Contracts

So far

- C0 function mystery
- Function contracts
  - @requires (pre-condition)
  - @ensures (post-condition)
  - @loop_invariant

Today

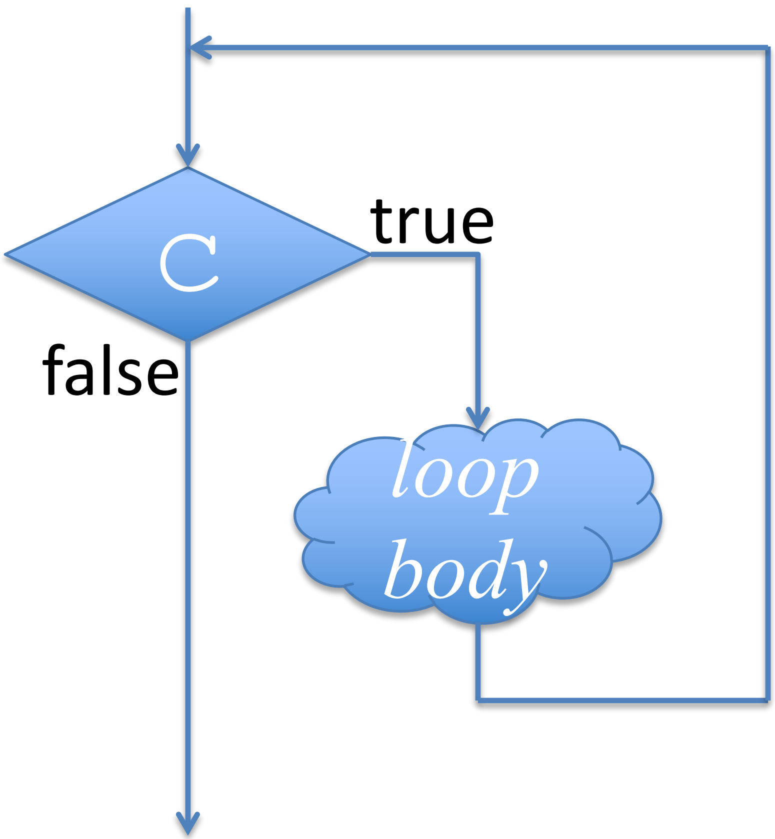
- Correctness: showing that function meets post- conditions when inputs meet pre- conditions
- Safe function call: showing preconditions are met
- Loop invariants: abstracting the workings of a loop
- Example proof of correctness using logical reasoning

```
int f (int x, int y)
//@requires y >= 0;
//@ensures POW(x, y) == \result;
{
   int b = x; /* Line 11 */
   int e = y; /* Line 12 */
   int r = 1; /* Line 13 */
   while (e > 1) /* Line 14 */
   //@loop_invariant e >= 0; /* Line 23 */
   //@loop_invariant POW(b, e) * r == POW(x, y);
   {
        if (e % 2 == 1)   { /* Line 18 */
           r = b * r;   /* Line 19 */
        }
        b = b * b;   /* Line 21 */
        e = e / 2;   /* Line 22 */
   }
    return r * b;
}
```

Using contracts

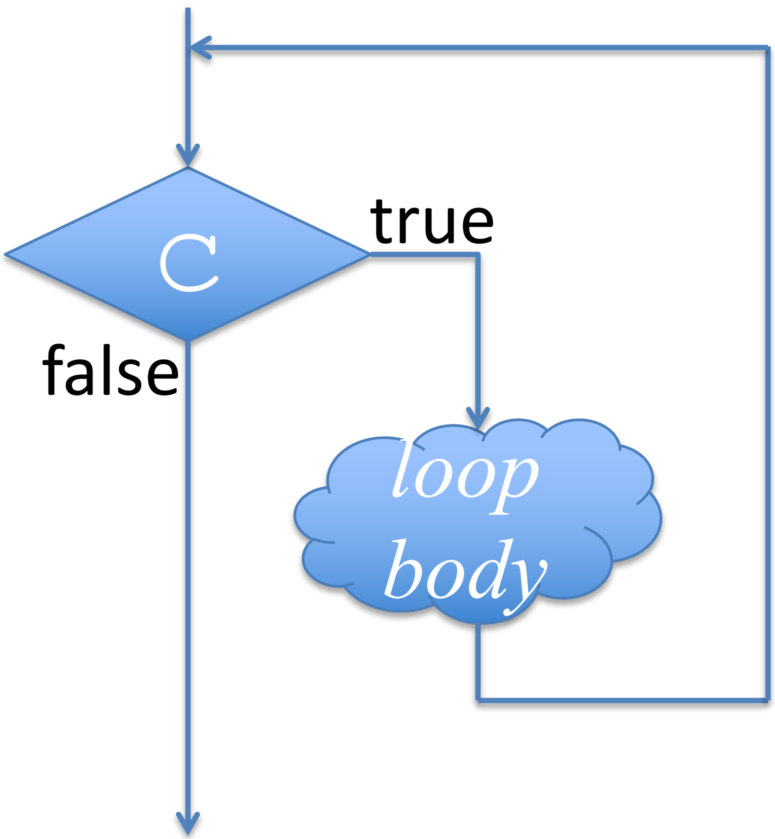
PROVING THE FUNCTION CORRECT

```
while (C) {
        loop body
}
```




```
int f (int x, int 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;
}
```

