Un peu de bases de données : le répertoire

Note 1 : lorsque les paramètres et les retours des fonctions sont décrits, il est d'usage d'indiquer le type attendu en spécifiant le prédicat de type vérifié par le paramètres ou le retour de la fonction, exemple : reverse prend un paramètre L (`list?`) et renvoie la liste inversée (`list?`) ou bien, présenté comme dans la doc de racket :

(reverse lst) → list?
  lst : list?

Note 2 : Les fonctions demandées doivent etre purement fonctionnelles (sans affectation), ce qui ne veut pas dire qu'on ne s'autorise pas à les utiliser avec des affectations (set!) :

```
> (define repertoire '())
> repertoire
'()
> (set!  repertoire (ajout-r repertoire "AA" "aa"))
> repertoire
'((0 "AA" "aa"))
> (set!  repertoire (ajout-r repertoire "aa" "zz"))
> repertoire
'((1 "aa" "zz") (0 "AA" "aa"))
```

Un **répertoire** est une liste de personnes. Une **personne** est une liste contenant 3 éléments, un **identifiant** numérique, une chaîne de caractères pour le **nom** et une autre pour le **prénom**.

Ecrire les fonctions permettant :

a- d'ajouter une personne dans le répertoire (l'identifiant numérique sera l'identifiant du dernier ajout + 1)
Cette fonction prend 3 paramètres : un répertoire (`list?`), le nom (`string?`) et le prénom (`string?`) et renvoie un répertoire (`list?`)

```
> (ajout-r '() "dupond" "jean")
'((0 "dupond" "jean"))
> (ajout-r (ajout-r '() "AA" "aa") "BB" "bb")
'((1 "BB" "bb") (0 "AA" "aa"))
```

b- d'afficher le répertoire
Cette fonction prend un paramètre : un répertoire (`list?`) et ne renvoie rien

```
> (affiche-r  (ajout-r (ajout-r '() "AA" "aa") "BB" "bb"))
1 : BB -- bb
0 : AA -- aa
```

c- de comparer deux personnes
Cette fonction prend deux paramètres p1(`list?`) et p2 (`list?`) et renvoie un booléen, vrai si p1 est avant p2 dans l'ordre alphabétique sur les noms et sur les prénoms si les noms sont égaux, faux sinon (comparaison sur les chaînes de caractères : string<?).

```
>  (define rep (ajout-r (ajout-r '() "AA" "aa") "AA" "bb"))
> (compare-p (car rep) (cadr rep))
#t
```

d- de trier un répertoire

```
> (define rep (ajout-r (ajout-r (ajout-r '() "AA" "aa") "AA" "bb") "BB" "bb"))
> (affiche-r rep)
2 : BB -- bb
1 : AA -- bb
0 : AA -- aa
> (affiche-r (tri-ins rep compare-p))
0 : AA -- aa
1 : AA -- bb
2 : BB -- bb
```

d- de supprimer une personne (par son identifiant) dans le répertoire

```
> (define rep (ajout-r (ajout-r (ajout-r '() "AA" "aa") "AA" "bb") "BB" "bb"))
> (affiche-r rep)
2 : BB -- bb
1 : AA -- bb
0 : AA -- aa
> (affiche-r (supprime-r rep 1))
2 : BB -- bb
0 : AA -- aa
```

e- de construire un répertoire avec toutes les personnes dont le nom commence par une lettre donnée en paramètre

La fonction qui permet de trouver le kième caractère d'une chaîne est :(string-ref string k)

```
> (define rep (ajout-r (ajout-r (ajout-r '() "AA" "aa") "AA" "bb") "BB" "bb"))
> (affiche-r rep)
2 : BB -- bb
1 : AA -- bb
0 : AA -- aa
> (affiche-r (extract rep #\A))
1 : AA -- bb
0 : AA -- aa
```

f- d'écrire un répertoire dans le fichier repertoire.txt

La fonction qui permet d'écrire dans un fichier (read) nécessite d'avoir ouvert le fichier en écriture, ce qui se fait grâce à la fonction :open-output-file, qui renvoit un descritpeur de fichier.
ex: (let ((out (open-output-file "repertoire.txt" #:exists 'replace))) (write rep out) (close-output-port out))
le #:exists 'replace sert à préciser ce qu'on veut faire si le fichier existe déjà, ici le remplacer
note : regardez comment le répertoire est écrit dans le fichier.
note : il sera peut-être utile de trier en décroissant suivant les identifiants le répertoire avant de l'écrire, pour être sûr que le plus grand identifiant est au début du répertoire

```
> (save-r (ajout-r (ajout-r (ajout-r '() "AA" "aa") "AA" "bb") "BB" "bb"))
```

g- de lire un répertoire dans le fichier repertoire.txt

Même chose que pour l'écriture, il faut avant de lire, ouvrir le fichier en lecture, par la fonction open-input-file. Ex : (let ((in (open-input-file "repertoire.txt"))) (read in)). En théorie il faudrait fermer ensuite le fichier, mais c'est pas évident à faire … je vous laisse chercher :-), sinon, on le laissera ouvert.

```
> (affiche-r (load-r))
2 : BB -- bb
1 : AA -- bb
0 : AA -- aa
```

Il faut maintenant assembler tout cela pour faire une petite application, avec un menu style DOS.

Proposition d'exercices supplémentaires

Si vous vous ennuyez, et si vous vous sentez d'humeur aventureuse, vous pouvez toujours tenter de faire une interface graphique pour ce répertoire. Attention, c'est à vos risques et périls, et l'agence niera avoir connaissance de vos activités …

Lien vers la bibliothèque graphique de racket:
http://docs.racket-lang.org/draw/index.html
et la bibliothèque des interfaces graphiques :
http://docs.racket-lang.org/gui/index.html