```
// Author of question: Snorri Agnarsson
// Permalink of question: https://rise4fun.com/Dafny/wuXa

// Author of solution:    Alexander Guðmundsson
// Permalink of solution: https://rise4fun.com/Dafny/9slQL

// Use the command
//   dafny SumOdds-skeleton.dfy
// or
//   compile compile SumOdds-skeleton.dfy
// to compile the file.
// Or use the web page rise4fun.com/dafny.

// When you have solved the problem put
// the solution on the Dafny web page,
// generate a permalink and put it in
// this file.

// Compute 1+3+5+...+(2*n-1),
// i.e. the sum of the first n odd numbers.
function SumOdds( n: int ): int
    decreases n;
    requires n >= 0;
{
    if n == 0 then
        0
    else
        SumOdds(n-1) + 2*n-1
}
```

```
// We want to prove that
// 1+3+5+...+(2*n-1) == n^2

lemma ProveSumOdds( n: int )
    // Put requires and ensures clauses here that
    // ensure that the formula to prove is true.
    decreases n;
    requires n >= 0;
    ensures SumOdds(n) == n*n;
    ensures SumOdds(n) == SumOdds(n-1) + 2*n-1;
{
    // Put a body here that suffices to convince
    // Dafny that the lemma is true.

    if n == 0 { return; }
    ProveSumOdds(n-1);


}

method ComputeSumOddsLoop( n: int ) returns (s: int)
    requires n >= 0;
    ensures s == SumOdds(n);
    ensures s == n*n;
{
    // Put a body here that computes the sum
    // in a loop where you add all the terms
    // in 1+3+5+...+(2*n-1) from left to right.
    // Recursion is not allowed and you may
    // not call ComputeSumOddsRecursive.
    s := 0;
    var k := 0;
    while k != n
        decreases n-k;
        invariant 0 <= k <= n;
        invariant s == SumOdds(k);
        invariant s == k*k;
    {
        k := k + 1;
        s := s + 2*k-1;
    }
}
```

```
method ComputeSumOddsRecursive( n: int ) returns (s: int)
    decreases n;
    requires n >= 0;
    ensures s == SumOdds(n);
    ensures s == n*n;
{
    if n == 0
    {
        s := 0;
        return;
    }
    s := ComputeSumOddsRecursive(n-1);
    s := s + 2*n-1;

}

// If SumOdds is correct then this lemma will work.
lemma SumOddsAll()
    ensures forall n | n >= 0 :: SumOdds(n) == n*n;
{
    forall n | n >= 0
    {
        ProveSumOdds(n);
    }
}
```