C stands for loop condition (guard)



C stands for loop condition (guard)

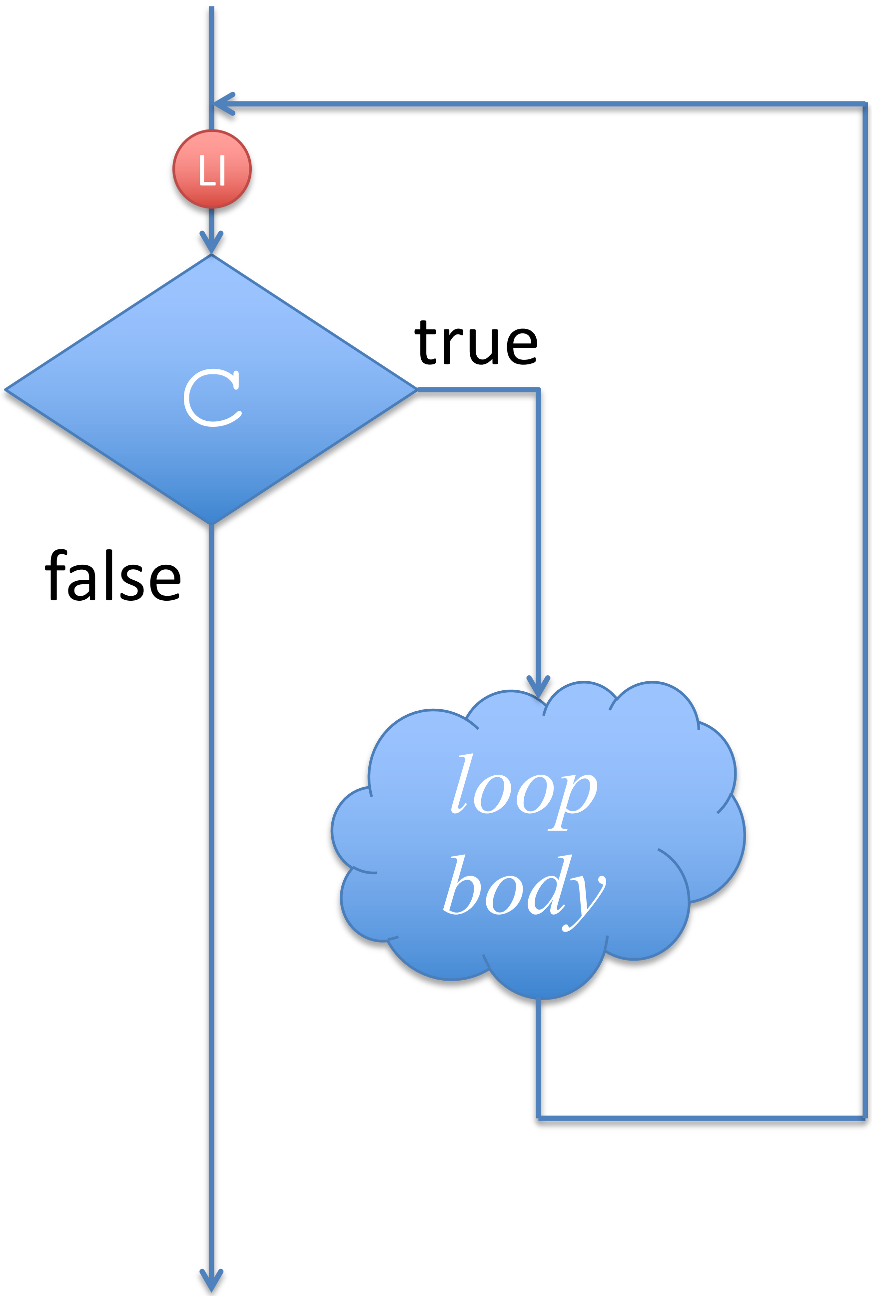
```
while (C)
//@loop_invariant LI;

{
        loop body
}
```

```
int f (int x, int y)
{
    int b = x;
    int e = y;
    int r = 1;
    while (e > 1)
    //@loop_invariant e >= 0;
    //@loop_invariant …
    {
        if (e % 2 == 1)    {
                r = b * r;
        }
          b = b * b;
          e = e / 2;
    }
     return r * b;
}
```

