Devoir maison 1 - Corrigé

M2 AIGEME, année 2008-2009

Quelques remarques générales :

- Concernant la notation, le premier exercice est noté sur 9 points et le deuxième sur 11. Pour chaque exercice, les explications qui accompagnent l'algorithme sont notées sur 2 points, le reste des points étant consacré à la compréhension du problème et à la correction de l'algorithme proposé.
- Expliquez ce que vous faites ! Mettre des commentaires dans le code c'est bien, mais ce qui est encore mieux c'est d'écrire avec des **phrases** en français le principe de l'algorithme (quelques phrases suffisent je ne vous demande pas un roman). Ceci permet à une personne extérieure (moi par exemple) de comprendre ce que vous voulez faire avant même de lire l'algorithme lui-même, et facilite donc la compréhension.
- Pour l'exercice 2, il n'est pas possible d'écrire un algorithme correct avec une seule boucle (ou alors la deuxième boucle est cachée dans l'appel à une fonction auxiliaire par exemple). On a besoin d'une boucle pour parcourir le texte, et d'une deuxième boucle pour vérifier si le mot apparaît à telle position dans le texte.
- Attention aux conditions dans les tests. Par exemple lorsque `var1` est une variable booléenne : `SI (var1=vrai)` est à éviter (même si c'est correct). Il vaut mieux écrire directement `SI (var1)`.
- Attention dans le choix des notations : il faut bien distinguer une affectation ($\leftarrow$ ou := ou encore =) d'une égalité (= ou == selon la façon dont vous notez l'affectation).
- Il est inutile de chercher à transformer une chaîne de caractères en un tableau de caractères. Vous avez sur les chaînes de caractères les opérations nécessaires (`long` qui vous donne la longueur du mot et `mot[i]` qui vous permet d'accéder au $i^{ième}$ caractère) qui font qu'elles se comportent exactement comme des tableaux de caractères.

Exercice 1 : Mots palindromes

On veut écrire un algorithme qui décide si un mot est un palindrome. Notre algorithme sera une fonction prenant en entrée une chaîne de caractères et renvoyant un booléen.

```
Fonction palindrome(mot : chaîne de caractères) : booléen

Var result : booléen

Debut
...
Renvoyer(result)
Fin
```

Un mot est un palindrome si sa première lettre est identique à la dernière, sa deuxième à l'avant-dernière etc ... Il va donc falloir comparer chacune des lettres formant le mot à une autre lettre. Si on appelle `n` la longueur du mot, et qu'on choisit comme convention de compter les lettres de `0` à `n-1`, on compare successivement :

- `mot[0]` et `mot[n-1]`,
- `mot[1]` et `mot[n-2]` ...
- plus généralement `mot[i]` et `mot[n-1-i]`.

On aura donc besoin d'une boucle qui permettra de faire varier un indice `i` variant de `0` à ... justement à quel moment peut-on s'arrêter ? En faisant varier `i` de `0` à `n-1` (c'est-à-dire que `mot[i]` va parcourir toutes les lettres du mot), on compare deux fois la première et la dernière lettre : lorsque `i`=`0` et `i`=`n-1`. De cette observation on déduit qu'il suffit que l'indice `i` ne parcourt que la moitié du mot, autrement dît $0 \leq i \leq \lfloor \frac{n}{2} \rfloor$, où $\lfloor \frac{n}{2} \rfloor$ est la partie entière de $\frac{n}{2}$.

Enfin, pour ne parcourir que la partie du mot qu'il est nécessaire de tester (par exemple pour le mot `perdu` une seule comparaison suffit pour voir que ce mot n'est pas un palindrome), on utilise un booléen `result` initialisé à `vrai`. L'algorithme peut alors s'écrire :

```
Fonction palindrome(mot : chaîne de caractères) : booléen

Var result : booléen
    i,n : entier

Debut
//Initialisation des variables
n ← long(mot);
result ← vrai;
i ← 0;

Tantque (result ET i ≤ ⌊n/2⌋)
Faire
   Si (mot[i] ≠ mot[n-1-i])
      Alors result ← faux;
   FinSi
   i ← i+1;
FinTantque

Renvoyer(result)
Fin
```

Exercice 2 : Décompte des occurences d'un motif dans un texte

On cherche à compter le nombre d'occurences d'un mot donné dans un texte. On va donc écrire un algorithme sous la forme :

```
Fonction compte_occurences(mot,texte : chaînes de caractères) : Entier

Var compteur : Entier

Debut
...
Renvoyer(compteur)
Fin
```

La variable `compteur` correspondra au nombre d'occurences du mot dans le texte. Le principe de l'algorithme est le suivant : on parcourt le texte et à chaque position `i`, on regarde si la chaîne de caractères commençant à la lettre `texte[i]` est le mot recherché. Si on appelle `n` la longueur du texte et `m` celle du mot, et toujours avec la convention de compter les lettres de `0` à `long(chaine)-1`, l'indice `i` doit varier de `0` à `n-m-1` (et non pas `n-1` car il faut avoir suffisamment de caractères pour trouver `mot`).

Dans un premier temps on suppose que l'on a déjà une fonction :

```
Fonction est_present(mot,texte : chaînes de caractères; i:Entier) : Booléen
```

qui nous dit si `mot` apparaît en position `i` dans `texte`. La fonction principale s'écrit alors :

```
Fonction compte_occurences(mot,texte : chaînes de caractères) : Entier

Var i,m,n,compteur : Entier
```

```
Debut
m ← long(mot);
n ← long(texte);
compteur ← 0;
Pour i de 0 à n-m-1 Faire
   Si (est_present(mot,texte,i))
      Alors compteur ← compteur+1;
   FinSi
FinPour
Renvoyer(compteur)
Fin
```

Il reste à écrire la fonction `est_present`. Le principe est le suivant : on initialise une variable entière `j` à `0`, et tant que les caractères `mot[j]` et `texte[i+j]` coïncident on incrémente `j`, ceci jusqu'à trouver deux caractères différents (et dans ce cas on renvoie `faux`) ou jusqu'à ce que `j` vaille `long(mot)-1` (et dans ce cas on a trouvé `mot` et on renvoie `vrai`).

```
Fonction est_present(mot,texte : chaînes de caractères; i:Entier) : Booléen

Var j,m : Entiers
    pareil : Booleen

Debut
j ← 0;
pareil ← vrai;
m ← long(mot);

TantQue (pareil ET j<m) Faire
   Si (mot[j]=texte[i+j])
      Alors j ← j+1;
   Sinon pareil ← faux;
   FinSi
Fin TantQue
Renvoyer(pareil);
Fin
```

On notera que cette fonction n'est jamais appelé avec `i>n-m`, et donc comme `j` varie de `0` à `m-1`, `i+j` ne sera jamais plus grand que `(n-m)+(m-1)=n-1`. Ceci pour justifier que `texte[i+j]` n'engendrera pas d'erreur par dépassement d'indice.