TD-TP 14 proba version 2

A faire en 2 heures (TD et TP) et … après le TP 14 première partie.

Concepts : type réel "point fixe" (**`delta`**) comme composant d'un type T.A.D. `T_Proba`, clause de représentation).

Conseils : Partir impérativement de votre travail fait au TP version 1 sur les probabilités. C'est l'hypothèse de travail de ce guide. Maîtrisez donc le copier-coller.

Rappel : On sait qu'une valeur `Pr(A)` est comprise entre `0.0 et 1.0`. Aussi doit-on restreindre les réels modélisant ces valeurs dans cet intervalle; c'est ce qui est fait dans les spécifications du paquetage `P_Proba1` (avec **`range`**). Mais il est **sûrement inutile** de travailler avec des réels "point-flottant" puisqu'avec les probabilités la précision n'a pas à être très pointue donc l'écart absolu (à fixer suivant les problèmes) s'impose. Il convient donc de travailler en réels "point-fixe" avec par exemple une précision absolue de 0.001 (ou plus ou moins). Toutes ces déclarations se feront en partie privée du paquetage.

1°) Ecrivez les spécifications du paquetage P_Proba2. Peu de changement (sauf en **`private`**).

```
package P_Proba2 is
      type T_Proba is private;
      function "+" (Un,Deux : in T_Proba) return T_Proba;
      function "*" (Un,Deux : in T_Proba) return T_Proba;
      function "-" (Un : in T_Proba) return T_Proba;
      procedure Get (Un : out T_Proba);
      procedure Put (Un : in  T_Proba);
      procedure Get (Fic : in File_type ; Un : out T_Proba) ;
      procedure Put (Fic : in File_Type; Un : in T_Proba);
      Certain : constant T_Proba; -- nouveau !
      Impossi : constant T_Proba;
private

end P_Proba2;
```

Intervenez ici ! Réfléchissez !

Par rapport au TP précédent vous changez le **`private`** (plus aucun `Float` S.V.P.! ne doit apparaître). Ajoutez y aussi la définition des deux constantes `Certain` et `Impossi` respectivement les probabilités de l'évènement toujours réalisé et celui de l'évènement impossible.

A une ligne près : **`with`** `P_Proba2` ; **`use`** `P_Proba2`; le programme de test est le même que dans le premier TP. Ce n'est pas un miracle c'est la puissance et la conception modulaire de Ada !

Vous devez réécrire le body avec toujours les mêmes problèmes de **surcharge qui ne doit pas être récursive.** Attention il y a un piège (il faut toujours convertir **mais sans `Float`**). Et bien sûr on n'instancie plus `Float_Io` mais `Fixed_Io` pour créer des `GET` et des `PUT` de qualité.

Quelques problèmes à régler :

1°) la traduction de `X "+" Y` par quelque chose du genre de :
`T_Fixe(X) + T_Fixe(Y) - T_Fixe(X) * T_Fixe(Y)` (correcte a priori) n'est pas sans surprise ! Voyons cela :

L'expression `T_Fixe(X) + T_Fixe(Y)` du calcul **partiel**, ci-dessus, de `X "+" Y` risque de sortir de l'intervalle `0.0..1.0` et ceci avant la soustraction qui rétablira (mais peut être trop tard) l'équilibre pour ramener le résultat final dans l'intervalle !! Prenez par exemple `0.8` et `0.4` comme jeu de premier essai.

2°) **Autre piège possible** : le produit de deux réels point-fixe ! Voyez votre cours sur les **opérateurs point-fixe**.

Quelques aides :

Pour résoudre le 1°) on peut utiliser la relation : (à vérifier!) :

a + b - a * b = 1 - ((1-a) * (1-b)). Ce savant calcul est vrai quels que soient les opérandes vérifiez le. La réécriture de cette formule avec cette fois les surcharges donne alors :

`X "+" Y = "-"(("-"(X)) "*" ("-"(Y)))` joli ! non ?

On peut donc écrire `"+"` en utilisant seulement `"*" et "-"` !! et l'intégrité du calcul est conservé. Fabuleux ! Non ?

Commencez donc par réécrire `"*"`, `"-"`. Puis écrivez `"+"` seulement à l'aide de ces deux là.

Testez par exemple avec a = 0.1 et b = 0.01 **en faisant varier le `delta`**.
Résultats attendus (évidemment) :

a ″+″ b = 0.109 et a ″*″ b = 0.001. Pas si sûr !
Et avec a = 0.8 et b = 0.4 a-t-on a ″*″ b = 0.32 ? A voir !

La solution est encore dans le cours avec la notion de **clause de représentation** (toujours le cours).

Pour éviter de nombreuses re-compilations quand on fait varier le **`delta`** on peut plaquer là dessus la généricité (de toute façon il faut toujours **penser généricité** quand on développe un paquetage).

```
generic
.........;
package P_Proba3 is
      type T_Proba is private;
      function "+" (Un,Deux : in T_Proba) return T_Proba;
      function "*" (Un,Deux : in T_Proba) return T_Proba;
      function "-" (Un : in T_Proba)      return T_Proba;
      procedure Get (Un : out T_Proba);
      procedure Put (Un : in  T_Proba);
      procedure Get (Fich : in File_Type, Un : out T_Proba);
      procedure Put (fich : in File_Type, Un : in  T_Proba);
      Certain : constant T_Proba;
      Impossi : constant T_Proba;
private
.....
-- Certain et Impossi décrits ici car T_Proba est privé
      Certain :.......;
      Impossi :.......;

end P_Proba3;
```

Le programme de test de cette troisième version est, là encore, peu différent. Le corps est totalement identique (lecture des valeurs, utilisation des méthodes définies dans le paquetage "'+","*" et "-", écriture des résultats). Ce qui change un peu c'est, après l'évocation du paquetage générique `P_Proba3`, l'instanciation avec un ou des types "point-fixe". On en profitera au moins ici pour fournir des calculs justes grâce à la clause de représentation.

Des valeurs pour des **tests supplémentaires** vous seront fournies au moment du TP. A ajouter dans vos fichiers de données.

Détail du TP (deuxième partie) :

- Réalisez le body de `P_Proba2` (voyez aussi les spécifications) en copiant `P_Proba1` sur `P_Proba2` et en utilisant l'option de remplacement de votre éditeur (c'est presque facile !).
- Testez avec `Pp_Prob2` (que vous fabriquez avec `Pp_Prob1` : juste le **`with`** à changer !). Faites des essais sans la clause de représentation puis recommencer avec celle ci ! Les choses devraient s'arranger !
- Lancez vous dans la troisième version (il n'y a presque rien à faire sinon de faire encore du copier-coller). Testez encore !
- En résumé vous aurez réalisé, à la fin, 3 programmes de test (pp_probX.adb)et 3 body de paquetage (p_probaX.adb).

Fin de la séance de **4 heures pour tout ce qui concerne les Probabilités**.