Loop Invariant

A boolean condition that is checked *immediately before every evaluation of the loop guard*.

LI




```
while (C)
//@loop_invariant LI;
{
    loop body
}
```

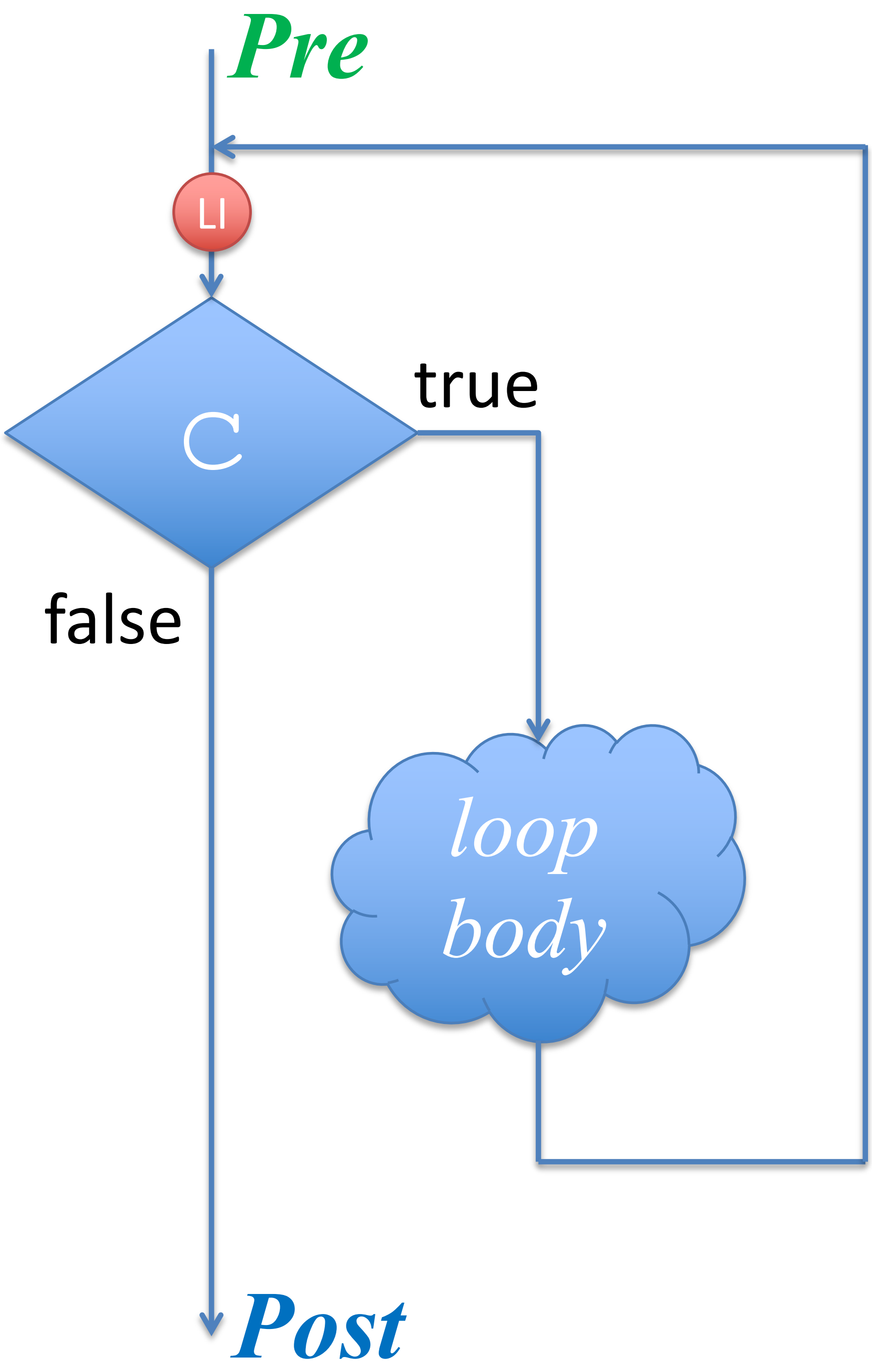
Loop Invariant

A boolean condition that is checked *immediately before every evaluation of the loop guard*.

