

# R1.01 – TD6 – partie #1

## Exercice 1 : écriture de prédicats

La fonction `trouveCarInStr ()` recherche la présence (ou l'absence) d'un caractère dans un ensemble. Or si l'ensemble est continu (il ne contient pas de trou) ou s'il est très petit (3,4 éléments au maximum), nous n'avons pas besoin de parcourir tout cet ensemble, seules quelques comparaisons suffisent.

Ecrire les prédicats suivants :

```
fonction isUpper (car : in caractere) renvoie booleen;  
fonction isLower (car : in caractere) renvoie booleen;  
fonction isDigit (car : in caractere) renvoie booleen;
```

Ces prédicats 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').

Ecrire le prédicat `isSpace ()`, qui renvoie `vrai` si le caractère passé en paramètre est un caractère d'espacement (espace, tabulation ou retour à la ligne).

## Exercice 2 : 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 (appui 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  
        declarerEtSaisirUneLigne;  
        si (ligneLueEstVide) sortie;  
  
        // La ligne traitée est non vide  
        effectuerLeTraitement; // comptage, en balayant la ligne  
        afficherLeResultat;  
    fboucle
```

```
fin
```

La phase suivante du développement consiste :

- soit à remplacer directement les trois actions `declarerEtSaisirUneLigne`, `effectuerLeTraitement` et `afficherLeResultat` 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 :

- `declarerEtSaisirUneLigne` et `afficherLeResultat` ne présentent pas de difficulté, et sont suffisamment simples pour être directement intégrées à l'algorithme de haut niveau,
- en revanche, `effectuerLeTraitement` est plus complexe, et mérite donc d'être isolé sous forme d'un sous-programme, plus précisément une fonction qui renvoie le résultat.

### Travail demandé

Ecrire la fonction `compterDoublons()` présentée ci-dessous. 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**.

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  $i - 1$ ), dans ce cas, le premier élément ne peut être comparé avec son prédécesseur. Il faut donc commencer les comparaisons à l'élément de rang 1.
2. soit en testant le caractère de rang suivant (de rang  $i + 1$ ), dans ce cas, le dernier élément ne peut être comparé avec son successeur. Il faut donc terminer les comparaisons par l'avant-dernier élément.

Ecrire une version de la fonction `compterDoublons()` en utilisant une boucle `pour`, puis une autre en utilisant une boucle `tant_que`.

## 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'espace **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'espace **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'espace doivent être ignorés :
  - il ne faut pas compter les doublons de caractères d'espace,
  - deux caractères identiques séparés par un nombre quelconque de caractères d'espace, forment un doublon.