# 15-122: Principles of Imperative Computation

## Recitation 5

Josh Zimmerman

## Basic linear search: recap

(Note: I assume the `is_in` and `is_sorted` functions exist as defined in class.)

```
1  int lin_search(int x, int[] A, int n)
2  //@requires 0 <= n && n <= \length(A);
3  //@requires is_sorted(A, 0, n);
4  /*@ensures (-1 == \result && !is_in(x, A, 0, n))
5        || ((0 <= \result && \result < n) && A[\result] == x); @*/
6  {
7     for (int i = 0; i < n; i++)
8     //@loop_invariant 0 <= i && i <= n;
9     //@loop_invariant !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 }
```

Now, let's look at this code and see if we can prove that it works. Work on your own or with other people to follow the four-step process to proving that linear search works. (Remember: Show that the loop invariants hold initially, that they are preserved, that the loop invariants and the negation of the loop condition imply the postcondition, and that the loop terminates.)

---

I claim we can search a sorted array faster than this. We'll discuss why in lecture tomorrow, but for now try to think about how you could improve on this search method.

## Linear search for integer square root

Recall linear search from lecture. We can apply the same concept 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.

## Contract Checking

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 occured if we compiled with `-d`. Not all of the blanks need to be filled in.

```
1  int mult(int x, int y)
2  //@requires x >= 0 && y >= 0;
3  //@ensures \result == x*y;
4  {
5    /* 1 */                        /* 1 */_____________________________
6    int k = x; int n = y;
7    int res = 0;
8
9    /* 2 */                        /* 2 */_____________________________
10   while (n != 0)
11   //@loop_invariant x * y == k * n + res;
12   {
13      /* 3 */                     /* 3 */_____________________________
14      if ((k & 1) == 1) res = res + n;
15      k = k >> 1;
16      n = n << 1;
17      /* 4 */                     /* 4 */_____________________________
18    }
19   /* 5 */                        /* 5 */_____________________________
20
21   /* 6 */                        /* 6 */_____________________________
22   return res;
23   /* 7 */                        /* 7 */_____________________________
24 }
25
26 int main() {
27   int a;
28
29   /* 8 */                        /* 8 */_____________________________
30   a = mult(3,4);
31
32   /* 9 */                        /* 9 */_____________________________
33   return a;
34 }
```