Exercice 1

1.
```
let str_to_lst s =
   let n = String.length s in
   let l = ref [] in
   for i = n - 1 downto 0 do
      if s.[i] <> ' ' then
         l := s.[i] :: !l
   done;
   !l;;
```

2. On commence par transformer la chaîne de caractères en une liste. On écrit une fonction auxiliaire qui prend en argument une pile (initialement vide) et la liste en question et fait les opérations suivantes :
   - Si la liste `l` est vide, c'est qu'on a terminé les calculs. Normalement, la pile ne contient qu'une seule valeur, et c'est celle qu'on renvoie.
   - Sinon, le premier élément de la liste `l` peut être :
     * Un entier, auquel cas on le rajoute à la pile et on continue le traitement.
     * Un opérateur, auquel cas on l'applique aux deux derniers éléments de la pile (qu'on enlève), et on rajoute à la pile le résultat obtenu. On fera attention à l'ordre dans lequel on effectue l'opération (pour la division et soustraction qui ne sont pas commutatives).

```
let postfixe s =
   let l = str_to_lst s in
   let rec aux pile l = match pile, l with
      | [x], [] -> x
      | x :: y :: q, z :: r when z = '+' || z = '-' || z = '*' || z = '/'
         -> aux ((ops z y x) :: q) r
      | _, z :: r -> aux (chiffre z :: pile) r in
   aux [] l;;
```

Exercice 2

1. On procède comme suit :
   - On traite les cas de base (liste vide ou avec un seul élément) qui se décodent d'une seule façon (même la liste vide!!).
   - Si le premier chiffre est 0, on renvoie 0 (car ce n'est pas possible avec notre système).
   - Si les deux premiers chiffres forment un nombre entre 10 et 26, on peut soit interpréter le premier chiffre comme une lettre et compter le nombre de possibilités restantes, soit interpréter ce nombre comme une lettre et compter le nombre de possibilités restantes.
   - Sinon, le premier chiffre correspond nécessairement à une lettre, et on compte le nombre de possibilités restantes

```
let rec compte = function
   | [] -> 1
   | [m] -> 1
   | m :: n :: q when m = 0 -> 0
   | m :: n :: q when m = 1 || (m = 2 && n < 7)
            -> compte (n :: q) + compte q
         | m :: n :: q -> compte (n :: q);;
```

2. On traite tous les cas possibles :

```
let rec comptebis = function
   | [] -> 1
   | [-1] -> 9
   | [n] -> 1
   | m :: n :: q when m = -1 && n < 7
      -> 9 * comptebis (n :: q) + 2 * comptebis q
   | m :: n :: q when m = -1
      -> 9 * comptebis (n :: q) + comptebis q
   | m :: n :: q when m = 0 -> 0
   | m :: n :: q when m = 1 && n = -1
      -> comptebis (n :: q) + 10 * comptebis q
   | m :: n :: q when m = 2 && n = -1
      -> comptebis (n :: q) + 7 * comptebis q
   | m :: n :: q when m = 1 || (m = 2 && n < 7)
      -> comptebis (n :: q) + comptebis q
   | m :: n :: q -> comptebis (n :: q);;
```