

1 Exercice 1

1.1 Indice_De

La fonction `Correspond` est une fonction auxiliaire utilisée par `Indice_De`. Elle prend en arguments deux chaînes (`Chaine` et `Dans`) qui sont les entrées de `Indice_De` et deux entiers (`Position_Chaine` et `Position_Dans`) qui correspondent aux positions de la recherche du motif dans le mot.

Elle renvoie vrai si `Chaine(Position_Chaine..Chaine'Last)` apparaît à la position `Position_Dans` dans `Dans`.

Exemple : `Correspond("tu",1,"voiture",4)` retourne vrai.

Cette fonction est réursive. Lorsque le $i^{\text{ème}}$ caractère de `Chaine` correspond au $n^{\text{ème}}$ caractère de `Dans`, on réitere jusqu'à ce que l'on arrive au bout de l'une des deux chaînes.

```
1 function Correspond (Chaine : String; Position_Chaine :  
    Natural;  
2                               Dans : String; Position_Dans : Natural)  
    return Boolean is  
3 begin  
4     if Position_Dans > (Dans'Last+1)  
5     then  
6         return(False);  
7     elsif Position_Chaine > Chaine'Last  
8     then  
9         return(True);  
10    elsif Chaine(Position_Chaine) /= Dans(Position_Dans)  
11    then  
12        return(False);  
13    else  
14        return(Correspond(Chaine, (Position_Chaine+1), Dans,  
            Position_Dans+1));  
15    end if;  
16 end Correspond;
```

`Indice_De` utilise `Correspond` sur tout les indices de `Dans`.

Cette méthode de recherche n'est pas optimale mais à l'avantage d'être facile à coder.

```
1 function Indice_De (Chaine : String; Dans : String) return  
    Natural is  
2     begin  
3         for I in 1..Dans'Last  
4         loop  
5             if Correspond (Chaine, 1, Dans, I)  
6             then  
7                 return(I);  
8             end if;  
9         end loop;  
10        return(0);  
11 end Indice_De;
```

1.2 Remplacement

Remplacement fait d'abord des tests sur les arguments puis remplace la chaîne De par la chaîne par.

```
1 function Remplacement (De : String; Par : String; Dans :  
    String) return String is  
2     Resultat : String := Dans;  
3     Indice_De_De : Integer;  
4 begin  
5     if De'Last /= Par'Last  
6     then  
7         raise Taille_Differente;  
8     end if;  
9     Indice_De_De := Indice_De(De, Dans) - 1;  
10    if Indice_De_De = -1  
11    then  
12        raise Motif_Non_Present;  
13    end if;  
14    for I in 1..(De'Last)  
15    loop  
16        Resultat(I+Indice_De_De) := Par(I);  
17    end loop;  
18    return(Resultat);  
19 exception  
20 when Taille_Differente =>  
21     Put_Line("Les entrées " & De & " et " & Par & " doivent  
    etre de meme longueur.");  
22     raise Taille_Differente;  
23 when Motif_Non_Present =>  
24     Put_Line(De & " n'apparaît pas dans " & Dans & ".");  
25     raise Motif_Non_Present;  
26 end Remplacement;
```

1.3 Problèmes

Si l'on veut que le remplacement soit effectué quel que soit les longueurs des chaînes, cela implique qu'il faut faire un décalage dans Dans.

Par exemple : remplacer Ind par Indice dans Ind_De doit renvoyer Indice_De. Il faut donc calculer la longueur de la nouvelle chaîne puis recopier l'ancienne avec un décalage à gauche ou à droite puis enfin remplacer.

Cela nécessite donc de coder plusieurs autres fonctions.

Pour remplacer toutes les occurrences d'une chaînes dans une autre, il faut modifier Indice_De pour que cette fonction ne renvoie non plus une seule occurrence mais plusieurs (par exemple avec une liste chaînée d'occurrence). Puis modifier Remplacement pour que la fonction prenne compte de la liste que renvoie Indice_De.

2 Exercice 2

2.1 Structure de donnée

Les notes sont des réels qui vont de 0 à 6. On suppose que l'on ne conserve que deux chiffres significatifs.

```
1  type T_Note is digits 2 range 0.0..6.0;
2  package Note_IO is new Float_IO(T_Note);
3  use Note_IO;
4
5  subtype T_Matiere is String(1..20);
```

Il n'y a pas plus de quarante étudiants à gérer. On déclare donc un tableau indexé de 1 à 40.

```
1  type T_Tableau_De_Note is array (Positive range 1..40) of
    T_Note;
```

On utilise un **record** pour le type T_Eleve.

```
1  subtype T_Nom is String(1..20);
2
3  type T_Eleve is
4      record
5          Nom : T_Nom := (others => ' ');
6          Prenom : T_Nom := (others => ' ');
7          Id : Positive;
8      end record;
```

T_Filiere est une liste chaînée qui contient comme information une matière et le tableau de note correspondant à cette matière.

```
1  type Element;
2  type T_Filiere is access Element;
3  type Element is record
4      Matiere : T_Matiere;
5      Note : T_Tableau_De_Note;
6      Suiv : T_Filiere;
7  end record;
```

2.2 Fonctions

Fonction récursive qui renvoie vrai si une matière est dans une filière et faux sinon.

```
1  function Fait_Partie (Matiere : T_Matiere; Filiere :  
    T_Filiere) return Boolean is  
2  begin  
3      if Filiere = null  
4      then  
5          return(False);  
6      elsif Filiere.all.Matiere = Matiere  
7      then  
8          return(True);  
9      else  
10         return(Fait_Partie(Matiere, Filiere.all.Suiv));  
11     end if;  
12 end Fait_Partie;
```

Fonction qui vérifie qu'une matière n'est pas simultanément dans deux filière

```
1  function Ne_Fait_Pas_Partie (Matiere : T_Matiere; Filiere1  
    : T_Filiere; Filiere2 : T_Filiere) return Boolean is  
2  begin  
3      return(not(Fait_Partie(Matiere, Filiere1) and then  
        Fait_Partie(Matiere, Filiere2)));  
4  end Ne_Fait_Pas_Partie;
```