15-122: Principles of Imperative Computation

Recitation 5 — Josh Zimmerman

Lecture Recap: Linear Search

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

```
int lin_search(int x, int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@requires is_sorted(A, 0, n);
/*@ensures (-1 == \result && !is_in(x, A, 0, n))
     || ((0 <= \result && \result < n) && A[\result] == x); @*/
{
  for (int i = 0; i < n; i++)
  //@loop_invariant 0 <= i && i <= n;
  //@loop_invariant !is_in(x, A, 0, i);
  {
    if (A[i] == x) return i; // We found what we were looking for!
    else if (x < A[i]) return -1; // Can't possibly be to the right
    //@assert A[i] < x;
  }
  return -1;
}
```

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 $m^2 \leq n$.

```
int isqrt (int n)
//@requires n >= 0;
//@ensures \result * \result <= n;
//@ensures n < (\result+1) * (\result+1) || (\result+1) * (\result+1) < 0;
{
   int i = 0;
   int k = 0;
   while (0 <= k && k <= n)
   //@loop_invariant i * i == k;
   //@loop_invariant i == 0 || (i > 0 && (i-1)*(i-1) <= n);
   {
      // Note: (i + 1)*(i + 1) == i * i + 2*i + 1 and k == i * i
      k = k + 2*i + 1;
      i = i + 1;
   }
   // This subtraction is necessary since we know k > n now
   // and i * i == k. i is barely too large to be the square root of n
   return i - 1;
}
```

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

```
int mult(int x, int y)
//@requires x >= 0 && y >= 0;
//@ensures \result == x*y;
{
  /* 1 */                           /* 1 */______________________________
  int k = x; int n = y;
  int res = 0;

  /* 2 */                           /* 2 */______________________________
  while (n != 0)
  //@loop_invariant x * y == k * n + res;
  {
     /* 3 */                        /* 3 */______________________________
     if ((k & 1) == 1) res = res + n;
     k = k >> 1;
     n = n << 1;
     /* 4 */                        /* 4 */______________________________
  }
  /* 5 */                           /* 5 */______________________________

  /* 6 */                           /* 6 */______________________________
  return res;
  /* 7 */                           /* 7 */______________________________
}

int main() {
  int a;

  /* 8 */                           /* 8 */______________________________
  a = mult(3,4);

  /* 9 */                           /* 9 */______________________________
  return a;
}
```