**M1102 - TD5**

## Exercice 1 : comptage de doublons dans une chaîne de caractères - V1

Une catégorie d'algorithmes, plus complexe, consiste à appliquer à chaque élément un traitement **qui dépend du contexte** de cet élément dans le tableau, par exemple de la valeur de l'élément qui le précède ou qui le suit, etc. C'est à ce type d'algorithme que correspond l'exercice proposé ici.   
  
L'algorithme demandé doit saisir (au clavier bien sûr !) une suite de lignes (des strings), jusqu'à ce que l'utilisateur tape une ligne vide (appuie sur la touche Entrée sans avoir saisi aucun caractère).   
  
Un traitement (précisé plus loin) doit être effectué sur les **doublons** de chaque ligne saisie (un doublon = deux caractères "consécutifs" identiques, définition précisée plus loin). Ce peut être par exemple un comptage.

L'algorithme de haut niveau est évident :

**algorithme** *CompterDoublons*

**debut**

boucle

declarer\_et\_saisir\_une\_ligne;

si (*ligne\_lue\_est\_vide*) sortie;

**// La ligne traitée est non vide**

effectuer\_le\_traitement; // comptage, en balayant la ligne

afficher\_le\_resultat;

fboucle

**fin**

La phase suivante du développement consiste :

* soit à remplacer directement les trois actions declarer\_et\_saisir\_une\_ligne, effectuer\_le\_traitement et afficher\_le\_resultat par les instructions qui les composent,
* soit à développer les trois actions sous forme de sous-programmes, l'algorithme de haut niveau se contentant de les appeler,
* soit de réaliser un compromis entre les deux solutions précédentes.

C'est cette dernière solution que nous proposons ici :

* declarer\_et\_saisir\_une\_ligne et afficher\_le\_resultat ne présentent pas de difficulté, et sont suffisamment simples pour être directement intégrées à l'algorithme de haut niveau,
* en revanche, effectuer\_le\_traitement est plus complexe, et mérite donc d'être isolé sous forme d'un sous-programme, plus précisément une fonction qui renvoie le résultat.

L'algorithme de haut niveau peut donc être réécrit :

**fonction** *CompterDoublons* (TabCar : **in** string)

renvoie entier\_naturel

**debut**

declarer Compt : entier\_naturel;

Compt <- 0;

// Calcul

renvoie Compt;

**fin**

**algorithme** *Nb\_de\_Doublons*

**debut**

boucle

// Saisie

declarer LigneLue : string;

saisir (LigneLue);

si (*ligne\_lue\_est\_vide*) sortie;

**// La ligne traitée est non vide** ***{1}***

// Comptage

declarer NbreDoublons : entier\_naturel;

NbreDoublons <- **CompterDoublons** (LigneLue);

// Affichage

afficher (LigneLue);

ligne\_suivante;

afficher ("Nombre de doublons : ", NbreDoublons);

ligne\_suivante;

fboucle

**fin**

Cet algorithme est conforme aux spécifications du problème : le comptage n'est effectué **que** sur des lignes non vides.   
  
**Travail demandé**   
  
Ecrire la fonction CompterDoublons() présentée ci-dessus.   
  
Les "règles du jeu" (les spécifications) sont les suivantes :

1. un doublon est une suite de deux caractères consécutifs (deux éléments dont les rangs diffèrent d'une unité ) identiques.
2. **trois** caractères consécutifs identiques constituent **deux** doublons.

les doublons de caractères d'espacement  **ne sont pas comptabilisés**. Pour savoir si un caractère est un caractère d'espacement, utiliser le prédicat IsSpace() de profil :

fonction IsSpace (Car : in caractere) renvoie booleen;

Lorsque le caractère courant est de rang i, il y a deux façons de tester s'il appartient à un doublon :

1. soit en testant le caractère de rang précédent (de rang i - 1),
2. soit en testant le caractère de rang suivant (de rang i + 1),

Dans le premier cas, le premier élément ne peut être comparé avec son prédécesseur. Il faut donc commencer les comparaisons à l'élément de rang 1.   
  
Dans le second cas, le dernier élément ne peut être comparé avec son successeur. Il faut donc terminer les comparaisons par l'avant-dernier élément.   
  
**Variantes**   
  
Quelques modifications, qui peuvent paraître mineures, dans les spécifications, peuvent nécessiter des modifications profondes de l'algorithme, comme par exemple l'obligation de changer de schéma itératif.   
  