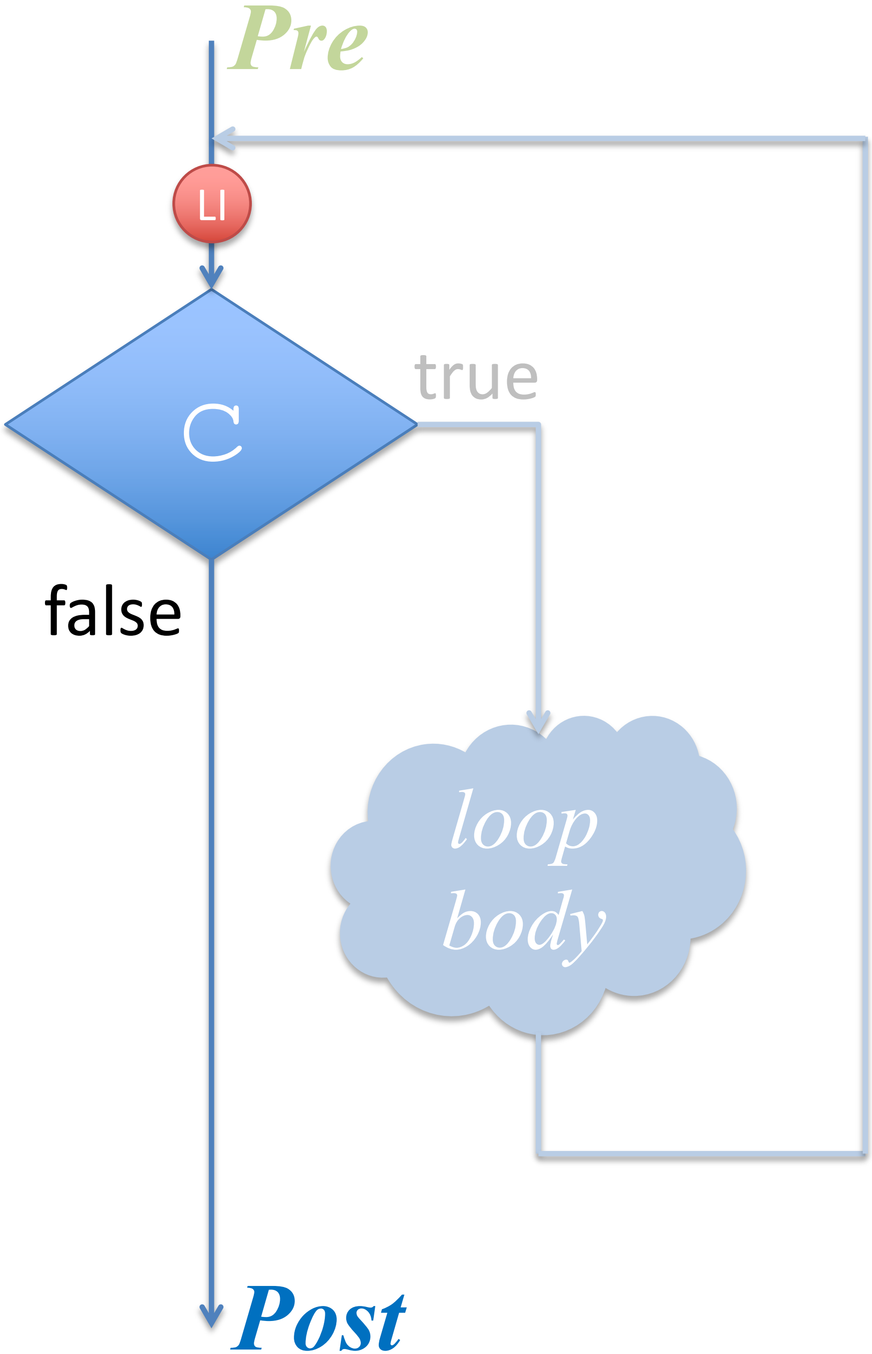
- It is true even if the loop runs 0 times (i.e., is skipped)
- It is true immediately before each evaluation of the loop guard, including the last evaluation if the loop terminates
- It is true immediately after the loop terminates, if the loop terminates

