Numéro d'anonymat :

Examen - Janvier 2011
LI101
Durée 2 heures

Aucun document ni machine électronique n'est permis à l'exception de la carte de référence de Scheme.

Le sujet comporte 17 pages. Ne pas désagrafer les feuilles.

Répondre sur la feuille même, dans les boîtes appropriées. La taille des boîtes suggère le nombre de lignes de la réponse attendue. Le barème apparaissant dans chaque boîte n'est donné qu'à titre indicatif. Le barème total est lui aussi donné à titre indicatif : 64 points.

La clarté des réponses et la présentation des programmes seront appréciées. Les questions peuvent être résolues de façon indépendante. Il est possible, voire utile, pour répondre à une question, d'utiliser les fonctions qui sont l'objet des questions précédentes.

Pour vous faire gagner du temps, il ne vous est pas systématiquement demandé, en plus de la définition, la spécification entière d'une fonction. Bien lire ce qui est demandé : seulement la définition ? la signature et la définition ? la spécification et la définition ? seulement la spécification ?

Lorsque la signature d'une fonction vous est demandée, vous veillerez à bien préciser, avec la signature, les éventuelles hypothèses sur les valeurs des arguments.

Première partie

Nombres triangulaires

Exercice 1

Question 1.1

On appelle nombre triangulaire de rang k le nombre égal à $\sum_{i=0}^{k} i$ c'est à dire $0+1+...+(k-1)+k$.

Donnez la signature et une définition **récursive** de la fonction `nb-triangulaire` qui étant donné un entier naturel `k` rend le nombre triangulaire de rang `k`. Exemples :

`(nb-triangulaire 0)` → `0`

`(nb-triangulaire 1)` → `1`

`(nb-triangulaire 2)` → `3`

`(nb-triangulaire 3)` → `6`

Numéro d'anonymat :

[2/64]

Question 1.2

Si l'on note u_k le nombre triangulaire de rang k on s'aperçoit que $u_k = u_{k-1} + k$.

Donnez la signature et une définition de la fonction `liste-nb-triang` qui étant donné un entier naturel `n` construit la liste contenant les nombres triangulaires u_n, u_{n-1}, ..., u_1, u_o, soit les `(n+1)` premiers nombres triangulaires en utilisant la relation ci-dessus. Notez bien que le premier nombre de la liste résultat est le nombre triangulaire de rang `n`.
Remarque : **l'efficacité de votre fonction sera prise en compte.**

Par exemple :

`(liste-nb-triang 0)` → `(0)`

`(liste-nb-triang 1)` → `(1 0)`

`(liste-nb-triang 2)` → `(3 1 0)`

`(liste-nb-triang 3)` → `(6 3 1 0)`

`(liste-nb-triang 4)` → `(10 6 3 1 0)`

`(liste-nb-triang 5)` → `(15 10 6 3 1 0)`

`(liste-nb-triang 6)` → `(21 15 10 6 3 1 0)`

[4/64]

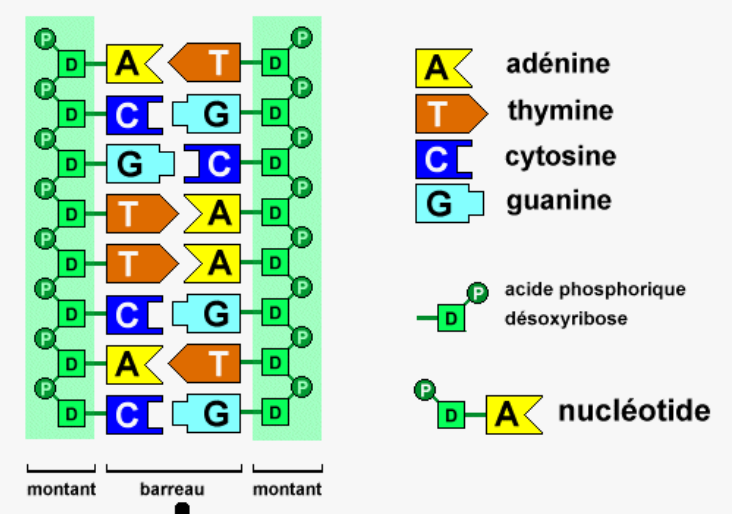
Numéro d'anonymat :

Deuxième partie

Autour de l'ADN

L'ADN est une molécule présente dans toutes les cellules vivantes, qui renferme l'ensemble des informations nécessaires au développement et au fonctionnement d'un organisme.

