```
method ComputePos(num: int, den: int) returns (n: int)
    requires num > 0 && den > 0
    ensures n > 0 && num == fusc(n) && den == fusc(n + 1)
{
    {true}
    {1 == 1}
    {forall x :: 1 == 1}
    var x := 1;
    {x == 1}
    {x == 1 && true}
    {forall y :: x == 1 && 1 == 1}
    var y := 1;
    {x == 1 && y == 1}
    {forall s :: x == 1 && y == 1}
    var s := 0;
    {x == 1 && y == 1}
    {x == 1 && y == fusc(1)}
    {x == 1 && y == fusc(2 * 1)}
    {true && x == 1 && y == fusc(2)}
    {1 > 0 && x == fusc(1) && y == fusc(1 + 1)}
    n := 1;
    {n > 0 && x == fusc(n) && y == fusc(n + 1)}
    while (num != x && den != y)
        invariant n > 0 && x == fusc(n) && y == fusc(n + 1)
        decreases -n
    {
        {n > 0 && x == fusc(n) && y == fusc(n + 1)}        --  strengthen with
invariants
        {n > 0}
        {(n + 1) >= 0}
        n := n + 1;
        {n >= 0}
        {n >= 0 && true}
        {n >= 0 fusc(n) == fusc(n)}
        {n >= 0 && forall b' :: b' == fusc(n) == fusc(n)}
        x := ComputeFusc(n);
        {n >= 0 && x == fusc(n)}
        {n >= 0 && x == fusc(n) && true}
        {n >= 0 && x == fusc(n) && fusc(n + 1) == fusc(n + 1)}
        s := n + 1;
        {n >= 0 && x == fusc(n) && fusc(s) == fusc(n + 1)}
        {n >= 0 && x == fusc(n) && forall b' :: b' == fusc(s) ==> b' == fusc(n +
1)}
        y := ComputeFusc(s);
        {n > 0 && x == fusc(n) && y == fusc(n + 1)}
    }
}
```