```
1  method {:verify false} Sqrt(n:nat) returns (a:nat)
2      ensures a*a ≤ n < (a+1)*(a+1)
3      {
4          var b : nat; //not specifying the nat made dafny mad since we did an
   assignment of nat to b.
5          // assert forall n: nat :: true ⟹ 0*0 ≤ n < n*n
6          // this is not correct, so the assignment a,b := 0,n does not satisfy the
   invariant of the loop (does not established it)
7          assert forall n: nat :: true ⟹ 0*0 ≤ n < (n+1)*(n+1);
8          a,b := 0,n+1;
9          assert a*a ≤ n < b*b;
10         while b≠a+1
11             invariant a*a ≤ n < b*b
12             decreases b*b - a*a // possible : b - a.
13             {
14                 a,b := loopBody(n,a,b);
15             }
16         assert a*a ≤ n < b*b; // the invariant is correct after the loop
17         assert b == a+1; // the negation of the loop guard
18         assert a*a ≤ n < (a+1)*(a+1);
19     }
20
21 method {:verify false} Sqrt'(n:nat) returns (a:nat)
22     ensures a*a ≤ n < (a+1)*(a+1)
23     {
24         var b : nat;
25         a,b := 0,n+1;
26         while b≠a+1
27             invariant a*a ≤ n < b*b
28             decreases b*b - a*a
29             {
30                 var m := (a+b)/2;
31                 if m*m ≤ n
32                 {
33                     a := m;
34                 }
35                 else
36                 {
37                     b := m;
38                 }
39
40             }
41         assert a*a ≤ n < b*b; // the invariant is correct after the loop
42         assert b == a+1; // the negation of the loop guard
43         assert a*a ≤ n < (a+1)*(a+1);
44     }
45
46
47 method {:verify false} loopBody (n:nat, a0:nat, b0:nat) returns (a:nat, b: nat)
48     requires a0*a0 ≤ n < b0*b0 // the loop invariant
49     requires b0 ≠ a0 + 1 // the loop guard
```

```
50      ensures a*a ≤ n < b*b //retuens a,b such that they preserve the loop invariant
51      ensures 0 ≤ b*b - a*a < b0*b0 - a0*a0 //guarantee loop termination
52      {
53          a,b := a0, b0; //convention, since we can't change a0 and b0
54          var m := (a+b)/2 ; //to avoid overflow could say a+(b-1)/2. here in dafny, our
nats and ints are not bounded
55          a,b := update(n,a,b,m); // dafny outputs no errors, update's precond,
postconds match and we just need to implement update's body.
56      }
57
58
59   predicate method Guard (n: nat,a: nat,b: nat,m: nat) // method for making it
executable. just predicate is like a function
60   {
61       m*m ≤ n
62   }
63
64   method {:verify false} update (n:nat, a0:nat, b0:nat, m:nat) returns (a:nat, b: nat)
65      requires a0*a0 ≤ n < b0*b0
66      requires b0 ≠ a0 + 1
67      requires a0 < m < b0
68      ensures a*a ≤ n < b*b
69      ensures 0 ≤ b*b - a*a < b0*b0 - a0*a0
70      {
71          a,b :=  a0,b0;
72          if Guard(n,a,b,m)
73          {
74              a := updateA(n,a,b,m);
75          }
76          else
77          {
78              b := updateB(m,a,b,m);
79          }
80      }
81
82  /**
83  This lemma is supposed to replace the complicated assert forall in updateA.
84  It is used here to help us prove the assignment, and for the reader of the code.
85  The body of the lemma is a proof.
86  note that the lemma should be right on its own (i.e not basing its correctes on past
computations and remarks)
87  lemmas can be called like methods
88   */
89  lemma {:verify false} proofOfUpdateA (n:nat, a:nat,b:nat,m:nat)
90      requires a*a ≤ n < b*b
91      requires b ≠ a + 1
92      requires a < m < b // can be also the exact value of m (the middle)
93      requires Guard (n,a,b,m)
94      ensures m*m ≤ n < b*b //substitution
95      ensures 0 ≤ b*b - m*m < b*b - a*a
96      {
97
```

```
98          assert 0 ≤ a < m; //not mandatory here as oppose to the assert forall used in
    updateA
99      }
100
101  method {:verify false} updateA (n:nat, a0:nat, b:nat, m:nat) returns (a:nat)
102     requires a0*a0 ≤ n < b*b
103     requires b ≠= a0 + 1
104     requires a0 < m < b // can be also the exact value of m (the middle)
105     requires Guard (n,a0,b,m)
106     ensures a*a ≤ n < b*b
107     ensures 0 ≤ b*b - a*a < b*b - a0*a0
108     {
109         a := a0;
110         // assert forall a : nat, n: nat, b: nat, m:nat, a0: nat  ::
111         //   a == a0 && a0 * a0 ≤ n < b*b && b≠= a0 + 1 && a0 < m < b &&
    Guard(n,a0,b,m)
112         //     ⟹ m*m ≤ n < b*b && 0 ≤ b*b - m*m < b*b - a0*a0 by
113         //     {
114         //         assert 0 ≤ a == a0 < m; // to convince dafny that  0 ≤ b*b - m*m
    < b*b - a0*a0 is true, since we know it because a0<m
115         //     }
116         proofOfUpdateA(n,a,b,m) ;
117         a := m;
118     }
119
120
121 lemma {:verify true} proofOfUpdateB (n:nat, a:nat,b:nat,m:nat)
122     requires a*a ≤ n < b*b  //pre0
123     requires b ≠= a + 1     //pre1
124     requires a < m < b      //pre2
125     requires !Guard(n,a,b,m) //pre3: ! (m*m ≤ n)
126     ensures a*a ≤ n < b*b   //post0,1
127     ensures 0 ≤ m*m - a*a < b*b - a*a //post 2,3
128     {
129         //post0
130         assert a*a ≤ n; // from left side of pre0
131         //post1
132         assert n < m*m; //from pre3
133         //post2
134         assert 0 ≤ m*m-a*a; //from post0 and post1 with transitivity
135         assert m*m - a*a < b*b - a*a by {
136             assert m*m < b*b by {
137                 assert 0≤ m < b;
138                 NatSquareMonotinicity(m,b);
139             } //right side from pre2 and m,b are natural numbers
140         }
141     }
142
143 lemma NatSquareMonotinicity (x:nat,y:nat)
144     requires x < y
145     ensures x*x < y*y
146     {
```

```
147
148     }
149  method {:verify false} updateB (n:nat, a:nat, b0:nat, m:nat) returns (b:nat)
150     requires a*a ⩽ n < b0*b0
151     requires b0 ≠ a + 1
152     requires a < m < b0
153     requires !Guard(n,a,b0,m)
154     ensures a*a ⩽ n < b*b
155     ensures 0 ⩽ b*b - a*a < b0*b0 - a*a
156     {
157         b := b0;
158         /**
159         We saw here assume and not assert because we didn't succeed proving the assert
160         //  */
161         //  assert forall a : nat, n: nat, b0: nat, m:nat, a0: nat  ::
162         //   b == b0 && a * a ⩽ n < b0*b0 && b0≠ a0 + 1 && a < m < b0 &&
    Guard(n,a,b0,m)
163         //     ⟹ a*a ⩽ n < m*m && 0 ⩽ m*m - a*a < b0*b0 - a*a by
164         //     {
165         //         assert 0⩽ m <  b0 == b;
166         //     }
167         proofOfUpdateB(n,a,b,m);
168         b := m;
169     }
```