DS2 - Algorithmes et Structures de Données
Jeudi 8 Janvier 2026

Durée 3h – Documents non autorisés

Exercice 1 : valeur décimale d'un nombre écrit en chiffres romains (6 pts)

Les conventions d'écriture des nombres en chiffres romains permettent de représenter les entiers de 1 à 3999.
Les chiffres sont les suivants : I = 1 ; V = 5 ; X = 10 ; L = 50 ; C = 100 ; D = 500 ; M = 1000.
Le zéro n'existe pas. La règle de base est de faire la somme des chiffres.
Exemples :
III = 1+1+1 = 3 ; VI = 5+1= 6 ; ; XIII = 10+1+1+1 = 13 ;
MMXXVI = 1000+1000+10+10+5+1 =2026
Cependant, on n'écrit jamais plus de 3 chiffres identiques consécutifs. On a donc recours aux 6 écritures particulières suivantes :
IV = -1+5 = 4 ; IX = -1+10 = 9 ; XL = -10+50 = 40 ; XC = -10+100 = 90 ;
CD = -100+500 = 400 ; CM = -100+1000 = 900
Autres exemples :
LXXIII = 50+10+10+1+1+1 = 73 ; CDXLIII = -100+500-10+50+1+1+1 = 443 ;
DCCIII = 500+100+100+1+1+1 = 703 ; CMIV = -10+1000-1+5 = 904 ;
MMMCMXCIX = 1000+1000+1000-100+1000-10+100-1+10 = 3999
Un nombre écrit en chiffres romains est représenté par une chaîne de caractères de longueur maximale égale à 15.
On souhaite écrire une fonction qui, étant donnée une chaîne représentant un nombre écrit (correctement) en chiffres romains, le convertit en un entier (valeur décimale). On dispose d'un tableau contenant chaque chiffre romain avec sa conversion dans le système décimal :

```
Type  valeur = Enregistrement
                   rom : car
                   dec : entier
                FinEnregistrement
      tab-val = tableau [1..7] de valeur
```

1.1. Donner le principe itératif de la méthode utilisée : discuter notamment la valeur (positive ou négative) du caractère courant en fonction du caractère suivant.
On initialise val à 0 et on parcourt de gauche à droite la chaîne nbr représentant le nombre romain. A chaque itération (i < lg(nbr)), on ajoute ou retranche la valeur décimale de nbr[i] (appelée v) à val.
- si la valeur décimale de nbr[i + 1] ≤ v alors on ajoute v à val,
- sinon on retranche v à val.

A la fin, on ajoute la valeur décimale du dernier caractère (pour i=lg(nbr)) à val.

1.2. Ecrire une fonction `donneval` qui renvoie la valeur décimale d'un **chiffre** romain :

```
Fonction donneval(cr : car, t : tab-val) : entier
Var i : entier
Début
i←1
TantQue t[i].rom≠cr Faire
      {on est sûr que le caractère cr est dans le tableau t}
      i←i+1
FintantQue
retourner (t[i].dec)
Fin
```

1.3. Ecrire une fonction `traduit` qui renvoie la conversion d'un **nombre** écrit en chiffres romains dans le système décimal et qui utilise `donneval` :

```
Fonction traduit(nbr : chaîne, t : tab-val) : entier
Var val,i,n,m : entier
Debut
val ← 0
m ← donneval(nbr[1],t)
Pour i←1 à lg(nbr)-1 inc +1 Faire
      n ← donneval(nbr[i + 1],t)
      Si n ≤ m
            Alors val ← val + m
            Sinon val ← val – m
      FinSi
      m←n
FinPour
val ← val + n {pour la dernière valeur}
retourner(val)
Fin
```

Exercice 2 : dessin récursif d'un arbre binaire (4 pts)

On souhaite dessiner récursivement un arbre binaire `A` dont la racine est de coordonnées `x` et `y` (donnés en nombre de pixels). On dispose des outils suivants :

`Procédure va-en(E x,y : entier)` qui permet de déplacer le crayon au point de coordonnées `(x,y)`.

`Procédure trace-ligne(E x,y : entier)` qui permet de tracer un trait partant du point courant du crayon jusqu'au point de coordonnées `(x,y)`.

Pour ce dessin récursif, on suppose qu'à chaque niveau, la distance `d` (horizontale entre chaque noeud) est divisée par 2 et la hauteur `h` (verticale entre chaque niveau) est multipliée par 2/3.

2.1. Expliquer en français la méthode **récursive** utilisée.

On effectue un parcours en profondeur à gauche de notre arbre binaire et on dessine au fur et à mesure les branches parcourues. Si notre arbre est vide, on ne fait rien. Sinon, si le fils-gauche de notre arbre n'est pas vide, on dessine la branche et on appelle récursivement la procédure sur le fils gauche. Quand il n'y a plus de fils gauche, on fait de même pour le fils droit. La procédure s'arrête s'il n'y a plus de fils gauche et plus de fils droit (une feuille).