Ecrire les modifications nécessitées par les spécifications suivantes :   
  
**Variante 1**

1. un doublon est une suite de deux caractères consécutifs identiques.
2. un même caractère ne peut appartenir à deux doublons différents. En conséquence, il faut **quatre** caractères consécutifs identiques pour constituer **deux** doublons.
3. les doublons de caractères d'espacement  **ne sont pas comptabilisés**

**Variante 2**

1. un doublon est une suite de deux caractères consécutifs identiques.
2. **trois** caractères consécutifs identiques constituent **deux** doublons.
3. les doublons de caractères d'espacement  **sont comptabilisés** : par exemple un espace suivi d'une tabulation (ou l'inverse) est un doublon.

**Variante 3**

1. un doublon est une suite de deux caractères consécutifs identiques.
2. **trois** caractères consécutifs identiques constituent **deux** doublons.
3. les caractères d'espacement doivent être ignorés :
   * il ne faut pas compter les doublons de caractères d'espacement,
   * deux caractères identiques séparés par un nombre quelconque de caractères d'espacement, forment un doublon.

## Exercice 2 : comptage de doublons dans une chaîne de caractères - V2

L'approche envisagée ici en est totalement différente : l'algorithme de haut niveau proposé consiste à se positionner successivement sur les doublons successifs, et à les comptabiliser au fur et à mesure. L'algorithme se termine lorsque plus aucun doublon n'est trouvé.   
  
L'analyse est rigoureusement analogue à celle qui a été étudiée dans ["Fonction de comptage d'un caractère dans une chaîne de caractères"](javascript:PopUpCours%20('../General/PopUpCours.php?SlxPage=TdAlgo_05#ComptCaractQcq', 'Algo');). Elle passe par l'écriture de la fonction FindFirstDoublon(), de profil :

fonction FindFirstDoublon (TabCar : in tableau\_de caractere,

Debut : in entier\_naturel)

renvoie entier\_naturel;

qui renvoie la position du **premier** caractère du **premier** doublon trouvé dans TabCar **à partir** de la position initiale Debut. Elle renvoie taille (TabCar) si aucun doublon n'est trouvé.   
  
**Travail demandé**   
  
Ecrire le corps de la fonction FindFirstDoublon(), puis écrire l'algorithme qui compte le nombre de doublons dans une chaîne saisie au clavier.   
  
**Remarques**

1. prendre n'importe laquelle des définitions des doublons de l'exercice ["Comptage de doublons dans un tableau - V1"](javascript:PopUpCours%20('../General/PopUpCours.php?SlxPage=TdAlgo_06#ComptDoublonsV1', 'Algo'););
2. la ligne peut être vide.

## Exercice 3 : recherche d'une sous-chaîne dans une chaîne de caractères

Comme on l'a vu dans l'exercice ["Comptage de doublons dans un tableau - V2"](http://infodoc.iut.univ-aix.fr/%7Emathieu/General/PopUpCours.php?SlxPage=TdAlgo_06#ComptDoublonsV2), le comptage d'un doublon (couple de lettres consécutives identiques) peut être considéré comme la répétition de recherches successives de ce doublon à partir de la position du doublon précédemment trouvé.   
  
Un doublon peut être considéré comme un **motif** (***pattern***) connu par sa **propriété**.   
  
On peut envisager des motifs définis par leur **valeur**, exemple ["Comptage des suites le, les, lle"](http://infodoc.iut.univ-aix.fr/%7Emathieu/General/PopUpCours.php?SlxPage=TdAlgo_06#ComptLLE).   
  
La forme générale de l'algorithme est totalement identique, seule change la reconnaissance du **motif**.   
  
Lorsque le motif dépasse deux caractères, il devient lourd et maladroit de mémoriser individuellement les caractères qui composent le motif. Il est préférable de considérer que l'on cherche une sous-chaine (de 2, 3 lettres ou plus) dans une chaîne (ou un sous-tableau dans un tableau). C'est ce qui vous est proposé dans l'exercice ci-dessous.   
  
Ecrire la fonction FindSubstrInStr() qui renvoie le rang de la première apparition d'une sous-chaîne dans une chaîne de caractères, à partir d'un rang de début de recherche, tous trois passés en paramètres. Plus précisément, elle renvoie le rang du premier caractère de la sous-chaîne dans la chaîne.   
  
La valeur de retour est obligatoirement dans l'intervalle [0, taille (Chaine) - 1] si la sous-chaîne est présente. On choisira donc de renvoyer la valeur taille (Chaine) si la sous-chaîne n'a pas été trouvée.   
  
