Practice Problems:

1. Use a loop invariant to prove that the following program is correct with respect to the initial assertion that $x$ is a positive integer and the final assertion that $\text{ans} = x^2$.

```
procedure square(x)
    i = 1
    j = 1
    while (i < x) do
        j = j + 2i + 1
        i = i + 1
    od
    return j
```

Solution:

Our Loop invariant is LI: $j = i^2$

You don’t really need to justify this, but this captures the idea that $j$ seems to be increasing as $i$ gets bigger through the loop, and when $i = x$, this will give us the answer that we want.

First, I label the relevant bits of code.

```
procedure square(x)
    i = 1 % SNIPPET 1
    j = 1 %
    while (i < x) do % CONDITION 1
        j = j + 2i + 1 % SNIPPET 2
        i = i + 1 %
    od
    return j
```

Then our proof format will be:

**STEP 1:** {} SNIPPET1 {LI}

**STEP 2:** {LI AND CONDITION 1 } SNIPPET2 {LI}

**STEP 3:** {LI ^ NOT CONDITION 1 } SNIPPET2 {j = x^2} j

**STEP 4:** Argue that the code always stops

**Step 1:**

```
{}
```

SNIPPET1 {LI}

SNIPPET 1 assigns $i=1$ and $j=1$

To prove that the loop invariant, notice that

$j = i^2$ becomes $1 = 1^2$ which is true.
Step 2:

\{LI \text{ AND CONDITION 1} \} \quad \text{SNIPPET2} \quad \{LI\}

After SNIPPET 2 runs:

\[ j' = j + 2i + 1 \]
\[ i' = i + 1 \]

Now we need to prove that \( j' = (i')^2 \)

\[ j' = j + 2i + 1 \]
\[ j' = i^2 + 2i + 1 \quad \% \text{ by the Loop invariant that held true before the loop} \]
\[ j' = (i+1)^2 \quad \% \text{ Factoring!} \]
\[ j' = (i')^2 \quad \% \text{ because } i' \text{ is } i+1 \]

So LI is true at the end of the loop

Step 3

\{LI \text{ AND NOT CONDITION 1} \} \rightarrow \{j = x^2\}

When the loop concludes
\( i = x^2 \), because \( i \) is incremented by 1 each time through the loop and the
loop ends when the condition \( i < x \) fails.

by the loop invariant,
\[ j = i^2 \]

but \( i = x \), so
\[ j = x^2 \]

STEP 4:
Every time through the loop \( i \) is increased by 1. \( I \) starts at 1 and ends when it gets to the input value \( x \), so the loop will run exactly \( x \) times, and therefore the loop ends.

2. Use a loop invariant to prove that the following program is correct with respect to the initial assertion that \( n \) is an integer \( \geq 0 \) and the final assertion that palindrome(n) returns the integer obtained by reversing the digits in \( n \). (For example, if \( n = 1432 \) then it should return 2341.). If it helps, you can use ”Reverse(x)” in your logical arguments to represent the number that you would get by reversing the digits of \( x \).

procedure palindrome(n)

reverse = 0
m = n
while m > 0
    temp = m % 10 \quad (\% \text{ is ”mod”, so } m \% 10 \text{ is } m \mod 10)
    reverse = reverse * 10 + temp
    m = (m - temp) / 10
return reverse

First, we label bits of code.
reverse = 0 % SNIPPET 1
m = n % .
while m > 0 % CONDITION 1
    temp = m%10 % SNIPPET 2
    reverse = reverse * 10 + temp % .
    m = (m - temp) / 10 % .
return reverse

Then, if you trace out what this loop does (really, you should do that!!), you will see a pattern in the values of reverse (the slowly growing answer) and m the number that started at n and is being slowly deconstructed.

Loop invariant LI: write down m then REVERSE(reverse) to get “n” (or, written another way, turn m and REVERSE(reverse) into strings and concatenate them to get n).

[Side note, sorry about naming a variable reverse and naming this as a function. No “pretty point” for this homework assignment for me].

STEP 1:
{ } SNIPPET 1 {LI}
The code sets: reverse = 0, m = n

m as a string is exactly n
REVERSE(0) is a zero length string

so writing down m gives us exactly n

STEP 2:

Prove
{LI} SNIPPET 2 {LI}

thinking of "m" and "reverse" in terms of strings, what does the loop do?

    temp is last digit of m
    m' = m without last digit
    reverse' = concatenate(reverse, tmp).

so what is:

    m' + REVERSE(reverse') % let ‘+’ here concatenate strings
= m without last digit + REVERSE(reverse, last digit of m) =
= m without last digit + last digit of m + REVERSE( reverse )
= m + REVERSE(reverse)
.... which is n by the Loop invariant.

STEP 3:

{LI AND CONDITION} --> reverse = REVERSE(n)

When the loop ends, m = 0.

By the LI, we know that

    m + REVERSE(reverse) = n

and m is zero, so
REVERSE(reverse) = n

this equality of strings holds if we reverse both sides:

REVERSE(REVERSE(reverse)) = REVERSE(n)

and the process of reversing a string is its own inverse, so

reverse = REVERSE(n)

Yatta!