2.2. Ecrire en pseudo-langage une procédure **récursive** qui réalise ce dessin.

```
Procédure dessine-ArbreBinaire (E A : ArbreBinaire ; x,y,d,h : entier)
Début
Si ¬ arbre-vide(A)
    Alors Si ¬ arbre-vide(fils-gauche(A))
              Alors va-en(x,y)
                    trace-ligne(x - d div 2, y + h)
                    dessine-ArbreBinaire(fils-gauche(A), x - d div 2, y+h,
                                         d div 2, h * 2 div 3)
          FinSi
          Si ¬ arbre-vide(fils-droit(A))
               Alors va-en(x,y)
                    trace-ligne(x + d div 2, y + h)
                    dessine-ArbreBinaire(fils-droit(A), x + d div 2, y+h,
                                         d div 2, h * 2 div 3)
          FinSi
FinSi
Fin
```

Exercice 3 : structure de données dynamique (10 pts)

3.1. Ecrire en pseudo-langage les types pour représenter cette structure de données.

```
Type disco : ^catégorie
     catégorie : Enregistrement
                   la : ^artiste
                   nomc : chaine
                   suiv : ^catégorie
                 FinEnregistrement
     artiste : Enregistrement
                   noma : chaine
                   nb : entier
                   suiv : ^artiste
               FinEnregistrement
```

3.2. Ecrire en pseudo-langage une procédure d'ajout d'une catégorie de musique sans artiste (la liste est triée dans l'ordre alphabétique).

```
Procédure ajoute-cat (E nom : chaine, E/S d : disco)
Var r,p : disco
Début
r←allouer(catégorie)
r^.nomc←nom
r^.la←nil
Si d=nil ou nom<d^.nomc
     Alors r^.suiv←d
           d←r
     Sinon p←d
           TantQue (p^.suiv≠nil) et (p^.suiv^.nomc<nom) Faire
               p←p^.suiv
           FinTantQue
           r^.suiv←p^.suiv
           p^.suiv←r
FinSi
Fin
```

3.3. Ecrire en pseudo-langage une procédure d'ajout d'un artiste et du nombre d'album (liste triée dans l'ordre alphabétique) dans une catégorie musicale donnée (existante).

```
Procédure ajoute-artiste (E a,c : chaine, n : entier, d : disco)
Var  p : disco
     r,q : ^artiste
Début
p←d
{on cherche la catégorie}
TantQue (p^.nomc≠c) et (p≠nil) Faire
     p←p^.suiv
FinTantQue
Si p≠nil      {sinon la catégorie n'existe pas}
     Alors r←allouer(artiste)
           r^.noma←a
           r^.nb←n
           q←p^.la
           Si q=nil ou a<q^.noma
               Alors r^.suiv←q
                     p^.la←r
               Sinon  TantQue (q^.suiv≠nil) et (q^.suiv^.noma<a) Faire
                           q←q^.suiv
                      FinTantQue
                      r^.suiv←q^.suiv
                      q^.suiv←r
          FinSi
FinSi
```

```
Fin
```

3.4. Ecrire en pseudo-langage une fonction qui retourne le nombre total d'albums.

```
Fonction donne-album (d : disco) : entier
Var  p : disco
     la : ^artiste
     nb : entier
Début
p←d
nb←0
TantQue (p≠nil) Faire
     la←p^.la
     TantQue (la≠nil) Faire
          nb←nb+la^.nb
          la←la^.suiv
     FinTantQue
     p←p^.suiv
FinTantQue
retourner(nb)
Fin
```

3.5. Ecrire en pseudo-langage une fonction qui retourne la catégorie musicale d'un artiste.

```
Fonction donne-catégorie (E a : chaine, d : disco) : chaine
Var  p : disco
     la : ^artiste
     cat : chaine
Début
p ← d
trouvé ← faux
cat ← ''
TantQue ¬ trouvé et p≠nil Faire
     la ← p^.la
     TantQue ¬ trouvé et (la^.noma>=a) et (la≠nil) Faire
          trouvé ← (la^.noma=a)
          la ← la^.suiv
     FinTantQue
     Si ¬ trouvé
          Alors p←p^.suiv
     FinSi
FinTantQue
Si trouvé
     Alors cat←p^.nomc
     Sinon ('pas d'artiste de ce nom')
FinSi
retourner(cat)
Fin
```

3.6. Ecrire en pseudo-langage une fonction qui retourne la catégorie ayant le plus d'artistes.

```
Fonction donne-max-a (d : disco) : chaine
Var  p : disco
     la : ^artiste
     max, nb : entier
     cat : chaine
Début
p ← d
max ← 0
TantQue (p≠nil) Faire
     la ← p^.la
     nb ← 0
     TantQue (la≠nil) Faire
          nb←nb+1
```

```
          la←la^.suiv
     FinTantQue
     Si nb > max
          Alors max ← nb
                cat ← p^.nomc
     FinSi
     p ← p^.suiv
FinTantQue
retourner(cat)
Fin
```