Ecrire l'algorithme qui saisit au clavier une ligne, puis dans une boucle :

* saisit une sous-chaîne jusqu'à une sous-chaîne vide,
* l'affiche **entre guillemets**, suivie de son rang dans la chaîne si elle est présente, ou d'un message indiquant qu'elle n'a pas été trouvée. Utiliser évidemment la fonction FindSubstrInStr() !

La solution qui est demandée ici est la plus simple : la sous-chaîne est comparée à la chaîne à partir de la première position de la chaîne. Si la coïncidence n'est pas totale, elle est de nouveau comparée à la chaîne à partir de la deuxième position, etc.

## Exercice 4 : comptage des mots dans une chaîne de caractères

Dans cet exercice un mot est défini comme une suite **non vide** de lettres (minuscules ou majuscules), de chiffres et ou de caractère "souligné" '\_' (***underscore***).   
  
Ecrire l'algorithme qui saisit au clavier une suite de lignes (des strings), jusqu'à ce que l'utilisateur tape une ligne vide (appuie sur la touche Entrée sans avoir saisi aucun caractère).   
  
Au fur et à mesure de la saisie :

* afficher la chaîne lue sur une nouvelle ligne,
* afficher sur la ligne suivante le nombre de mots de la ligne lue, précédé de la chaîne
* Nombre de mots de la ligne :
* sauter une ligne.

Les prédicats suivants sont supposés exister :

fonction IsUpper (Car : in caractere) renvoie booleen;

fonction IsLower (Car : in caractere) renvoie booleen;

fonction IsDigit (Car : in caractere) renvoie booleen;

et renvoient vrai si le caractère qui leur est passé en paramètre est respectivement une majuscule, une minuscule ou un chiffre décimal (de '0' à '9').   
  
**Remarques**

1. Il ne s'agit pas d'identifier complètement les mots mais seulement de les compter.   
     
   Or il y a autant de mots que de débuts de mots ! Il suffit donc de compter les débuts de mots.   
     
   Le début d'un mot peut être défini comme une lettre (majuscule ou minuscule), un chiffre ou un '\_', précédé par un caractère qui n'est aucun de ces caractères.
2. Un mot peut commencer au rang 0. Il ne faut donc pas commencer la boucle de traitement au caractère de rang 1. On ne peut pas non plus comparer le premier caractère (de rang 0) avec son prédécesseur.   
     
   Il y a donc deux possibilités : soit traiter à part le premier caractère, ce qui est lourd, soit comparer le caractère courant avec son prédécesseur, qui aura été mémorisé dans une variable intermédiaire et qui, pour le rang 0, aura correctement été initialisé.

Comme nous l'avons signalé dans plusieurs exercices précédents, il y a deux approches algorithmiques **extrêmement différentes**, qui apparaissent clairement dans le cœur des deux algorithmes de haut niveau ci-dessous :   
  
**Approche caractère/caractère**

....

Se\_positionner\_en\_debut\_de\_chaine;

pour (chaque\_caractere\_de\_la\_chaine)

faire

si (c\_est\_un\_debut\_de\_mot)

Compter\_le\_mot;

fsi

Passer\_au\_caractere\_suivant;

ffaire

....

Dans cet algorithme, la chaîne est vue comme une simple suite de caractères, qui sont donc traités séquentiellement.   
  
**Approche mot/mot**

....

Se\_positionner\_en\_debut\_de\_mot;

jusqua (fin\_de\_ligne)

faire

Compter\_le\_mot;

Passer\_a\_la\_fin\_du\_mot;

Se\_positionner\_en\_debut\_de\_mot;

ffaire

....

Dans ce cas, l'algorithme superpose à la structure **physique** du tableau vu comme une "suite de caractères", une structure **logique** de "suite de mots".   
  
Il est recommandé d'essayer les deux solutions.   
  
**Variante 1** : compter les mots en recherchant les **fins de mots**.   
  
La fin d'un mot peut être définie comme une lettre (majuscule ou minuscule), un chiffre ou un '\_', suivi par un caractère qui n'est aucun de ces caractères, sauf éventuellement pour le dernier caractère qui n'est suivi de rien.   
  
**Variante 2** : à la définition précédente d'un mot (*"un mot est défini comme une suite* ***non vide*** *de lettres (minuscules ou majuscules), de chiffres et ou de caractère "souligné"* '\_'*"*) est ajoutée une contrainte supplémentaire : *un mot doit* ***commencer*** *par une lettre (minuscule ou majuscule)*.   
  
Dans la ligne suivante :

A440 123 \_B732 44 \_25

il n'y a que **deux** mots : A440 et B732.   
  
Modifier l'algorithme en conséquence.