CSE 160A, Discrete Mathematics, Winter 1993
Using loop invariants to design and analyze algorithms

The examples of programs that Epp uses the loop invariant method to analyze are rather simple and inefficient. Here are some more complicated examples, where we are also interested in the running time as well as correctness of the algorithms. Some of these examples will be discussed in class.

Notation: for r a real number , $\lfloor r \rfloor$ represents the integer part of r, r rounded down to the nearest integer. For example $\lfloor 32.7 \rfloor = 32$, $\lfloor 15.1 \rfloor = 15$, $\lfloor 13 \rfloor = 13$. Formally, we can define $\lfloor r \rfloor$ as the (unique) integer y so that $r \geq y > r - 1$.

Approximating the square root of a positive integer Discussion: We want to write a program that takes as input a positive integer x and computes $\lfloor \sqrt{x} \rfloor$. We could just square all possible numbers less than x until we find one bigger, but this is very slow. Instead, let's use the fact that $\lfloor \sqrt{x} \rfloor \geq a$ is the same as $a^2 \leq x$ to narrow in on $\lfloor \sqrt{x} \rfloor$ using "binary search". In binary search, you are trying to locate an unknown number z by asking questions of the form "is z bigger than y?". If you know z is between max and min, the best value to ask is $y = (max + min)/2$, since then either answer halves the number of possibilities. This means that in about $log_2(max - min)$ questions we can find the value of z to the nearest integer. We have to worry about rounding when we take the average, unless $max - min$ is a power of 2. So in the following program, we first try to find values for max and min of the form 2^{k+1} and 2^k so that $max - min = 22^k - 2^k = 2^k$. Then in the binary search we will want to maintain : $max > \lfloor \sqrt{x} \rfloor \geq min$ and $max - min$ is always a power of 2 and getting smaller as the loop progresses.

Program: SqareRoot

Input: Integer x;{we assume x ¿0}

beginenumerate

begin;{program}

{In the first stage we want to find the unique value 2^k so that $2^k \leq \lfloor \sqrt{x} \rfloor < 2^{k+1}$}

z:=1;

while $4z^2 \leq x$ do: z :=2z; {after k times through the loop $z = 2^k$, stop when $2z > \lfloor \sqrt{x} \rfloor$}

{In the second stage, we use binary search to find $\lfloor \sqrt{x} \rfloor$}

max:= 2z;

min:= z;

while $max > min + 1$ do :

begin;{while}

ave := $(max + min)/2$;{most useful comparison question is average}

IF $(ave^2 \leq x)$ THEN min:= ave ELSE max:= ave;{want to maintain $min \leq \sqrt{x} < max$.}

end;{while}

output min {at end of loop,$max = min+1$ and $min \leq \lfloor\sqrt{x}\rfloor < max = min+1$ so $min = \lfloor\sqrt{x}\rfloor$. }

end;{program}

Proof of correctness: The following proof has all the technical bells and whistles. This makes it longer and more complicated than if I were being slightly less careful, so don't be intimidated! We want to show:

Theorem: On any input x that meets the pre-condition $x > 0$. the program outputs $\lfloor\sqrt{x}\rfloor$.

Proof: The outline of the proof follows the comments in the program. We first show that the first while loop indeed finds two consecutive powers of two "sandwiching" $\lfloor\sqrt{x}\rfloor$.

Lemma 1: (Post-condition,first while loop) On any input $x > 0$, the first loop eventually terminates and there is an integer $k \geq 0$ so that the value of z at the end is 2^k and so that $2^k \leq \lfloor\sqrt{x}\rfloor < 2^{k+1}$.

Proof of lemma 1: We first show the following by induction on t:

Claim 2: If the first while loops runs for at least t loops, then the value of z_t of z after the t'th loop is given by $z_t = 2^t$.

Proof of claim 2: We prove this by induction on t.

Base Case: $t = 0$. For $t = 0$, after 0 loops, z_0 is the value z is initialized to, 1. So $z_0 = 1 = 2^0$, so the claim is true for $t = 0$. $\square$(Claim 2)

Induction Step: Assume the claim is true after t loops, i.e., $z_t = 2^t$. Assume the program loops once more. Then the new value of z, $z_{t+1} = 2z_t = 22^t = 2^{t+1}$. Thus if the claim is true after t loops, then it is true after $t + 1$ loops.

Therefore, since the claim is true for 0 loops, and if it is true for t loops it is also true for $t + 1$ loops, by induction it is true for any number of loops.