L'ADN est composé de deux brins d'ADN se faisant face, et formant une double hélice. Un brin d'ADN est composé d'une séquence de nucléotides. Chaque nucléotide comporte une base azotée. Il existe quatre bases azotées différentes : l'adénine (notée A), la thymine (notée T), la cytosine (notée C) et la guanine (notée G). Ce sont ces quatre bases azotées qui assurent la variabilité de la molécule d'ADN, ainsi que la complémentarité des deux brins : en face d'une adénine (A), il y a toujours une thymine (T) ; en face d'une cytosine (C), il y a toujours une guanine (G). On dit que les bases A et T sont complémentaires et que les bases G et C sont complémentaires. La figure ci-dessous montre un morceau d'ADN avec ses bases complémentaires.



Pour un brin d'ADN, on peut retrouver la séquence du brin complémentaire et reconstituer la double séquence de la double hélice :
-ATTGCCGTATGTATTGCGCT-
-TAACGGCATACATAACGCGA-

Dans la suite de cet exercice, on utilisera le type `Base` pour désigner une base azotée. Les bases azotées seront représentées par les chaînes de caractères "A", "T", "G" et "C".
Un brin d'ADN sera représenté par une liste de bases azotées.

Exercice 2

Question 2.1

Donnez la signature et une définition de la fonction sans paramètres `liste-base-compl` qui construit la liste d'associations qui associe à chaque base sa base complémentaire. On aura donc :
`(liste-base-compl)` → `(("A" "T") ("T" "A") ("G" "C") ("C" "G"))`

Numéro d'anonymat :

[2/64]

Question 2.2

Donnez la signature et une définition de la fonction `base-compl` qui étant donnée une base `b` rend la base complémentaire de `b`. Votre définition devra utiliser la fonction `liste-base-comp`. Par exemple :

```
(base-compl "A") → "T"
(base-compl "T") → "A"
(base-compl "G") → "C"
(base-compl "C") → "G"
```

[2/64]

Question 2.3

Donnez la signature et une définition de la fonction `brin-compl?` qui étant donnés deux brins d'ADN `B1` et `B2` teste si ces deux brins sont complémentaires. Par exemple :

```
(brin-compl? (list) (list)) → #t
(brin-compl? (list) '("A" "T")) → #f
(brin-compl? '("G" "C") (list)) → #f
(brin-compl? '("G") '("C")) → #t
(brin-compl? '("A" "T" "G" "C") '("A" "A" "C" "G")) → #f
(brin-compl? '("A" "T" "G" "C") '("T" "A" "C" "G")) → #t
```

Numéro d'anonymat :

[4/64]

Question 2.4

Donnez une définition de la fonction `brin-compl` qui étant donné un brin d'ADN `B` rend le brin d'ADN complémentaire correspondant. Par exemple :

```
(brin-compl '()) → ()
(brin-compl '("A" "A" "A" "T" "T" "T" )) → ("T" "T" "T" "A" "A" "A")
(brin-compl '("A" "T" "G" "C")) → ("T" "A" "C" "G")
(brin-compl '("C" "C")) → ("G" "G")
(brin-compl '("G")) → ("C")
(brin-compl '("A" "T" "A" "T" "A" "T" )) → ("T" "A" "T" "A" "T" "A")
```

[2/64]

Question 2.5

On dit que le brin `B1` est préfixe du brin `B2` si `B2` commence par la même séquence de bases que celle contenue dans `B1`. Donnez la signature et une définition de la fonction `brin-prefixe?` qui étant donnés deux brins d'ADN `B1` et `B2` teste si `B1` est préfixe de `B2`. Exemples :

```
(brin-prefixe? '() '("A" "T")) → #t
```

Numéro d'anonymat :

```
(brin-prefixe? '("A") '("A" "T")) → #t
(brin-prefixe? '("A" "T") '("A" "T")) → #t
(brin-prefixe? '("A" "T" "G") '("A" "T")) → #f
(brin-prefixe? '("A") '()) → #f
(brin-prefixe? '("C") '("A" "T")) → #f
```

[4/64]

