Floor Square Root

Find a function Floor_Sqrt s.t.

> $\{ x \geq 0.0 \}$
> r := Floor_Sqrt(x)
> $\{ r = \lfloor \sqrt{x} \rfloor \}$

where $\lfloor y \rfloor$ ("floor(y)") = greatest integer $\leq$ y **e.g.** $\lfloor 3.14 \rfloor = 3$ and $\lfloor -314 \rfloor = -4$

Alternative Defn. $\lfloor x \rfloor$ "floor(x)"

$$n = \lfloor x \rfloor \equiv n \leq x < n+1$$

e.g. $r = \lfloor \sqrt{x} \rfloor \equiv r \leq \sqrt{x} < r+1$

$\equiv r^2 \leq x < (r+1)^2$

> 'Floor' **Exercise**: Prove, for x:Real: $\lfloor \sqrt{\lfloor x \rfloor} \rfloor = \lfloor \sqrt{x} \rfloor$

Find Floor_Sqrt s.t.

> $\{ x \geq 0.0 \}$
> r := Floor_Sqrt(x)
> $\{ r^2 \leq x < (r+1)^2 \}$

> **Note**: To find $\sqrt{x}$ to n decimal places:
> multiply x by 10^{2n} ;
> Get Floor Square Root;
> Divide the result by 10^n.
> e.g. If x = 2 then Floor_Sqrt(100 * x) / 10 gets $\sqrt{2}$ to one decimal place.

By iterating r until $(r+1)^2 >$ x we get

```
simp_sqrt (x: REAL): INTEGER is
        require
                pre_sq_rt: x >= 0.0
        local
                r: INTEGER
        do
                from
                        r := 0
                until
                        (r + 1) ^ 2 > x
                loop
                        r := r + 1
                end ;
                Result := r
        ensure
                post_sq_rt: Result^2 <= x  and  x < (Result+1)^2
        end ;
```

Alternative Program via Odd numbers

By induction it can be shown that the sum of the first n odd numbers is n^2

$$\sum_{k=1}^{n} 2k - 1 = n^2$$

Other Notation:

(+ k | $1 \le k \le n$: 2k-1) or (Σ k | $1 \le k \le n$: 2k-1)

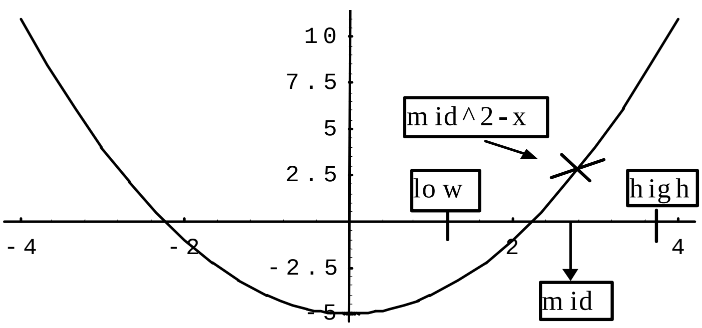
e.g. $\sum_{k=1}^{5} 2k - 1 = 1 + 3 + 5 + 7 + 9 = 5^2 = 25$

By summing odd numbers until result > x we can get floor_sqrt(x) by:

```
floor_sqrt (x: REAL): INTEGER is
    require
        pre_sq_rt: x >= 0.0
    local
        r, n, s: INTEGER
    do
        from
            r := 0; n := 1; s := 1
        until
            s > x
        loop
            r := r + 1; n := n + 2; s := s + n
        end ;
        Result := r
    ensure
        post_sq_rt: Result^2 <= x and  x < (Result+1)^2
    end ;
```

Finding Square Root by Binary Search

To find the square root of x, consider finding (an approximation of) the root of $r^2 - x = 0$ e.g. $x = 5$; $r^2 - 5 = 0$



```
bin_sqrt_r (low,high:REAL;  eps:REAL;  x:REAL):REAL is
        -- (Recursive version)
    require
        within: low ^ 2 - x <= 0 and  0 < high ^ 2 - x
    local
        mid: REAL
    do
        if  low + eps < high then
            mid := (low + high) / 2;
            if  mid^2 - x <= 0 then
                Result := bin_sqrt_r (mid, high, eps, x)
            else
                Result := bin_sqrt_r (low, mid, eps, x)
            end
        else
            Result := low
        end
    ensure
        Result ^ 2 <= x and  x < (Result + eps) ^ 2
    end ;
```

Comment:

The root lies between low and high. We split this interval and find which half the root is in, e.g. if mid^2 - x > 0 then we reset high to be mid (see diagram). More generally, if f(mid) > 0 then reset high to be mid

When function halts, we have

$$\text{high} \leq \text{low} + \text{eps}$$

Also, $\text{low}^2 \leq x < \text{high}^2$,

tf. $\text{low} \leq \sqrt{x} < \text{high}$

At termination we get

$$\text{low} \leq \sqrt{x} < \text{high} \leq \text{low} + \text{eps}$$

i.e. $\text{low} \leq \sqrt{x} < \text{low} + \text{eps}$.

Picking initial interval: (low, high)

Let low := 0;
high := x+1;

tf. we have

$$\text{low}^2 \leq x < \text{high}^2$$

e.g. x = 10,000 tf. $\sqrt{x} = 100$

Initialisation above gives us initial interval (0, 10,001)

Alternative:

Consider a smaller initial interval by finding least power of 2 greater that $\sqrt{x}$,
i.e. least $2^n > \sqrt{x}$

e.g. $x = 10{,}000$ tf. $\sqrt{x} = 100$

Alternative gives initial interval (0, 128).

```
sqrt_r (x: REAL): REAL is
        require
                pre_sqrt: x >= 0.0
        local
                y: REAL
        do
                from
                        y := 1
                until
                        y ^ 2 > x
                loop
                        y := 2 * y
                end ;
                -- 0 ≤ x < y²
                Result := bin_sqrt_r (0, y, 0.0001, x)
        ensure
                post_sqrt: result^2 <= x and x < (result+0.0001)^2
                -- i.e. result ≤ √x < result + 0.0001
        end ;
```

```
root (low,high:REAL;  tiny_val:REAL;
                poly:ARRAY[REAL]):REAL is

- Getting root of a polynomial stored  as an array of
-- coefficients in an array, poly.

    require
        -- monotonic:  Polynomial, poly,  is monotonic in [low, high]

        within:      eval(poly, low) <= 0  and 0 < eval(poly, high)
    local
        mid: REAL
    do
        if  low + tiny_val < high then
            mid := (low + high) / 2;
            if  eval(poly, mid) <= 0 then
                Result := root (mid, high, tiny_val, poly)
            else
                Result := root (low, mid, tiny_val, poly)
            end
        else
            Result := low
        end
    ensure
        eval(poly, Result) <= 0 and 0 < eval(poly, Result + tiny_val)
    end ;
```

The function call, eval(poly, mid) evaluates the polynomial, poly, at the value, mid.