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'

`Aa bb cc`

`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)
Var   i,j : entier
      mot : chaine
      dansa, dansg, bs : booléen
Début
i←0
dansa←faux       {pour savoir si on est dans une '}
dansg←faux       {pour savoir si on est dans un "}
bs←faux          {pour savoir si on est après un \}
mot←''
Pour j ← 1 à lg(c) inc +1 faire
      Si bs                          {après un \, pas de car spécial}
            alors mot ← mot + c[j]
                  bs ← Faux
            Sinon Selon c[j]
                  {apostrophe}
                  '''' : Si not dansg
                              Alors dansa ← not dansa
                              Sinon mot ← mot + c[j]
                         FinSi
                  {guillemet}
                  '"' : Si not dansa
                              Alors dansg ← not dansg
                              Sinon mot ← mot + c[j]
                        FinSi
```

```
                  {backslash}
                  '\' : Si not dansa {\ n'est pas spécial sans une '}
                              Alors bs ← vrai
                              Sinon mot ← mot + c[j]
                        FinSi
                  {espace}
                  ' ':  Si not dansa et not dansg
                           Alors Si lg(mot) > 0
                                       Alors i ← i+1
                                             arg[i] ← mot {on garde le mot}
                                             mot ← '' {on initialise un nouv. mot}
                                 FinSi
                           Sinon mot ← mot + c[j]  {l'espace n'est pas spécial}
                        FinSi
                  Sinon mot ← mot + c[j]
                  FinSelon
      FinSi
FinPour
{il ne faut pas oublier de récupérer le dernier argument}
i ← i+1
arg[i] ← mot
Fin
```

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`

```
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

Fonction créer (l : liste) : liste-double-circ
Var ldc,t : liste-double-circ
Debut
ldc ← nil
TantQue l<>nil Faire
      t ← allouer(cellule-double)
      t^.val ← l^.val
      Si ldc=nil
            Alors ldc  ← t
                  ldc^.suiv ← t
                  ldc^.prec ← t
            Sinon t^.suiv ← ldc
                  t^.prec ← ldc^.prec
                  ldc^.prec^.suiv ← t
                  ldc^.prec ← t
      FinSi
      l ← l^.suiv
FinTantQue
Retourner(ldc)
Fin
```

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)
Var i, nt, nc : entier
Début
Pour i←2 à 20 inc +1 Faire
      t[i]←nil
FinPour
Ecrire ('Entrez le nb de tas initial >=2 : ')
lire(nt)
Pour i←1 à nt inc +1 faire
      ecrire(' Entrez le nb de cartes du ',i,' tas')
      lire(nc)
      p←allouer(tas)
      p^.nbcarte←nc
      p^.suiv←t[1]
      t[1]←p          {je fais les insertions en tête}
FinPour
Fin
```

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)
Var tl, pl, q : ^tas
Début
tl←t[l]
```

```
TantQue tl≠nil faire
      pl←allouer(tas)
      pl←tl^.nbcarte
      Si t[l+1]=nil
            alors t[l]←pl
                  q←pl
            sinon q^.suiv←pl
                  q←q^.suiv
      FinSi
      tl←tl^.suiv
FinTantque
q^.suiv←nil
Fin
```

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)
Var tl : ^tas
Début
tl←t[l]
nbc←0
TantQue tl≠nil faire
      tl^.nbcarte←tl^.nbcarte-1
      nbc←i+1
      tl←tl^.suiv
FinTantque
Fin
```

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)
Var tl, pl : ^tas
Début
tl←t[l]
TantQue tl≠nil faire
      Si t[l]^.nbcarte=0      {suppression en tête}
            Alors pl←t[l]
                  t[l]←t[l]^.suiv
                  récupérer(pl)
                  tl←t[l]
            Sinon
                  Si tl^.suiv≠nil et tl^.suiv^.nbcarte=0
                        alors pl←tl^.suiv   {suppression en milieu ou fin}
                              tl^.suiv←tl^.suiv^.suiv
                              récupérer(pl)
                        sinon tl←tl^.suiv
                  FinSi
      FinSi
FinTantque
Fin
```

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)
Var pl : ^tas
Début
pl←allouer(tas)
pl^.nbcarte←n
pl^.suiv←t[l]
t[l]←pl
Fin
```

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
Var pl : ^tas
    suite : booléen
Début
pl ← t[l]
{on considère qu'il y a au moins 2 tas dans la ligne}
suite←vrai
Finsi
TantQue (pl^.suiv<>nil) et suite Faire
      suite ←(pl^.nbcarte-pl^.suiv^.nbcarte=1)
      {suite consécutive décroissante}
      pl ← pl^.suiv
FinTantQue
retourner(suite)
Fin
```

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.

```
Programme solitaire
Var t : tablig
    l,nbc : entier
Début
créerjeu(t)
l ← 1
TantQue non suitedec(t,l) et l<20 Faire
      copierlig(t,l)
      l ← l+1
      prendrecartes (t,l,nbc)
      compacter(t,l)
      ajoutertas(t,l,nbc)
FintantQue
écrire('La suite converge à la ',l,'e ligne')
Fin
```