REDUCE / ACCUMULATE / FOLD.

We will use the terms accumulate, fold, and reduce interchangeably.

Here is a definition of the accumulate function that we will use:

```
(define (accumulate op initial sequence)
  (if (null? sequence)
      initial
      (op (car sequence)
           (accumulate op initial (cdr sequence)))))
```

**Problem 1.** Using the substitution model, carry out the following statement, including all expansions and reductions to the final numeric result. You may directly evaluate the if statement (don’t write it out), and any

```
(accumulate + 0 '(1 2 3))
```

**Problem 2.** Does the definition above produce a recursive process or an iterative one? Why?
Problem 3. Accumulate can also be used to transform lists into new lists. The combining function becomes `cons` and the initial element is the empty list `'( )`.

E.g.,

\[(\text{accumulate } \text{cons } '( ) '(1 2 3))\]

will cdr down down the input list and build up a new one that is a copy of it.

Based on this insight, write an `op` function that will double each element in a list. Hint: it will be lambda function of two parameters that uses `cons`.

Problem 4. Use accumulate to write a version of filter.

Here is a skeleton of the function. You will write a lambda expression for the `op` function. Below, `pred` is the filter predicate function which will operate on individual elements of the input `lst`.

\[(\text{define } (\text{filter-using-accumulate} \text{ pred } \text{ lst})
  \quad (\text{accumulate}
    \quad (\lambda (a b) <\text{fill in here}> )
    \quad '(())
    \quad \text{lst}))\]

Write the entire lambda expression.