Proving the Correctness
of a function with one loop

Correctness: if preconditions hold,
then postconditions must hold



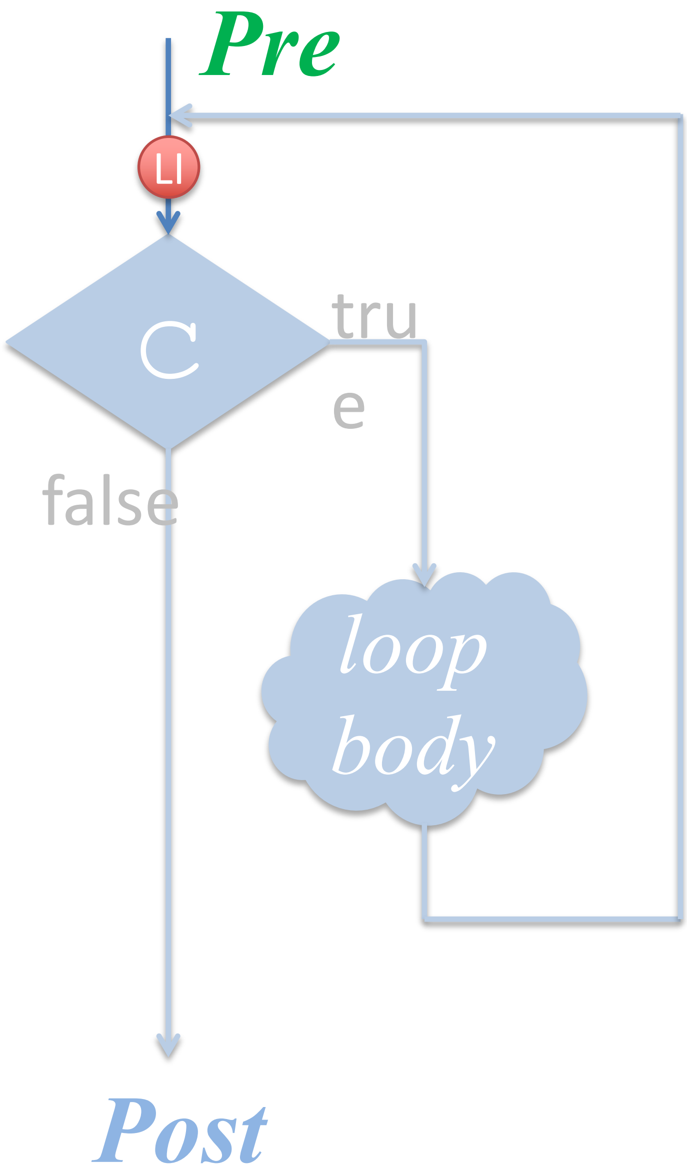

```
//@requires Pre;
//@ensures Post;

  ...

while (C)
//@loop_invariant LI;
{
    loop body
}
```



EXIT

If loop invariant is valid, show that:
the logical conjunction of the loop invariant ***LI*** and the negation of the loop guard ***C*** implies the desired postcondition ***Post***.

