```
method ComputeFusc(N: int) returns (b: int)
    requires N >= 0
    ensures b == fusc(N)
{
    {true}

    {fusc(N) == fusc(N)}

    {fusc(N) == fusc(N) + 0 * fusc(N + 1)}

    b := 0;

    {fusc(N) == fusc(N) + b * fusc(N + 1)}

    {forall n :: fusc(N) == fusc(N) + b * fusc(N + 1)}

    var n := N;

    {fusc(N) == fusc(n) + b * fusc(n + 1)}

    {forall a :: 1 * fusc(N) == fusc(n) + b * fusc(n + 1)}

    var a := 1

    {fusc(N) == a * fusc(n) + b * fusc(n + 1)}

    while (n != 0)
        invariant fusc(N) == a * fusc(n) + b * fusc(n + 1)
        decreases n
    {
        {fusc(N) == a * fusc(n) + b * fusc(n + 1)} // strengthen (A || A == B -> A
as this reduces sample space of A)

        {fusc(N) == a * fusc(n) + b * fusc(n + 1) || a * fusc(n) + b * fusc(n + 1)
== a * fusc(n) + b * fusc((n + 1) / 2)}

        {(fusc(N) == a * fusc(n) + b * fusc(n + 1) && (n % 2 == 0 || n % 2 == 1))
||
                                    (fusc(N) == a * fusc(n) + b * fusc(n + 1) &&
fusc(N) == a * fusc(n) + b * fusc((n + 1) / 2)}

        {false || (n % 2 == 1 && fusc(N) == a * fusc(n) + b * fusc(n + 1) ||
(fusc(N) == a * fusc(n) + b * fusc(n + 1) && n % 2 == 0) ||
                                                    (fusc(N) == a * fusc(n) + b
* fusc(n + 1) && fusc(N) == a * fusc(n) + b * fusc((n + 1) / 2)}

        {(n % 2 == 1 && n % 2 == 0) || (n % 2 == 1 && fusc(N) == a * fusc(n) + b *
fusc(n + 1) / 2) ||
                            (fusc(N) == a * fusc(n) + b * fusc(n + 1) && n % 2 ==
0) || (fusc(N) == a * fusc(n) + b * fusc(n + 1) && fusc(N) == a * fusc(n) + b *
```

```
fusc(n + 1) / 2)}

        {(n % 2 == 1 || fusc(N) == a * fusc(n) + b * fusc(n + 1)) && (n % 2 == 0 ||
fusc(N) == a * fusc(n) + b * fusc(n + 1) / 2)}

        {n % 2 == 0 ==> fusc(N) == a * fusc(n) + b * fusc(n + 1) && n % 2 == 1 ==>
fusc(N) == a * fusc(n) + b * fusc(n + 1) / 2)}

        if (n % 2 == 0) {

            {fusc(N) == a * fusc(n) + b * fusc(n + 1)}          -- as an input of a
number (n) with a multiple of 2 is equal to an input of itself (n) {rule iii}

            {fusc(N) == a * fusc(n / 2) + b * fusc(n + 1)}

            {fusc(N) == a * fusc(n / 2) + b * fusc(2 * (n / 2) + 1)}

            {fusc(N) == a * fusc(n / 2) + b * fusc(n / 2) + b * fusc((n / 2) + 1)}

            {fusc(N) == (a + b) * fusc(n / 2) + b * fusc((n / 2) + 1)}

            a := a + b;

            {fusc(N) == a * fusc(n / 2) + b * fusc((n / 2) + 1)}

            n := n / 2;

            {fusc(N) == a * fusc(n) + b * fusc(n + 1)}

        } else {

            {fusc(N) == a * fusc(n) + b * fusc((n + 1) / 2)}

            {fusc(N) == a * (fusc(2 * ((n - 1) / 2) + 1)) + b * fusc((n + 1) / 2)}

            {fusc(N) == a * (fusc((n - 1) / 2) + fusc(((n - 1) / 2) + 1)) + b *
fusc((n + 1) / 2)}

            {fusc(N) == a * fusc((n - 1) / 2) + a * fusc(((n - 1) / 2) + 1) + b *
fusc((n + 1) / 2)}

            {fusc(N) == a * fusc((n - 1) / 2) + a * (fusc(((n - 1) / 2) + 1) + b *
fusc(((n - 1) / 2) + 1)}

            {fusc(N) == a * fusc((n - 1) / 2) + (b + a) * fusc(((n - 1) / 2) + 1)}

            b := b + a;

            {fusc(N) == a * fusc((n - 1) / 2) + b * fusc(((n - 1) / 2) + 1)}
```

```
            {fusc(N) == a * fusc((n - 1) / 2) + b * fusc(((n - 1) / 2) + 1)}
            n := (n - 1) / 2;
            {fusc(N) == a * fusc(n) + b * fusc(n + 1)}
        }
        {fusc(N) == a * fusc(n) + b * fusc(n + 1)}
    }
    {fusc(N) == a * fusc(n) + b * fusc(n + 1) && n == 0}
}
```