Question 2.6

On dit qu'un brin `B1` est une sous-séquence du brin `B2` si la séquence des bases de `B1` apparaît dans `B2`. Donnez la signature et une définition de la fonction **`sous-brin?`** qui étant donnés deux brins d'ADN `B1` et `B2` teste si la séquence représentée par `B1` est une sous-séquence du brin `B2`. Par exemple :

```
(sous-brin? '() '()) → #t
(sous-brin? '("A" "T") '()) → #f
(sous-brin? '() '("A" "T")) → #t
(sous-brin? '("A" "G") '("T" "C" "A" "A" "G")) → #t
(sous-brin? '("A" "G") '("C" "A" "A" "G")) → #t
(sous-brin? '("A" "G") '("A" "A" "G")) → #t
(sous-brin? '("A" "G") '("A" "G")) → #t
(sous-brin? '("A" "G") '("G")) → #f
(sous-brin? '("T") '("T" "C" "A" "A" "G")) → #t
(sous-brin? '("T" "G") '("T" "C" "A" "A" "G")) → #f
```

Remarque : il est conseillé d'utiliser la fonction **`brin-prefixe?`** pour répondre à cette question.

Numéro d'anonymat :

[5/64]

Question 2.7

Donnez la signature et une définition de la fonction `nb-occ-brin` qui étant donnés deux brins `B1` et `B2` rend le nombre d'occurrences du brin `B1` dans le brin `B2`, c'est-à-dire le nombre que fois que la séquence des bases de `B1` apparaît dans `B2`. On supposera que `B1` n'est pas un brin vide. Par exemple :

```
(nb-occ-brin '("A" "G") '()) → 0
(nb-occ-brin '("A" "G") '("G" "C")) → 0
(nb-occ-brin '("A" "G") '("A" "G" "A" "G")) → 2
(nb-occ-brin '("A" "G") '("G" "A" "G")) → 1
(nb-occ-brin '("A" "G") '("A" "G")) → 1
(nb-occ-brin '("A" "A") '("A" "A" "A" "A")) → 3
(nb-occ-brin '("C" "T") '("A" "A" "A" "A")) → 0
```

Remarque : il est conseillé d'utiliser la fonction `brin-prefixe?` pour répondre à cette question.

[5/64]

Numéro d'anonymat :

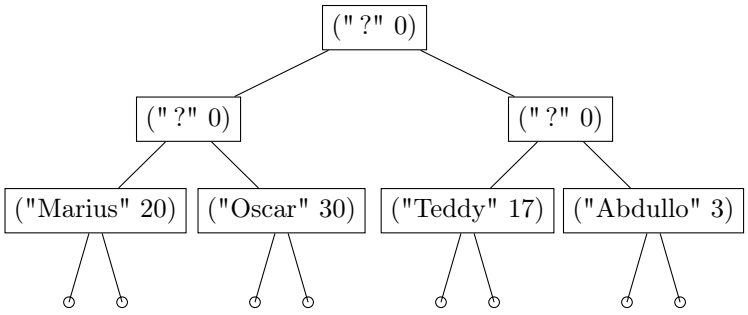
Troisième partie

Arbre de tournoi sportif

Exercice 3

On cherche ici à représenter l'évolution d'un tournoi de sport. Afin de connaître les différents affrontements entre les participants, on décide d'utiliser un arbre binaire. Chaque participant possède un nom et un nombre de points d'endurance au départ. Ces informations sont placées au début du tournoi au niveau des feuilles de l'arbre, les autres nœuds possédant l'étiquette par défaut `("?" 0)`.

Soit `ABTournoi` l'arbre de tournoi suivant :

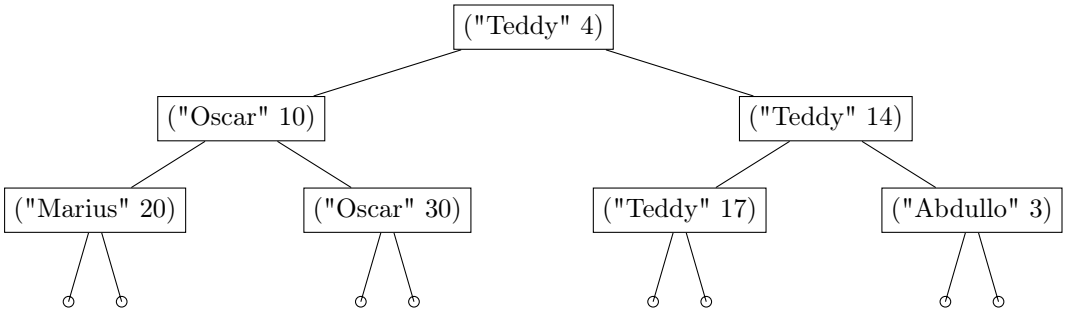


Les affrontements entre les participants ont lieu en partant des feuilles et en remontant vers la racine. Le vainqueur du tournoi est alors le participant situé à la racine de l'arbre. Lorsque deux participants s'affrontent, celui ayant le plus d'endurance est déclaré vainqueur mais il perd durant cet affrontement le nombre de points correspondant à l'endurance de son adversaire. En cas d'égalité, on choisira arbitrairement le participant de gauche.

