Lecture Recap: Linear Search

(The is_in and is_sorted functions used here are the same as defined in class)

1 `int lin_search(int x, int[] A, int n)`
2 // @requires \( 0 \leq n \&\& n \leq \text{\textbackslash length}(A); \)
3 // @requires \textbf{is_sorted}(A, 0, n);
4 // @ensures \((-1 == \textbf{\textbackslash result} \&\& \text{\textbackslash is\_in}(x, A, 0, n))
5 || ((0 <= \textbf{\textbackslash result} \&\& \textbf{\textbackslash result} < n) \&\& A[\textbf{\textbackslash result}] == x); \) @ /
6 {
7    for (int i = 0; i < n; i++)
8       // @loop_invariant \( 0 \leq i \&\& i <= n; \)
9       // @loop_invariant !\text{\textbackslash is\_in}(x, A, 0, i);
10      {
11         if (A[i] == x) return i; // We found what we were looking for!
12         else if (x < A[i]) return -1; // Can’t possibly be to the right
13         // @assert A[i] < x;
14      }
15    return -1;
16 }

Checkpoint 0

Prove the correctness of linear search. Use the guidelines below:

Loop invariants hold initially

Preservation of loop invariants
Loop invariants imply postcondition

Termination

Linear search for integer square root

We can apply the same concept of linear search to find the integer (since C0 doesn’t have floats!) square root of a given number. The integer square root of \( n \) is defined to be the greatest non-negative integer \( m \), such that \( m^2 \leq n \).

```
1 int isqrt (int n)
2 //@requires n >= 0;
3 //@ensures \result * \result <= n;
4 //@ensures n < (\result+1) * (\result+1) || (\result+1) * (\result+1) < 0;
5 {
6     int i = 0;
7     int k = 0;
8     while (0 <= k && k <= n)
9         //@loop_invariant i * i == k;
10        //@loop_invariant i == 0 || (i > 0 && (i-1)*(i-1) <= n);
11        {
12            // Note: (i + 1)*(i + 1) == i * i + 2*i + 1 and k == i * i
13            k = k + 2*i + 1;
14            i = i + 1;
15        }
16    // This subtraction is necessary since we know k > n now
17    // and i * i == k. i is barely too large to be the square root of n
18    return i - 1;
19 }
```

Note that this function is very similar to the linear search function we discussed. It’s essentially equivalent to searching through a sorted array containing all non-negative ints less than \( n \), looking for the square root of \( n \). There is a similar improved algorithm that we’ll discuss on Friday.
Checkpoint 1
Prove the correctness of this integer square root function
Loop invariants hold initially

Preservation of loop invariants

Loop invariants imply postcondition

Termination
**Checkpoint 2**

A water main break in GHC has, confusingly, broken the C0 compiler’s `-d` option! C0 contracts are now being treated as comments, and the only way to generate assertion failures is with the `assert()` statements.

Insert `assert()` statements into the code below so that, when the code runs, all operations (C0 statements, conditional checks, and assertions) are performed at runtime in the *exact same sequence* that would have occurred if we compiled with `-d`. Not all of the blanks need to be filled in.

```c
int mult(int x, int y)
//@requires x >= 0 && y >= 0;
//@ensures \result == x*y;
{
    int k = x; int n = y;
    int res = 0;

    while (n != 0)
        //@loop_invariant x * y == k * n + res;
        {
            if ((k & 1) == 1) res = res + n;
            k = k >> 1;
            n = n << 1;
        }

    return res;
}

int main() {
    int a;

    a = mult(3,4);

    return a;
}
```