CSC236 2015 Winter, Assignment 2
Due Monday March 9th at 8PM

Notice: The due date for this assignment has been postponed to Monday March 9th at 8PM.
You may work in groups of up to three people currently enrolled in CSC236.
Submit your solutions as a PDF file named `a2.pdf` to MarkUs.
Your file must be produced using a word processor or editor that exports files in PDF format (no scanned handwriting accepted).
Late assignments have a deduction of 5% per hour, for up to 20 hours.
You will receive 20% of the marks for any question (or part of a question) that you either leave blank or for which you write "I cannot answer this."

1. Let the set of boolean formulas $\mathcal{F}$ be defined inductively by:

   - The "variables" $X_1, X_2, \ldots$ are in $\mathcal{F}$.
   - If $A \in \mathcal{F}$ then $\neg A \in \mathcal{F}$.
   - If $A \in \mathcal{F}$ and $B \in \mathcal{F}$ then $A \wedge B \in \mathcal{F}$ and $A \vee B \in \mathcal{F}$.

   Let the pair of mutually-recursive functions T and N from $\mathcal{F}$ to $\mathcal{F}$ be defined by:

   If $A \in \mathcal{F}$ and $B \in \mathcal{F}$ and $i \in \mathbb{N}$ then

   $$\begin{aligned} T(A \wedge B) &= T(A) \wedge T(B) \\ T(A \vee B) &= T(A) \vee T(B) \\ T(\neg A) &= N(A) \\ T(X_i) &= X_i, \\ N(A \wedge B) &= N(A) \vee N(B) \\ N(A \vee B) &= N(A) \wedge N(B) \\ N(\neg A) &= T(A) \\ N(X_i) &= \neg X_i \end{aligned}$$

   (a) Prove by Structural Induction on the definition of $\mathcal{F}$ that for all $A \in \mathcal{F}$:

   $T(A)$ is logically equivalent to A and has negation only on variables.

   To do so, define a stronger predicate that includes a claim about function N.

   (b) Write a recursive Python function to compute T, but **without writing any helper functions**. Use the following representation of boolean formulas:
   - Variables are represented by their natural number index, e.g. X_{236} is represented by 236.
   - If A represents a boolean formula then so does two-element Python tuple ('not', A).
   - If A and B represent boolean formulas then so do the three-element Python tuples (A, 'and', B) and (A, 'or', B).

   (c) Prove that your Python function from (b) is correct: if A represents a boolean formula then $T(A)$ returns a logically equivalent representation that has negation only on variables. To do so, define (in your proof, not in code) a "size" function for boolean formulas to do Complete Induction on.

2. Consider the following function g that takes two positive natural numbers and returns a triple of natural numbers.

```
# PRE-condition: m and n are positive natural numbers.
# POST-condition: returns a tuple of natural numbers (k, l, b) such that k*m = l*n = b.
def g(m, n):
  a = m
  b = n
  k = 1
  l = 1
  while a != b:
    if a < b:
      a += m
      k += 1
    else:
      b += n
      l += 1
  return (k, l, b)
```

(a) Prove that g is correct.
NOTE: if you wish, you may skip this and do only parts (b) and (c).

(b) Prove that g with the following strengthened post-condition is correct:

```
# POST-condition: returns a tuple of natural numbers (k, l, b) such that k*m = l*n = b,
#  and if natural number c is a positive multiple of both m and n then b <= c.
```

If you solved part (a), you may use that result and/or reference parts of your proof of it here.

(c) Prove that the following function h, which is a version of g that doesn't include k and l, is correct:

```
# PRE-condition: m and n are positive natural numbers.
# POST-condition: returns a natural number b that is a multiple of both m and n,
#  and if natural number c is a positive multiple of both m and n then b <= c.
def h(m, n):
  a = m
  b = n
  while a != b:
    if a < b:
      a += m
    else:
      b += n
  return b
```