Par exemple, dans l'arbre de tournoi `ABTournoi` précédent, l'affrontement entre Marius et Oscar verra la victoire d'Oscar mais celui-ci n'aura plus que 10 points d'endurance $(30 - 20)$ au tour suivant.

On dira qu'un arbre de tournoi est rempli lorsque tous les affrontements ont été calculés, autrement dit lorsqu'il n'y a plus d'étiquette `("?" 0)` dans l'arbre de tournoi.

Par exemple, l'arbre de tournoi rempli correspondant à `ABTournoi` sera :



Dans tout ce qui suit, on utilisera le type `Tournoi` ≡ `ArbreBinaire[Couple[string nat]]` pour désigner le type d'un tel arbre de tournoi. De plus, les arbres de type `Tournoi` sont des arbres binaires ***stricts*** : chaque nœud est soit une feuille, soit un nœud qui a exactement deux fils non vides.

Numéro d'anonymat :

Par ailleurs, notez bien que dans les questions de cet exercice **les arbres de tournoi paramètres des fonctions sont toujours, en plus d'être stricts par définition, des arbres binaires non vides**.

Question 3.1

Donnez la signature et une définition des deux fonctions accesseurs `nom` et `point` qui permettent d'accéder respectivement au nom et aux points d'endurance du participant situé à la racine d'un arbre de type `Tournoi` non vide.

`(nom (ab-noeud (list "Pierre" 10) (ab-vide) (ab-vide)))` → `"Pierre"`

`(point (ab-noeud (list "Pierre" 10) (ab-vide) (ab-vide)))` → `10`

`(nom ABTournoi)` → `"?"`

`(point ABTournoi)` → `0`

[1.5/64]

Question 3.2

Donnez une définition du prédicat `ab-feuille?: ArbreBinaire[alpha] -> bool` qui teste si un arbre binaire non vide est une feuille.

`(ab-feuille? (ab-noeud (list "Pierre" 10) (ab-vide) (ab-vide)))` → `#t`

`(ab-feuille? ABTournoi)` → `#f`

[2/64]

Numéro d'anonymat :

Question 3.3

Donnez la signature et une définition de la fonction `liste-participants` qui, étant donné un arbre de tournoi non vide, renvoie la liste des noms des participants du tournoi.

`(liste-participants (ab-noeud (list "Pierre" 10) (ab-vide) (ab-vide)))` → `("Pierre")`

`(liste-participants ABTournoi)` → `("Marius" "Oscar" "Teddy" "Abdullo")`

[4/64]

Question 3.4

Donnez une définition de la fonction `etiquette-vainqueur` dont la spécification est la suivante :

```
;;; etiquette-vainqueur: string * nat * string * nat -> COUPLE[string nat]
;;; (etiquette-vainqueur n1 p1 n2 p2) renvoie le couple (n p) dans lequel n
;;; est le nom du vainqueur de l'affrontement entre les participants n1 et n2,
;;; possédant respectivement p1 et p2 points, et p est le nombre de points restant
```

On rappelle qu'en cas d'égalité, c'est `n1` qui est arbitrairement déclaré vainqueur.

`(etiquette-vainqueur "Pierre" 10 "Marc" 2)` → `("Pierre" 8)`

`(etiquette-vainqueur "Pierre" 10 "Marc" 30)` → `("Marc" 20)`

`(etiquette-vainqueur "Pierre" 10 "Marc" 10)` → `("Pierre" 0)`

Numéro d'anonymat :

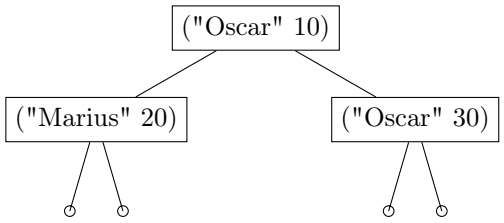
[3/64]

Question 3.5

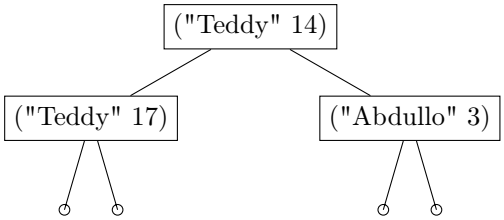
Donnez une définition de la fonction `noeud-tournoi : Tournoi * Tournoi -> Tournoi` qui, étant donnés deux tournois `T1` et `T2` non vides et remplis, renvoie l'arbre dont la racine est étiquetée par le vainqueur de l'affrontement entre les vainqueurs de `T1` et de `T2` et les sous-arbres gauche et droit sont `T1` et `T2`. Vous pourrez pour cela vous servir de la fonction précédente.