$$LI \wedge \sim C \rightarrow Post$$

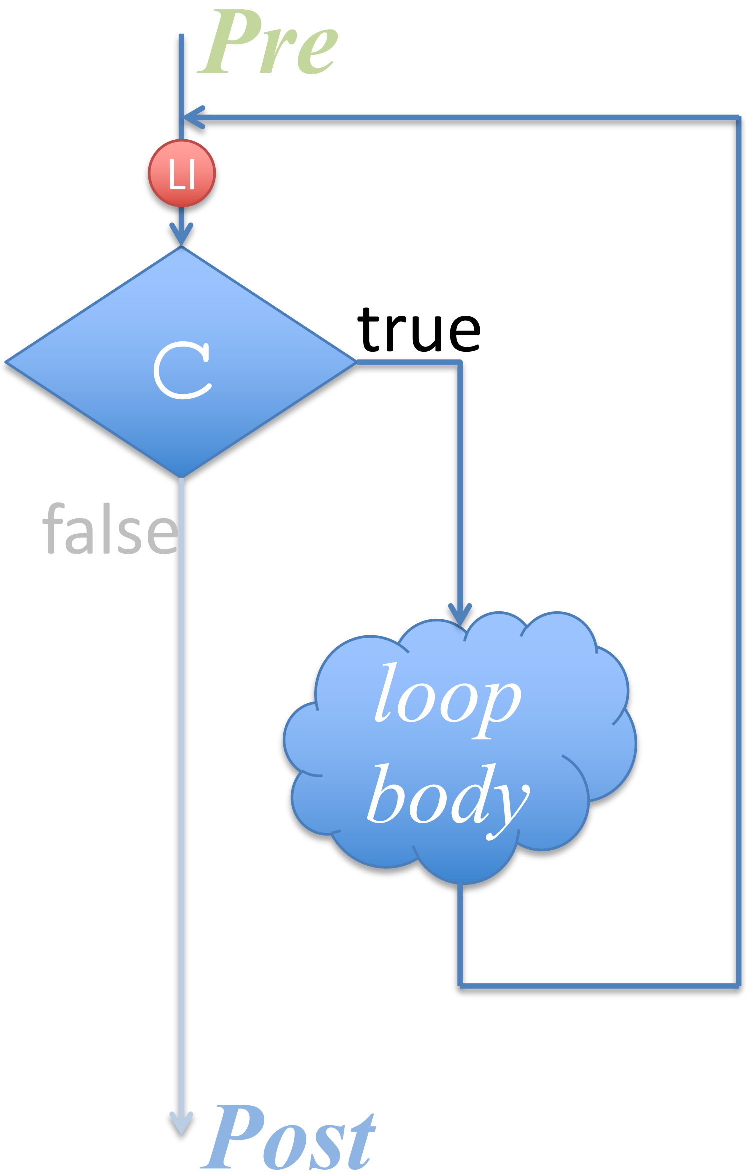


Showing ***LI*** valid – 1

INIT

Show that the loop invariant ***LI*** is true immediately before the first evaluation of the loop guard ***C***.

```
int f (int x, int y)
//@requires y >= 0;
//@ensures POW(x, y) == \result;
{
   int b = x; /* Line 11 */
   int e = y; /* Line 12 */
   int r = 1; /* Line 13 */
   while (e > 1) /* Line 14 */
   //@loop_invariant e >= 0; /* Line 15 */
   //@loop_invariant POW(b, e) * r == POW(x, y);
   {
        if (e % 2 == 1)   { /* Line 18 */
           r = b * r;   /* Line 19 */
        }
        b = b * b;   /* Line 21 */
        e = e / 2;   /* Line 22 */
   }
    return r * b;
}
```



