Correction de l'examen ADA

Exercice 1

A- Ecrire un paquetage (.ads et .adb) qui contient les 2 fonctions ci-dessous

package Chaine **is**

Longueurs_Differentes, Chaine_A_Remplacer_non_Presente : **exception**

function Indice_De(Chaine :string ;Dans :string) **return** natural;

function Remplacement(De :string ; Par :string ; Dans :string) **return** string;

end chaine;

package body Chaine **is**

// voir ci-dessous les differentes implementations

end Chaine;

1. Recherche d'une occurrence

On ne regarde pas tous les éléments de *Dans* : on s'arrête avant la fin quand il n'en reste pas assez pour que cela corresponde à *Chaine* c'est à dire à l'élément en position dernier de *Dans* – Dimension de Chaine + 1.
La tranche de *Dans* commençant en position I s'arrête en position I+Dimension de Chaine-1.

```
function Indice_De(Chaine :string ;Dans :string) return natural is
begin
   for I in (Dans'first..Dans'last-Chaine'length+1) loop
       if Dans(I..I+Chaine'length-1)=Chaine then
           return I ;
       end if ;
   end loop ;
   return 0 ;
end Indice_De
```

ou encore

```
function Indice_De(Chaine :string ;Dans :string) return natural is
   resu : Natural:=0;
   I: Positive:=Dans'first;
   trouve:Boolean:=false;
begin
   while not trouve and I<=Dans'last-Chaine'length+1 loop
       if Dans(I..I+Chaine'length-1)=Chaine then
```

```
            Trouve:=true ;
            Resu:=I;
        else
            I:=I+1;
        end if ;
    end loop ;
    return resu ;
end Indice_De ;
```

2. Remplacement d'une occurrence

function *Remplacement(De :string ; Par :string ; Dans :string)* ***return*** *string ;*
son rôle est de remplacer dans la chaîne *Dans* la première occurrence de la chaîne *De* par la chaîne *Par*.
On suppose pour simplifier que les chaînes *De* et *Par* ont la même longueur. Si ce n'est pas le cas ou si la chaîne *De* n'est pas présente dans la chaîne *Dans*, **le remplacement ne peut être effectué**.

Dans le corps de cette fonction, déclencher des exceptions lorsque les chaînes *De* et *Par* ne sont pas de la même longueur ou lorsque la chaîne *De* n'est pas présente dans la chaîne *Dans*. On utilisera 2 exceptions différentes.

```
function Remplacement(De :string ; Par :string ; Dans :string) return string is
    Indice : Natural := Indice_De(De, Dans) ;
begin
   if De'length /=Par'length then
      put_line (" longueurs différentes ");
      raise Longueurs_Differentes;
   end if;
    if Indice /= 0 then
       return  Dans(Dans'first..Indice-1)& Par & Dans(Indice+De'length..Dans'last) ;
    else
      put_line (" chaîne a remplacer non trouvee ");
      raise Chaine_A_Remplacer_non_Presente ;
    end if ;
end Remplacement ;
```

B- Quels sont les problèmes posés lors de l'utilisation de cette fonction

- si l'on veut que le remplacement soit effectué quelles que soient les longueurs des chaînes à remplacer et de remplacement
- si l'on veut remplacer toutes les occurrences de la chaîne à remplacer par la chaîne de remplacement.

Dans les 2 cas, il est difficile d'évaluer à l'avance quelle sera la longueur de la chaîne résultat Nouvelle. On est obligé alors de déclarer la chaîne résultat sans donner la dimension et en même temps de la dimensionner par un appel à la fonction remplacement.

```
declare
```

```
Nouvelle : String := Remplacement (Rech(1..Long_rech),
                                   Remp(1..Long_remp),
                                   Chaine(1..Long_chaine));
begin
```

Exercice 2

Une école possède un certain nombre de filières. Une filière est composée d'une liste de matière qui la compose et un tableau pour gérer les notes des étudiants par matières de cette filière. Ce tableau servira par exemple à calculer la moyenne annuelle de chaque étudiant de la filière et la moyenne de la filière. On suppose qu'un étudiant est décrit complètement par son nom, son prénom (maximum de 20 caractères) et son identificateur (un entier positif). Le nombre maximum d'étudiants par filière à gérer est de 40.

Elève / Matière	**Maths**	**Physique**	**Info**	**Anglais**	**Français**
125 Charlie Cedric	6.0	5.0	5.0	4.5	4.0
425 Boucher Didier	5.5	5.0	3.0	5.5	6.0
23 Hermann Marc	3.5	1.0	5.5	4.5	3.5

1. Définir proprement toutes les structures de données qui interviennent dans le problème (Pas de valeur numérique dans les types, que des constantes).
Note, Matière, Tableau de notes, Elève, Ligne "*Elève, Tableau de Notes*" et Filière

```
-- Nombre maximal de caractères pour le nom ou le prenom d'un élève
Max_longueur_Nom: constant Integer := 20;


-- Nombre maximal d'élèves
Max_Nombre_Eleves: constant Integer := 40;

-- Type pour représenter les notes des élèves
type T_Note is delta 0.1 range 1.0..6.0;

-- Type pour représenter les matières
type T_Matiere is (Maths, Physique, Info, Anglais, Francais);

-- Type pour representer le tableau des notes
type T_Tab_Notes is array(T_Matiere'First..T_Matiere'Last) of T_Note;

-- Type pour représenter un elève
type T_Eleve is record
      Identificateur : Positive;
      Nom            : String(1..Max_longueur_Nom);
      Prenom         : String(1..Max_longueur_Nom);
end record;

-- Type d'une ligne du tableau élève-matière
type T_Ligne is record
```

```
  Eleve     : T_Eleve;
  Tab_Notes : T_Tab_Notes;
end record;

-- Classe tableau de lignes
type T_Classe is array(1..Max_Nombre_Elèves) of T_Ligne;

-- Type d'une Filière est une liste de matieres T_Listes_Matieres et un tableau --
pour gérer les notes des étudiants par matières de cette filière (T_Classe)
type Element ;
type Liste is access Element ;
type Element is record
     Nom_matiere : T_Matiere;
     Suivant : Liste ;
end record ;

type T_Filière is record
  Classe     : T_Classe;
  Liste_Matieres : Liste;
end record;
```

2. Ecrire une fonction qui vérifie si une matière donnée fait partie ou pas d'une filière

```
function Matiere_In_Filiere (Matiere: T_matiere; Filiere: T_Filiere) return Boolean is
    Reponse : Boolean := False;
    Aux : Liste := Filiere.Liste_Matieres;
begin  -- Matiere_In_Filiere
   -- Parcourir toute la classe
      while Aux /= null loop
            if (Matiere = Aux.Nom_matiere) then
                  Reponse=True;
                  exit,
            else
                  Aux := Aux.all.Suivant;
            end if;
      end loop;
      return Reponse;
end Matiere_In_Filiere;
```

3) En utilisant la fonction définie dans 2, écrire une fonction permettant de savoir si une matière **ne fait pas partie** en même temps dans deux filières différentes.

```
function Matiere_In_2_Filieres (Matièee: T_matiere; Filiere1, Filiere2: T_Filiere) return Boolean is
begin  -- Matiere_In_2_Filières
   return not (Matiere_In_Filiere (Matiere, Filiere1) and Matiere_In_Filiere (Matiere, Filiere2));
end Matiere_In_2_Filieres;
```