Par exemple :

```
(noeud-tournoi (ab-noeud (list "Marius" 20) (ab-vide) (ab-vide))
               (ab-noeud (list "Oscar" 30) (ab-vide) (ab-vide))) →
```




```
(noeud-tournoi (ab-noeud (list "Teddy" 17) (ab-vide) (ab-vide))
               (ab-noeud (list "Abdullo" 3) (ab-vide) (ab-vide))) →
```



Numéro d'anonymat :

[2/64]

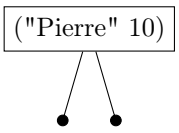
Question 3.6

Donnez une définition de la fonction `tournoi-rempli: Tournoi -> Tournoi` qui, étant donné un arbre de tournoi non vide, renvoie l'arbre de tournoi rempli, c'est à dire dans lequel les étiquettes `("?" 0)` sont remplacées par les noms et points d'endurance restants du vainqueur du tour précédent.

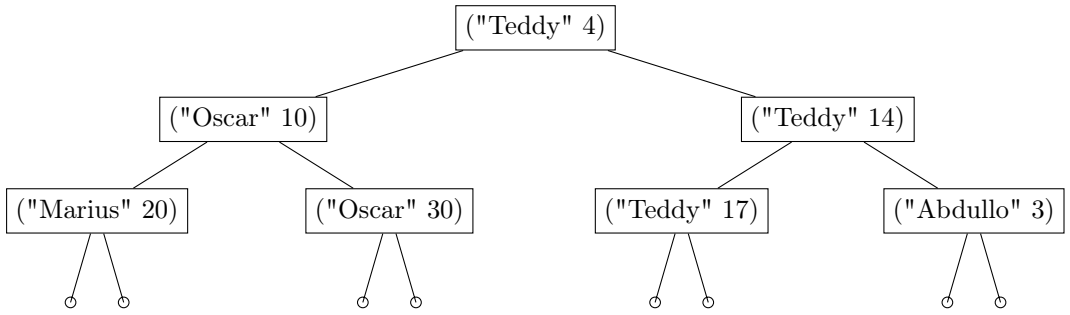
Pour ce faire, vous appliquerez la procédure suivante pour remplir un arbre de tournoi `T` :
– Si `T` est une feuille, le résultat du remplissage est `T` lui-même.
– Sinon, il faut :

1. remplir les sous-arbres gauche et droit de `T`,
2. renvoyer le nouvel arbre correspondant au résultat de l'affrontement entre les vainqueurs des sous-arbres gauches et droits

`(tournoi-rempli (ab-noeud (list "Pierre" 10) (ab-vide) (ab-vide)))` →

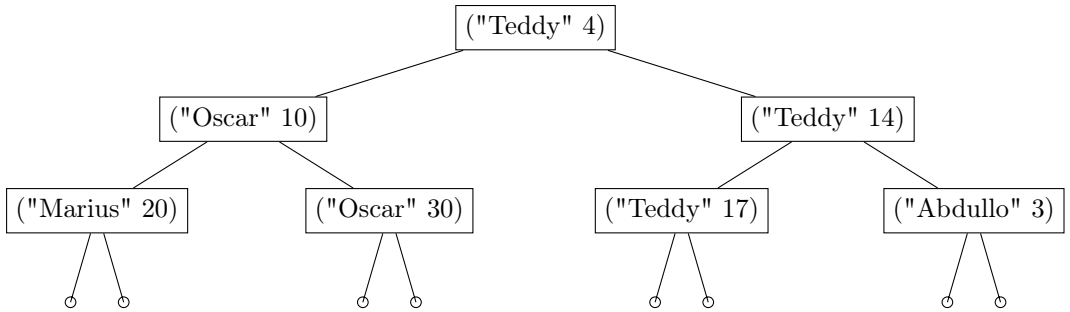


`(tournoi-rempli ABTournoi)` →



Remarque : le remplissage d'un arbre déjà rempli recalcule le même arbre de tournoi. Par exemple :
`(tournoi-rempli (tournoi-rempli ABTournoi))` →

Numéro d'anonymat :



[3/64]

Question 3.7

Donnez la signature et une définition de la fonction `nom-vainqueur-tournoi` qui, étant donné un arbre de tournoi non vide, renvoie le nom du vainqueur du tournoi une fois les affrontements réalisés.