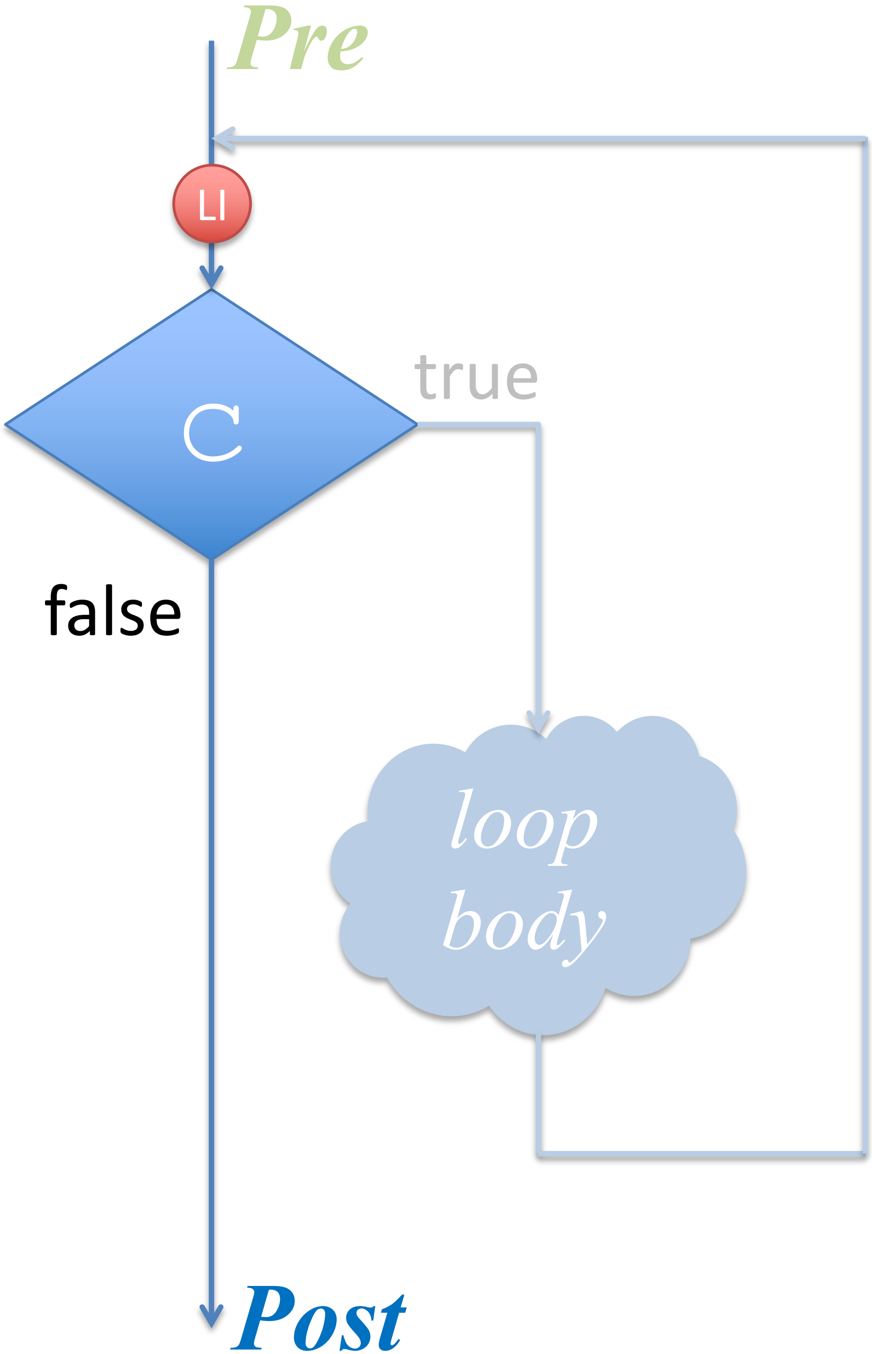
Showing ***LI*** valid – 2

PRESERVATION

Show that:

if the loop invariant ***LI*** is true immediately before the evaluation of the loop guard ***C***, *then* ***LI*** is true immediately before the next evaluation of the loop guard ***C***.

```
int f (int x, int y)
//@requires y >= 0;
//@ensures POW(x, y) == \result;
{
   int b = x; /* Line 11 */
   int e = y; /* Line 12 */
   int r = 1; /* Line 13 */
   while (e > 1) /* Line 14 */
   //@loop_invariant e >= 0; /* Line 15 */
   //@loop_invariant POW(b, e) * r == POW(x, y);
   {
        if (e % 2 == 1)   { /* Line 18 */
           r = b * r;    /* Line 19 */
        }
        b = b * b;    /* Line 21 */
        e = e / 2;    /* Line 22 */
   }
    return r * b;
}
```



EXIT

If loop invariant is valid, show that:
the logical conjunction of the loop invariant ***LI*** and the negation of the loop guard ***C*** implies the desired postcondition ***Post***.

