TP 5 – Correction et exercice récapitulatif

Dans ce TP, nous allons revenir et corriger le contrôle continu du vendredi 25 octobre. La correction suis l'énoncé version 1 du contrôle continu, les types d'erreurs et les explications étant les mêmes, il ne sert à rien de faire plusieurs corrections.

Pour chaque question, vous trouverez un exemple de code fonctionnel et l'explication adéquate. Bien sur, il y a plusieurs programmes fonctionnels et il existe plusieurs explications correctes, la correction vous montre une possibilité de réponse.

Question 1 : Variables...

Mon programme doit afficher "*Bonjour je suis un programme très intelligent*". Cependant il ne fonctionne pas, expliquez pourquoi et modifiez le en suivant les instructions:

- le contenu des variables ne doit pas être **changé**
- aucunes instructions déjà présentes dans le "local" ne peuvent être **modifiés**
- il est **interdit** de rajouter un Browse dans le local
- il faut **utiliser** le contenu de Texte3

Correction Programme :

```
declare
Texte1 = 'bonjour'
Texte2 = 'je suis un programme'
Texte4
local
Texte3 = 'tres intelligent'
in
Texte4 = Texte3
{Browse Texte1}
{Browse Texte2}
end
{Browse Texte4}
```

Correction Explication :

Lorsque l'on veut créer des variables globales dans un programme il faut utiliser l'instruction *declare*. Ensuite il faut donner les **identificateurs** de chaque variable globale, ces identificateurs commencent par une **majuscule**. Une fois nos variables déclarées ont peut la lier à une valeur, ici une chaîne de caractère.

Cependant la variable *Texte3* est déclaré dans un local, la **portée** de cette variable est donc limité au bloque d'instructions *local in end*.

Pour corriger l'exercice tout en respectant les consignes, on va donc créer une nouvelle variable globale à laquelle on va liée la valeur présente dans *Texte3* dans le *in* du bloc d'instruction *local.* Ainsi on ne modifie pas les portées des variables, on utilise le contenu de la variable *Texte3* et on ne modifie aucune instruction du *local*.

On affiche ensuite la variable *Texte4* afin de faire apparaître le texte en entier.

Question 2 : Variable... le retour

Mon programme doit calculer le carré d'un nombre. Cependant il ne fonctionne pas, expliquez pourquoi et modifiez le en suivant les instructions:

- il est **interdi**t de rajouter un Browse
- il ne faut pas **modifier la portée des variables** déjà présentes

Correction Programme :

```
declare
Square
local
   fun{Enix A}
         A*A
   end
in
   Square = Enix
end
{Browse {Square 5}}
```

Correction Explication :
La première erreur de se programme vient de l'**appel de fonction**. Lorsque l'on appelle une fonction on peut lui passer des **arguments**. Il faut donc qu'il y ait une correspondance entre les arguments passés à la fonction et les arguments demandés par la fonction. Ici, nous voulions calculer le carré d'un nombre, il fallait donc enlever un des arguments envoyé à la fonction.
Le deuxième problème venait du fait que nous appelions *Square* plutôt que *Enix*. Afin de respecter les instructions, il fallait utiliser le lien **Variable – Variable** qui va permettre de lier *Square* et *Enix* ensemble tout en respectant **la portée des variables.** En effectuant *Square = Enix* dans le *in* du *local* nous permettons l'appel de la fonction *Enix* à travers la variable *Square* (lien variable – variable).

Question 3 : Le métro Parisien...

Expliquer le comportement de ce programme en utilisant le vocabulaire adapté.

Correction Explication :
Dans ce programme, nous essayons d'afficher une variable *V2* qui n'a pas de valeur, qui n'est pas **liée**. Le programme va donc mettre la file d'exécution relative à *V2* « en pause » et attendre qu'un autre file d'exécution lie la variable *V2*. On voit que le programme affiche *'valeur'* au bout de 2 secondes d'attente. Tout simplement car on utilise le lien **variable – variable** pour **lier** *V2* à *V1*.
Le comportement qui permet à un file d'exécution d'attendre s'appelle le **dataflow** ou encore « flux de données ».