STEP 4:
Every time through the loop, the value of m is decreased by a factor of 10 (or, the number of digits in base 10 of m is reduced by 1. So in a finite number of steps, m will become 0 and the loop condition will fail.

Problems to turn in:

1. Use the idea of the “fast exponentiation” algorithm discussed in class to make a “faster multiplication” algorithm that returns the product of (x,y). Your program should be asymptotically faster than one that repetively adds x to itself y times. (or y to itself x times). You are not allowed to multiply in your algorithm. You are permitted to divide by two. [as an aside, for binary numbers, this can be implemented as a simple bit shift, so it isn’t really cheating]. Finally, you should prove that your code is correct.

   multiply(x,y)
   
   a = x;
   b = y;
   answer = 0;
   while b > 0 do
      if b is even
         a = a + a;
         b = b/2;
      else
         answer = answer + a;
         b = b-1;
      fi
   od

(1b) Prove that your algorithm is correct.

First, lets break the code up into chunks.

S1 = "a = x;
    b = y;
    answer = 0"

loop = everthing between "while" and "od".
loopOne = "a = a + a;
    b = b/2;"

loopTwo = "answer = answer + a;
    b = b-1;"

ok... So, the preconditions should be:

P <=] P, y are positive integers."

then our loop invariant:

L <=] answer + a*b = x*y

So, lets prove our step by step thing, starting with the precondition:

P {S1} L:

Well, S1 assigns answer = 0, a = x, and b = y. So
answer + a * b = 0 + x * y = x * y. weehaa!

Next, lets prove that L {loop} L

we need to do this with two cases (because there is an if statement
in the loop).

Assume L is true before the loop, ( so "answer + a * b = x * y" ).

Case One: b is even.

After the loop, a' = a+a, and b' = b/2.

So, answer + a' * b' =
    answer + (a+a) * b/2 =
    answer + 2*a * b/2 =
    answer + a * b = ... and from above, we assume this is x * y.

Case Two:

After the loop, answer' = answer + a, and b' = b-1.

So, answer' + a * b' =
    (answer + a) + a * (b-1) =
    (answer) + a + a * (b-1) =
    (answer) + a (1 + (b-1)) =
    (answer) + a ( b ) =
    answer + a * b = ... and from above, we assume this is x * y.

so in either case, L { loop } L.

Finally, we need to prove that (L and not (b > 0)) --> answer = x*y.
So, when the loop ends, it ends because b is no longer greater than zero. In this case b = 0. So, because L is true, answer + a * b = x * y, but, since the loop is over, b = 0, so:
answer + a * 0 = x * y,
answer = x * y,
which is what we should, long ago, have claimed as our final assertion.

Lastly, we need to argue that the loop always ends. In either case (when b is even or odd), the value of b decreases by at least 1. Therefore, eventually the loop stops.

2. Define the following “Fibonacci-Like” recursive sequence:

   \( a_1 = 1, a_2 = 5, \text{ and } \forall n \geq 3, a_n = 5a_{n-1} - 6a_{n-2} \)

   (a) Compute the terms \( a_3, a_4, a_5 \) (calculators are acceptable).

   \[
   \begin{align*}
   a_3 &= 5a_2 - 6a_1 = 25 - 6 = 19 \\
   a_4 &= 5a_3 - 6a_2 = 95 - 30 = 65 \\
   a_5 &= 5a_4 - 6a_3 = 325 - 114 = 211
   \end{align*}
   \]

   (b) Prove by strong induction that \( \forall n \geq 1, a_n = 3^n - 2^n \)

   **Base Case:** \( P(1) \). We need to prove \( a_1 = 3^1 - 2^1 \)

   \( a_1 = 1 \).

   Also, \( 3^1 - 2^1 = 3 - 2 = 1 \), so \( a_1 = 3^1 - 2^1 \)

   **Induction Step. Assume** \( P(k) \text{ and } P(k - 1) \).

   [Since we are assuming more than just \( P(k) \), this counts as strong induction.] Explicitly, this assumption is:

   \( a_k = 3^k - 2^k \text{ and } a_{k-1} = 3^{k-1} - 2^{k-1} \)

   **Now prove** \( P(k-1) \): \( a_{k+1} = 3^{k+1} - 2^{k+1} \)

   \[
   \begin{align*}
   a_{k+1} &= 5 \cdot a_k - 6 \cdot a_{k-1} \\
   &= 5 \cdot (3^k - 2^k) - 6 \cdot (3^{k-1} - 2^{k-1}) \\
   &= 5 \cdot 3^k - 10 \cdot 2^k - 6 \cdot 3^{k-1} + 6 \cdot 2^{k-1} \\
   &= 15 \cdot 3^{k-1} - 10 \cdot 2^{k-1} - 6 \cdot 3^{k-1} + 10 \cdot 2^{k-1} \\
   &= 9 \cdot 3^{k-1} - 4 \cdot 2^{k-1} \\
   &= 3^2 \cdot 3^{k-1} - 2^2 \cdot 2^{k-1} \\
   &= 3^{k+1} - 2^{k+1}
   \end{align*}
   \]

   So \( P(k) \text{ and } P(k-1) \rightleftharpoons P(k + 1) \),

   so, \( \forall n \geq 1{\text{P}}(n) \)