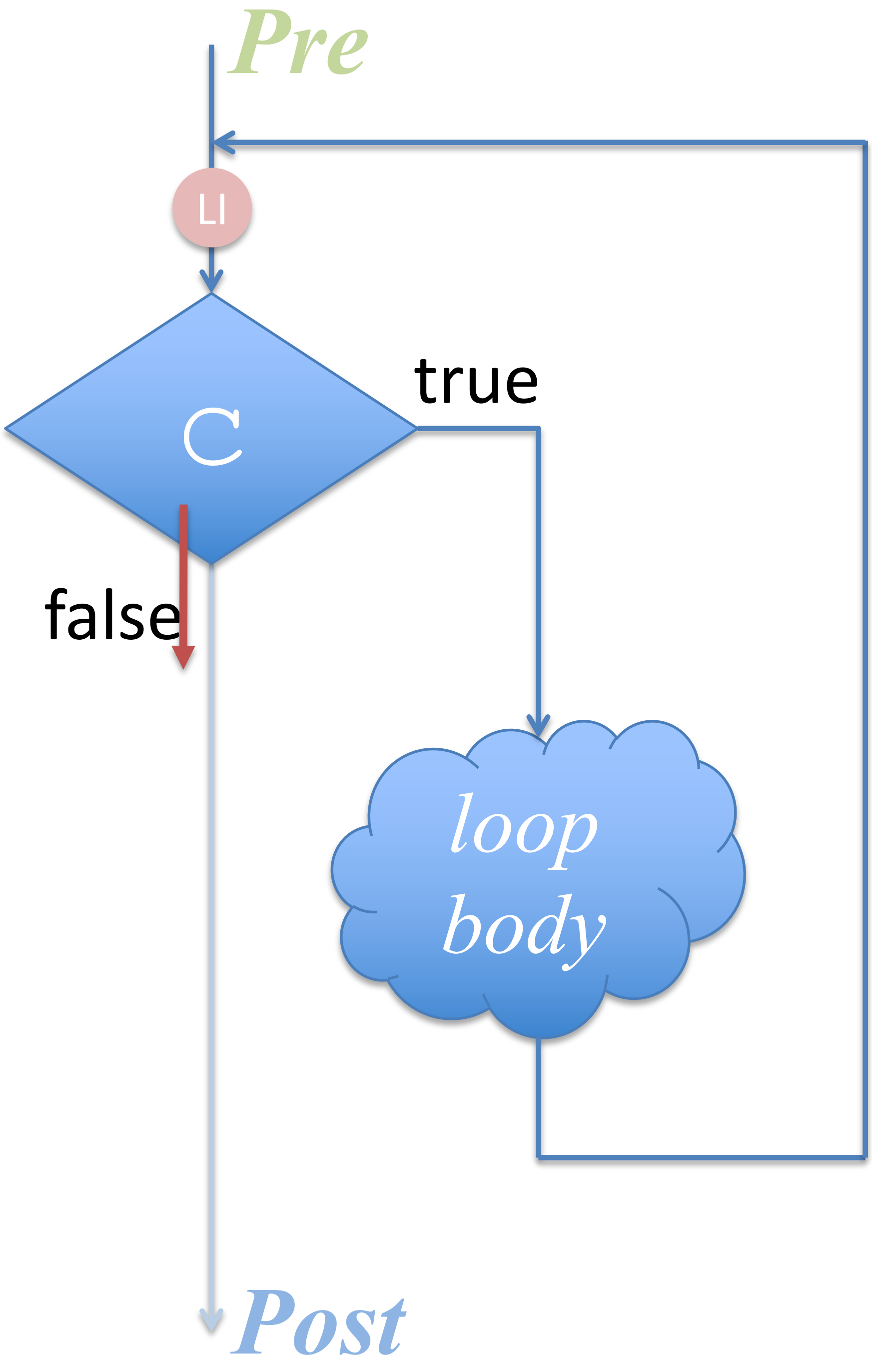
$$LI \wedge \sim C \rightarrow Post$$

Bug fixed

```
int f (int x, int y)
//@requires y >= 0;
//@ensures POW(x, y) == \result;
{
    int b = x; /* Line 11 */
    int e = y; /* Line 12 */
    int r = 1; /* Line 13 */
    while (e > 0) /* Line 14 */
    //@loop_invariant e >= 0; /* Line 15 */
    //@loop_invariant POW(b, e) * r == POW(x, y);
    {
         if (e % 2 == 1)   { /* Line 18 */
            r = b * r;   /* Line 19 */
         }
         b = b * b;   /* Line 21 */
         e = e / 2;   /* Line 22 */
    }
     return r;
}
```

With a fact asserted

```
int f (int x, int y)
//@requires y >= 0;
//@ensures POW(x, y) == \result;
{
   int b = x; /* Line 11 */
   int e = y; /* Line 12 */
   int r = 1; /* Line 13 */
   while (e > 0) /* Line 14 */
   //@loop_invariant e >= 0; /* Line 15 */
   //@loop_invariant POW(b, e) * r == POW(x, y);
   {
        if (e % 2 == 1)   { /* Line 17*/
            r = b * r;    /* Line 18 */
        }
        b = b * b;    /* Line 21 */
        e = e / 2;    /* Line 22 */
    }
    //@assert e == 0;
    return r;
}
```



TERMINATION
Show that the loop will always terminate (i.e., that ***C*** must eventually be false)

Correctness of a function with one loop

- Show that ***LI*** is valid
  - **INIT**: ***LI*** holds initially
  - **PRES**: ***LI*** is preserved by an arbitrary iteration

- **EXIT**: $\boldsymbol{LI} \wedge \sim \boldsymbol{C} \rightarrow \boldsymbol{Post}$

- **TERM**: loop terminates