Your homework is due at 11:59pm on Sept 26, 2018. You will use the Learn OCaml platform to submit and automatically grade your homework. You can submit your homework as often as you like until the due date and immediately check your grade.

Important note: some questions in this exercise are optional. You can still run the grader on these questions to check your implementation, but they will be worth 0 points. If you complete only the mandatory questions, the grader may tell you the exercise is incomplete, even if you get a score of 100%. You can remove this warning by deleting the find_all and eval functions, or reverting them to their default behaviour of raise NotImplemented.

Tries

Tries are an important data-structure in computer science. In this exercise, we use a trie to efficiently store words by sharing their prefix strings. We will represent a dictionary using a list of tries.

Example: To illustrate, consider the following problem of storing the words bee, beer, beef, beefy, bear, beard, and bree. We will use a trie to store these words in such a way that prefixes are shared. Every node contains one character, and every path in the trie corresponds to exactly one word. The leftmost path for example corresponds to the word bee.

The trie is a tree, where each node contains one character and has an arbitrary number of children. We use the following data-structure to represent a trie in OCaml.

type 'a trie = Node of 'a * ('a trie) list | Empty
Empty marks the end of a word and corresponds to • in the picture above. The representation of the dictionary containing the words above is implemented as the list of tries t defined in the prelude.

1. Before we begin, we implement two functions to simplify the manipulation of words:

- string_explode : string -> char list
- string_implode : char list -> string
  string_explode turns a string into a list of characters and string_implode turns a list of characters back into a string. To implement these two functions, use a selection of the following higher-order functions: List.map, List.fold_right, List.fold_left and tabulate. tabulate is implemented for you in the prelude.

  You may also find the following functions from the OCaml string and char libraries useful:

- String.get : string -> int -> char returns the character at index n of string s.

- String.length : string -> int returns the length (number of characters) of the given string.
- Char.escaped : char -> string returns a string representing the given character
  In order to get the full 10 marks for each question, you **must** use higher-order functions. Solutions using manual recursion will be capped at half marks, with each test case worth only 1 point instead of 2.

2. Complete the function insert which allows us to insert a string s into a list of tries t, by writing a function ins : char list -> char trie list -> char trie list. Use the helper function unrollthat is implemented for you in the prelude.

   A given word may be inserted multiple times, and we allow tries to contain duplicates (however, the resulting dictionary must not contain multiple tries with the same key). It does not matter where in the list you insert new nodes.

3. Complete the function lookup which allows us to check whether a given string w is in the list of tries t.
   For this task write a function lkp : char list -> char trie list -> bool. This function returns true, if there exists a trie t' in t such that the trie t' has a path which corresponds to l, i.e. the word corresponding to l is in the list of tries t. The function returns false otherwise. Use the helper function contains_empty that is implemented for you in the prelude.

   Optional: Challenge Question

   The following question is optional and will not count towards your grade. You can still see your test results by running the grader, but they will be worth 0 points.

4. Complete the function find_all : string -> char trie list -> string list which returns a list of all words with the given prefix in the given trie list. If no path matching the given prefix exists in the trie list, you should return an empty list.
   For this task write an auxiliary function find_all' which takes as input a list of characters corresponding to the prefix, and a trie list.

   Here are a few hints:

- Think how you would need to modify the earlier function lkp to achieve what you want.
- Use higher-order functions in appropriate places.
- We strongly recommend writing a helper function to_words : 'a trie list -> 'a list listthat returns a list of all the words contained in a dictionary (with each word represented as a list of characters).

  Optional: More Higher Order Functions

  The following question is optional and will not count towards your grade. You can still see your test results by running the grader, but they will be worth 0 points.

  For this question, we will make use of the type labeled_pred as defined in the prelude:
  type labeled_pred = string * (int -> bool)
  A labeled predicate is a function int -> bool, paired with a label string describing what the function does. We use the labeled predicate type in this exercise solely to allow the grader to

indicate what functions it is using for grading, as function values are simply printed as <fun>. See the prelude for some example values of type labeled_pred.

5. Consider the following logical statement, which contains a bound variable y and free variables P, Q, N, and x:

   $$\forall y \in [0,N):(P(y) \text{ or } Q(x+y))$$

   Write a function eval : labeled_pred -> labeled_pred -> int -> int -> bool which takes as input two predicates p and q together with their labels (which you can safely discard as per the template) and an integer n, and returns a function. This returned function, when given an integer x as input, should evaluate the statement above, with P, Q, N, and x bound to their respective inputs, returning either true or false.
   Use any function you have written before or functions discussed in class. Your answer should be very short, and not more than one or two lines. Make appropriate use of higher-order functions. We recommend looking at the documentation for OCaml's List module.
   Note that [0,N) is the set of integers $\{0,1,2,\ldots,N-1\}$.