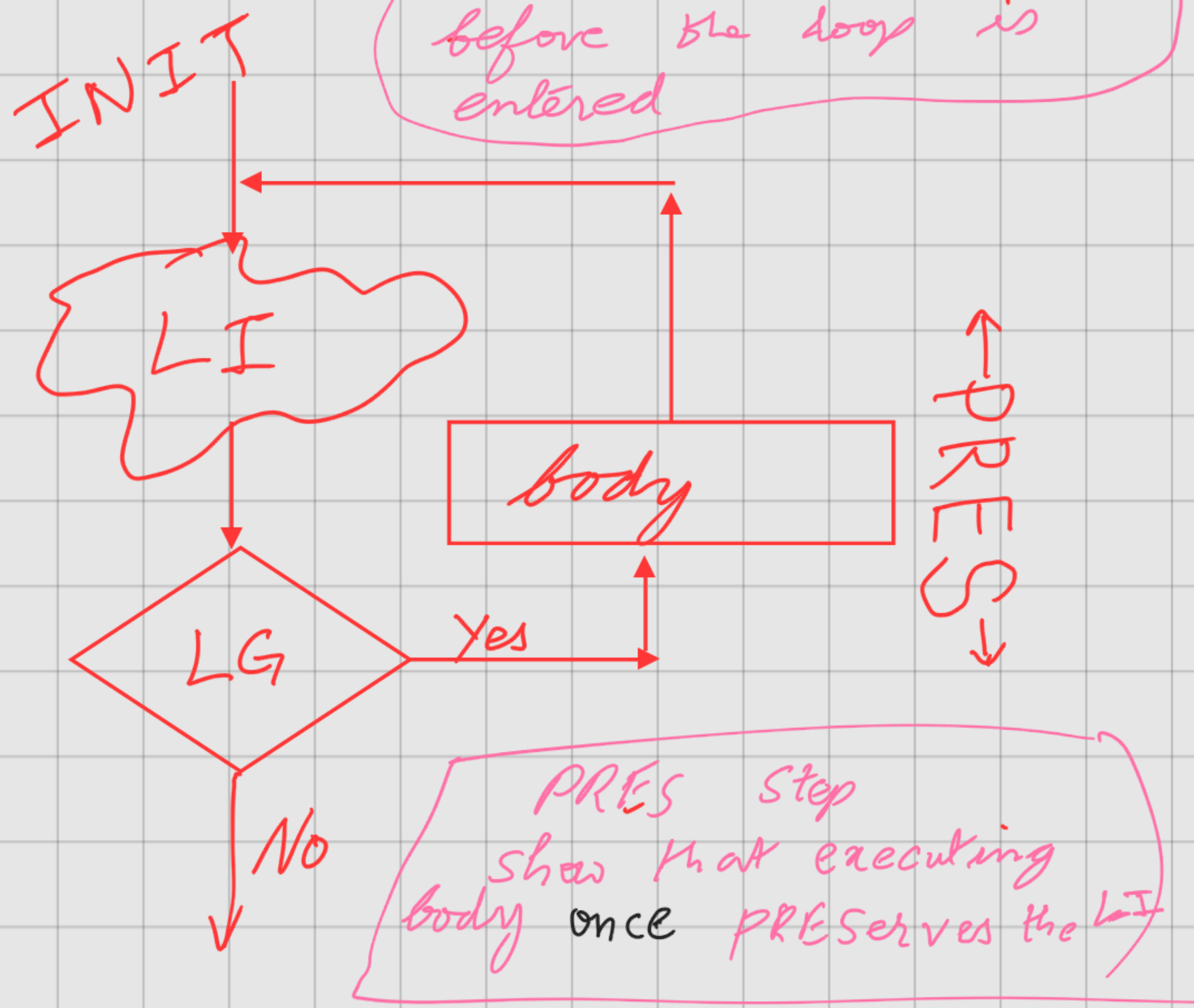
When we exit the loop

① LI is true

② loop guard is false.

$\neg LG$

Proving a LI.



For INIT we need to show that the following lines preserve L.I.

```
int b = x;
int e = y;
int r = 1;
```

```
int POW(int x, int y)
//@requires y >= 0;
{
        if (y == 0) {
                return 1;
        }
        else {
                return POW(x, y-1) * x;
        }
}

int f(int x, int y)
//@requires y >= 0;
//@ensures \result == POW(x,y);
{
        int b = x;
        int e = y;
        int r = 1;

        while (e > 1) {
                if (e % 2 == 1) {
                        r = b * r;
                }
        b = b * b;
        e = e / 2;
        }
        return r * b;
}
```

For PRES step we need to show that the loop body

```
if (e%2 == 1){
        r = b*r;
}
b = b*b
e = e/2;
```

preserves the L.I.

LI 1 :

$e \geq 0$

INIT

$y \geqslant 0$ by L1

$e = y$ by L2

$\Rightarrow e \geqslant 0$

PRES step

$e \geqslant 0$

assume that at the end of the loop e gets changed to e'.

T.P. if $e \geqslant 0$
then $e' \geqslant 0$

if $e \geqslant 0$ & $e' := e/2$ } L3
then $e' \geqslant 0$

```
int POW(int x, int y)
//@requires y >= 0;
{
        if (y == 0) {
                return 1;
        }
        else {
                return POW(x, y-1) * x;
        }
}

int f(int x, int y)
//@requires y >= 0;
//@ensures \result == POW(x,y);
{
        int b = x;
        int e = y;
        int r = 1;

        while (e > 1) {
                if (e % 2 == 1) {
                        r = b * r;
                }
        b = b * b;
        e = e / 2;
        }
        return r * b;
}
```

L2

L2

L3

LI 2 :

Le γ = xy .

INIT step

b = x L1

e = y L2

r = 1 L3

Le γ = xy

```
int POW(int x, int y)
//@requires y >= 0;
{
        if (y == 0) {
                return 1;
        }
        else {
                return POW(x, y-1) * x;
        }
}

int f(int x, int y)
//@requires y >= 0;
//@ensures \result == POW(x,y);
{
        int b = x;          — L1
        int e = y;          — L2
        int r = 1;          — L3

        while (e > 1) {
                if (e % 2 == 1) {
                        r = b * r;
                }
                b = b * b;  — L5  |
                e = e / 2;  — L4  |
        }                   — L6
        return r * b;
}
```

PRES step :-

at the end of the loop the variable b, e, r get changed to b', e', r' .

Divide the proof into two smaller (easier) parts

① when e is odd

② when e is even

① if e is odd :-

then $e = 2n+1$ for some int n.

by LC4 : $\gamma' = L\gamma$

by LS : $b' = L^2$

by LC6 : $e' = \frac{e}{2} = \frac{2n+1}{2} = n$

$b' \cdot e' \gamma' = L^e \gamma = x^y$

$\downarrow$

$(L^2)^n L\gamma = L^{2n+1}\gamma$

$L^e \gamma = L^{2n+1}\gamma$

as $e = 2n+1$

Exercise: Show that when e is even then LI. holds.

Show that INIT & PRES steps hold.

Question to think about:

Why is it enough to prove the INIT and PRES steps to prove a loop invariant?