# DS - Algorithmes et Structures de Données
# Vendredi 10 Janvier 2025
# Durée 3h – Cours et TD NON autorisés

## 1. Chaines de caractères (6 pts)

On souhaite séparer une ligne de commande en un tableau de chaines `arg[i]`.
`arg1 arg2 … argn` ⇒ `arg[1] arg[2] … arg[n]`

La liste des caractères spéciaux est la suivante :
<*espace*> ( ) : séparateur d'arguments
<*backslash*> (\) : le caractère d'après n'est pas spécial
<*apostrophe*> (') : plus aucun caractère spécial jusqu'au prochain <*apostrophe*>
<*guillemet*> (") : <*espace*> et <*apostrophe*> ne sont pas spéciaux (seul <*backslash*> est spécial)

Exemple :

`aa 'b b' 'c c c'`

| | | |
|---|---|---|
| aa | b b | c c c |

`a "bb cc 'dd'"`

| | |
|---|---|
| a | bb cc 'dd' |

`a "b \"c'" 'a "b'`

| | | |
|---|---|---|
| a | b "c' | a "b |

On supposera que le nombre maximal d'arguments `arg[i]` est 10.
Ecrire en pseudo-langage la procédure `analyse` qui sépare une chaine de caractères `c` en plusieurs paramètres, enregistrés dans un tableau de chaines.

```
Type tabarg = tableau [1..10] de chaine
Procedure analyse (E c : chaine ; S arg : tabarg)
```

## 2. Création d'une liste doublement chainée circulaire à partir d'une liste simplement chainée (6 pts)

On souhaite écrire en pseudo-langage une fonction qui, à partir d'une liste `l` simplement chainée de type `liste`, crée une liste doublement chainée circulaire de type `liste-double-circ`.
**2.1.** Expliquer en français le principe de cette création et faire un dessin avec la liste en entrée et la liste en sortie.
**2.2.** Ecrire en pseudo-langage la fonction `créer`

```
Fonction créer (l : liste) : liste-double-circ

Type  liste-double-circ = ^cellule-double
      cellule-double =  Enregistrement
                            val : chaine
                            suiv, prec : ^cellule-double
                        FinEnregistrement
      liste = ^cellule
      cellule =  Enregistrement
                        val : chaine
                        suiv : ^cellule
                 FinEnregistrement
```

## 3. Jeu du solitaire bulgare (8 pts)

On veut illustrer le jeu du solitaire bulgare :

- partager un paquet de N cartes en 2 tas ayant chacun un nombre quelconque de cartes,
- prendre une carte sur chacun des tas pour former un nouveau tas (placé à gauche des précédents),
- itérer l'étape précédente jusqu'à ce que les nombres de cartes des tas forment une suite consécutive décroissante (on suppose que le jeu converge).

*Exemple :*

Soit un jeu de 10 cartes. On commence avec deux tas : par exemple, un tas de 6 cartes et un tas de 4 cartes (ligne 1). Les situations successives sont les suivantes :

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ligne 1 : | | | | | | 6 | 4 | ⇒ | (6,4) | |
| ligne 2 : | | | | | 2 | 5 | 3 | ⇒ | (2,5,3) | |
| ligne 3 : | | | | 3 | 1 | 4 | 2 | ⇒ | (3,1,4,2) | |
| ligne 4 : | | | 4 | 2 | | 3 | 1 | ⇒ | (4,2,3,1) | |
| ligne 5 : | | 4 | 3 | 1 | | 2 | | ⇒ | (4,3,1,2) | |
| ligne 6 : | 4 | 3 | 2 | | | 1 | | ⇒ | (4,3,2,1) | On s'arrête |

On se propose d'implémenter ce solitaire avec la structure de données suivante :

```
Type  tas   = Enregistrement
                  nbcarte : entier
                  suiv : ^tas
              FinEnregistrement
      ligne = ^tas
      tablig = tableau [1..20] de ligne  chaque case du tableau pointe sur une
                                         ligne (liste de tas), max 20 lignes
```

Pour toutes les questions ci-dessous, ne pas utiliser les opérations du TAD liste.

**3.1.** Faire un dessin de la structure de données sur l'exemple.

**3.2.** Ecrire en pseudo-langage une procédure `créerjeu` qui initialise le tableau de lignes `t` en demandant à l'utilisateur de rentrer un nombre de tas et un nombre de cartes pour chaque tas.

```
Procédure créerjeu (S t : tablig)
```

**3.3.** Ecrire en pseudo-langage une procédure `copierlig` qui copie la ligne `l` du tableau `t` à la ligne suivante.

```
Procédure copierlig (E/S t : tablig ; E l : entier)
```

**3.4.** Ecrire en pseudo-langage une procédure `prendrecartes` qui décrémente tous les tas d'une ligne `l` et renvoie le nombre `nbc` de cartes prises.

```
Procédure prendrecartes (E t : tablig, l : entier ; S nbc : entier)
```

**3.5.** Ecrire en pseudo-langage une procédure `compacter` qui supprime les tas ayant 0 carte d'une ligne `l`.

```
Procédure compacter (E/S t : tablig ; E l : entier)
```

**3.6.** Ecrire en pseudo-langage une procédure `ajoutertas` qui ajoute un tas de `n` cartes en tête d'une ligne `l`.

```
Procédure ajoutertas (E/S t : tablig ; E l, n : entier)
```

**3.7.** Ecrire en pseudo-langage une fonction qui retourne vrai si les nombres de cartes dans chaque tas d'une ligne `l` forment une suite consécutive décroissante (ex. lig6 : 4,3,2,1).

```
Fonction suitedec (t : tablig, l : entier) : booléen
```

**3.8.** Ecrire en pseudo-langage un programme complet simulant le jeu du solitaire bulgare. On supposera que le jeu converge au bout de 20 lignes max.