2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

(a) A prime number sieve is an algorithm for finding all prime numbers up to a given limit \( n \). The algorithm maintains a list, which initially holds the integers from 2 to \( n \). The following step is then repeated: remove the head of this list, which will be a prime number, and remove all its multiples from the list. Write code for the algorithm above as an OCaml function of type \( \text{int} \rightarrow \text{int list} \). [4 marks]

(b) Consider the problem of eliminating all duplicates from a list of strings. Write code for a function of type \( \text{string list} \rightarrow \text{string list} \) such that the output contains the same elements as the input, possibly reordered, but where every element occurs exactly once. The worst-case performance must be better than quadratic in the length of the list. [6 marks]

(c) Consider the task of determining whether a given word (a string) can be expressed by joining together various chunks (non-empty strings). If the chunks are \( \text{abra} \), \( \text{cad} \) and \( \text{hal} \), then the word \( \text{abracadabra} \) can be expressed as \( \text{abra|cad|abra} \). Note that if the available chunks are \( \text{ab} \), \( \text{bra} \), \( \text{cad} \) and \( \text{abra} \), then the first two are no good for expressing \( \text{abracadabra} \), and yet a solution can be found using \( \text{cad} \) and \( \text{abra} \). The chunks can be used any number of times and in any order.

Write code for a function that takes a list of chunks along with a word, and returns a list of chunks that yield the word when concatenated. For the examples above, the result should be \[ ["abra"; "cad"; "abra"] \]. Exception \text{Fail} should be raised if no solution exists. [10 marks]

\textbf{Note}: You are given a function \text{delPrefix} for removing an initial part of a string. For example, \text{delPrefix} "abra" "abracadabra" returns "cadabra", but \text{delPrefix} "bra" "abracadabra" raises exception \text{Fail}.

All OCaml code must be explained clearly and should be free of needless complexity. Well-known utility functions may be assumed to be available.