`(nom-vainqueur-tournoi (ab-noeud (list "Pierre" 10) (ab-vide) (ab-vide)))` → `"Pierre"`

`(nom-vainqueur-tournoi ABTournoi)` → `"Teddy"`

[1.5/64]

Numéro d'anonymat :

Quatrième partie

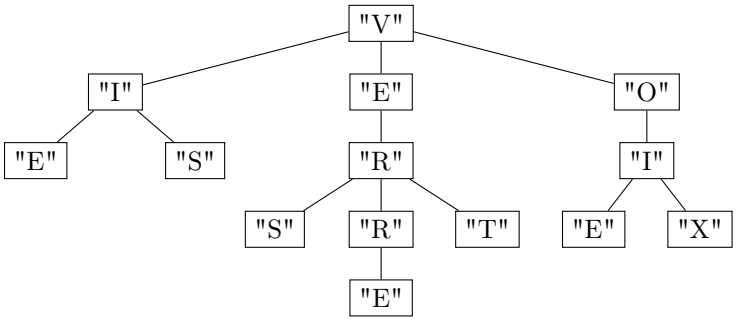
Arbre général représentant un glossaire

Exercice 4

Le but de cet exercice est de proposer des fonctions permettant la manipulation d'un glossaire de mots. On choisit ici de représenter les mots par des listes de lettres. Ainsi, le mot "vie" sera représenté par la liste `("v" "i" "e")`.

Une représentation efficace d'un glossaire de mots consiste à utiliser un arbre général dont chaque nœud est étiqueté par une lettre. Un mot du glossaire est alors défini comme la succession des lettres rencontrées dans un chemin partant de la racine et arrivant à l'une des feuilles.

Soit le glossaire `GlosV` suivant :



Ce glossaire contient donc les mots "vie", "vis", "vers", "verre", "vert", "voie" et "voix". Notez que le glossaire ne contient pas le mot "ver" car il n'existe pas de chemin étiqueté par les lettres composant ce mot et arrivant à une feuille.

Dans ce qui suit, on utilisera le type `Lettre` pour désigner une chaîne de caractères de longueur 1, le type `Mot` ≡ `LISTE[Lettre]` pour désigner le type des mots et `Glossaire` ≡ `ArbreGeneral[Lettre]` pour désigner celui des glossaires.

Question 4.1

Donnez une définition de la fonction `longueur-max: Glossaire -> nat` qui calcule la longueur du mot le plus long dans un glossaire.

`(longueur-max GlosV)` → `5`

Numéro d'anonymat :

[4/64]

Question 4.2

Puisqu'un mot est représenté par un chemin dans le glossaire, il est facile de voir qu'il y a autant de mots dans le glossaire que de feuilles dans l'arbre général qui le représente.

Donnez une définition de la fonction `nb-mots: Glossaire -> nat` qui calcule le nombre de mots contenus dans un glossaire :

`(nb-mots GlosV)` $\rightarrow$ `7`

[4/64]

Numéro d'anonymat :

Question 4.3

Donnez la signature et une définition de la fonction `nb-occ-lettre` qui, étant donnés une lettre `l` et un glossaire `G`, renvoie le nombre d'occurrences de la lettre `l` dans `G`.

`(nb-occ-lettre "A" GlosV )` → `0`

`(nb-occ-lettre "E" GlosV )` → `4`

[4/64]

Question 4.4

Donnez la signature et une définition de la fonction `existe-mot?` qui teste si un mot appartient à un glossaire.

`(existe-mot? '() GlosV )` → `#f`

`(existe-mot? '("A" "V" "I" "O" "N") GlosV )` → `#f`

`(existe-mot? '("V" "E" "R") GlosV )` → `#f`

`(existe-mot? '("V" "E" "R" "S") GlosV )` → `#t`

`(existe-mot? '("V" "E" "R" "S" "E" "R") GlosV )` → `#f`

Numéro d'anonymat :

[5/64]