Question 4 : Les listes...

Mon programme doit vérifier qu'un individu symbolisé par une lettre a le droit d'entrer dans un bâtiment. Pour avoir ce droit, il doit faire parti d'une liste d'employés (Nom) et il doit avoir explicitement cette autorisation (Mod avec 0 comme interdiction et 1 pour autorisation). Avec l'exemple du programme ci-dessous, 'j' a les autorisations pour entrer tandis que 'd' et 'a' ne les ont pas.
Cependant il ne fonctionne pas, **expliquez** pourquoi et **modifiez** le pour le rendre fonctionnel.

Correction Programme :

```
declare
Nom = [d e h j k l n o p q r s t u v w y]
Mod = [0 0 1 1 1 0 0 0 1 0 1 0 1 1 0 1 1]
fun{Droit Id Nom Mod}
   case Nom of H|T then
      if H==Id then
    case Mod of Hmod|Tmod then Hmod end
      else {Droit Id T Mod.2} end
   else 0
   end
end
{Browse {Droit j Nom Mod}}
{Browse {Droit d Nom Mod}}
{Browse {Droit a Nom Mod}}
```

Correction Explication :
La première erreur venait du fait qu'une instruction *end était* en trop. Un bloc d'instruction se compose de deux (voir trois) parties : *if* (on débute le bloque), *then* (on effectue des instructions), *end* (on ferme le bloque).
La seconde erreur venait du **pattern matching**. Le pattern matching ou **correspondance de formes** est un manière de regarder les composants d'une liste. Le pattern matching utilise l'instruction *case* et permet d'extraire en même temps la tête et la queue d'une liste. Ici, la correspondance des formes étaient mélanger avec **les opérateurs de sélections** *.1* et *.2* qui permettent aussi de faire ressortir la tête et la queue de la liste.

Question 5 : Addition d'une table de multiplication...

Mon programme doit faire le carré de la somme des termes d'une table de multiplication (par exemple avec les 4 premiers termes de la table de 2 : 1*2 + 2*2 + 3*2 + 4*2 = 20 puis 20 * 20 = 400). Le programme proposé ci-dessous est sensé effectuer ce calcul, cependant il ne fonctionne pas, **expliquer** pourquoi et **modifiez** le pour le rendre fonctionnel.

Correction Programme :

```
declare
fun{Multiplication N C}
   local
         Resultat = {NewCell 0}
   in
         for I in 1..N do Resultat := @Resultat + (I*C) end
         Resultat := @Resultat * @Resultat
         @Resultat
   end
end
{Browse {Multiplication 4 2}}
```

Correction Explication :
De part la **portée des variables**, il nous faut déclarer notre variable Re*sultat* dans un local. Ensuite, en regardant le fonctionnement du programme, nous voyons que la variable Re*sultat* va devoir changer. Or, une variable ne peut avoir qu'une **affectation unique**. Afin de pouvoir calculer notre addition de multiplication, il va falloir donner à notre fonction une mémoire. Ce genre de mémoire est appelé **l'état explicite**. Le moyen le plus simple d'utiliser l'état explicite est d'utiliser **les cellules** mémoires. Il faut donc bien faire la différence entre **cellule** qui est une sorte de boite dans laquelle on peut insérer du contenu arbitrairement et les **variables** qui sont des raccourcies vers des valeurs.

Exercice récapitulatif : Comptage du nombre d'occurence

Vous avez maintenant le niveau pour effectuer l'exercice suivant : programmer une fonction qui permet de compter le nombre d’occurrence d'un élément dans une liste.

Pour vous aider vous pouvez divisez votre programme en deux fonctions : la première va compter le nombre d'élément de la liste, la seconde va compter le nombre d’occurrence dans la liste. La première fonction est simple à réaliser et je ne vous donne donc aucune aide. Pour la seconde vous devrez utiliser (mais ce n'est pas une obligation) **l'état explicite** et **les opérateurs de séléctions**.