We now use the claim to prove Lemma 1. Let $t_0 = \lfloor\log_2\lfloor\sqrt{x}\rfloor\rfloor$. Then $t_0 > \log_2\sqrt{x} - 1$ by the definition of the floor function. So if the program loops at least t_0 times, by the claim the value of $z_{t_0} = 2^{t_0} > 2^{\log_2\sqrt{x}-1} > 1/2\sqrt{x}$ and $4z_{t_0}^2 > x$. So after no more than t_0 steps, the guard will fail and the loop will terminate. Say it terminates after t_1 steps. Then the value of z when it does so satisfies: $z = 2^{t_1}$ and $4z^2 > x$ (since now the guard fails) and $4(z/2)^2 \leq x$ (since the guard was true before the t_1'th loop.). Thus, $2z > \sqrt{x} \geq z$ and $z = 2^{t_1}$ is a power of 2. □(Lemma 1)

We now prove that the second loop maintains the following invariants: Let t_1 be the number of loops the first loop took, as above.

Claim 3: For any t, if $0 \leq t \leq t_1$, then the second loop runs at least t times, and the values min_t and max_t of min and max after the t'th loop satisfy the following: both min_t and max_t are integers, $min_t \leq x < max_t$, and $max_t - min_t = 2^{t_1 - t}$.

We prove this claim by induction on t.

Base Case: Before the loop $max_0 = 2z, min_0 = z$. By Lemma 1, $z = 2^{t_1}$ and $min_0 = z \leq \sqrt{x} < 2z = max_0$. So the first two parts of the claim for $t = 0$ are true. Then $max_0 - min_0 = 2z - z = z = 2^{t_1} = 2^{t_1 - 0}$ so the last part is also true.

Induction Step: Assume, for some $0 \leq t < t_1$, the claim is true. (Note that if $t + 1 > t_1$, the claim is automatically true for $t + 1$ since one false statement implies a false statement, so the only non-trivial case is $t < t_1$. Thus we are justified in assuming $t < t_1$.) In other words,min_t and max_t are integers, $min_t \leq \sqrt{x} < max_t$, and $max_t - min_t = 2^{t_1 - t}$. Since $max_t = min_t + 2^{t_1 - t} \geq min_t + 2$, the loop will execute again. In this loop $ave_{t+1} = (max_t + min_t)/2 = (2min_t + 2^{t_1 - t})/2 = min_t + 2^{t_1 - t - 1}$.

Case 1: If $ave_{t+1}^2 \leq x$, then $max_{t+1} = max_t = min_t + 2^{t_1 - t}$ and $min_{t+1} = ave_{t+1} = min_t + 2^{t_1 - t - 1}$. So both are integers, and $min_{t+1} = ave_{t+1} \leq \sqrt{x} < max_t = max_{t+1}$. Finally, $max_{t+1} - mint + 1 = min_t + 2^{t_1 - t} - min_t - 2^{t_1 - t - 1} = 2(2^{t_1 - t - 1}) - 2^{t_1 - t - 1} = 2^{t_1 - (t+1)}$. So all parts of the claim are true in this case.

Case 2: If $ave_{t+1}^2 > x$, then $min_{t+1} = min_t$ and $max_{t+1} = ave_{t+1} = min_t + 2^{t_1 - t - 1}$. So both are integers, and $min_{t+1} = min_t \leq \sqrt{x} < ave_{t+1} = max_{t+1}$. Finally, $max_{t+1} - mint + 1 = min_t + 2^{t_1 - t - 1} - min_t = 2^{t_1 - (t+1)}$. So all parts of the claim are true in this case.

Thus in either case, all parts of the claim are true for $t + 1$.

Thus, by the principle of induction the claim is true for all t. □ (Claim 3)

Now we prove the main theorem: Applying claim 3 to the case $t = t_1$, we see that after the t_1'th loop max and min have integer values with $min \leq sqrtx < max$, and $max = min + 2^{t_1 - t_1} = min + 1$. Then since $max = min + 1$, the loop halts. Also, $sqrtx - 1 < max - 1 = min \leq sqrtx$, so $min = \lfloor \sqrt{x} \rfloor$ by the definition of the floor function. □ (Theorem).

Time Analysis: The above proof also gives us a bound on how long the program will take . The proof of Lemma 1 showed that the first loop executes at most $\lfloor \log_2 \sqrt{x} \rfloor \leq 1/2log_2 x$ times, Claim 3 shows that the second loop executes the same number of times. Thus the total number of multiplications is at most $log_2 x$, the length of x in binary. This is much better than the x multiplications that exhaustive search would take.

Also note, that while better algorithms are possible for square roots, the same method above would work to compute the inverse function for any easily computable increasing function f. (Just replace squaring by computing f everywhere.) The number of times you would need to compute f is $2log_2(f^{-1}(x))$.