1. Write a composition function that takes 2 functions which have the following signatures:
   1. f = fn : 'b -> 'c option
   2. g = fn : 'a -> 'b option
   3. The composition is done in such a way that
      1. If either f or g return NONE, the entire function returns NONE
      2. if g returns SOME <value1> the returned value becomes input to f.
      3. If the value returned by is SOME <value2> then <value2> is the return value of the whole function.
2. Write a do_until function that takes a function (f) and a predicate (p) (a function that takes a value and returns a boolean). It will apply the f repeatedly on the input variable until the predicate applied to the value returned by f becomes false. eg.: `do_until (fn x => x + 1) (fn x => x <= 100) x` will keep incrementing x (as per first anonymous function) until the value become > 100 (as per predicate function)
3. Use the above function to implement factorial.
4. Write a map function for pairs; each value in a pair of type ('a * 'a) gets mapped to a pair of type ('b * ‘b) eg. If a doubles function used passed to the map function (2, 3) will return (4, 6).
5. Use builtin List.foldl, String.size functions to find the longest string in a list of strings. If there are more than 1 strings with a max length, return the string that is closer to the beginning of the list. Eg: ["a", "abc", "defg", "hi", "jkl", "mnop", "qr"] should return "defg" (not “mnop") as it occurs earlier in the list.
6. Write a second version of the function above which returns the string that is closer to the end of the list in case of a tie. Eg: ["a", "abc", "defg", "hi", "jkl", "mnop", "qr"] should return "mnop" (not "defg")
7. Use MLs o operator to derive a function that reverses a string and capitalizes the string. Passing “Hello" returns “olleh". Use builtin functions from the SML library.