M2 CCI – Algorithmique – Devoir surveillé 2

Durée 2h, sans documents **4 décembre 2025**

NE PAS RECOPIER les énoncés des questions. Ne pas perdre de temps à un soin excessif de la présentation. Les questions sont indépendantes. Le barème est indicatif.

1. À propos de séquences d'entiers (représentation contiguë de séquences)

On traite des séquences d'entiers représentées sous forme contiguë avec longueur explicite :

Lmax : constante de type entier > 0 *{longueur maximum des séquences considérées}*

Q1 [4 points]

On spécifie les deux fonctions suivantes :

PrésentUneSeuleFois : fonction (E : entier, T : tableau sur [1…Lmax] d'entier,
L : entier sur [0…Lmax]) → booléen

*{vrai si et seulement si l'entier **E** est présent **une fois et une seule** dans la séquence représentée dans le tableau **T** entre **1** et **L**.}*

Exemple : si la séquence est [**3**, **10**, 2, **7**, 25, **3**, **10**, **7**, 1], représentée avec ces valeurs dans **T** et de longueur **L** = 9, alors :
PrésentUneSeuleFois (2, T, L) = vrai car 2 n'apparaît qu'une seule fois
PrésentUneSeuleFois (10, T, L) = faux car 10 apparaît 2 fois

TousPrésentsUneSeuleFois : fonction (TR : tableau sur [1…Lmax] d'entier,
LR : entier sur [0…Lmax],
TS : tableau sur [1…Lmax] d'entier,
LS : entier sur [0…Lmax]) → booléen

*{vrai si et seulement chaque entier de la séquence représentée dans le tableau **TR** entre **1** et **LR** est présent une fois et une seule dans la séquence représentée dans le tableau **TS** entre **1** et **LS**, et ceci dans un ordre quelconque. **Pré-condition** : les entiers de la séquence représentée dans **TR** sont distincts deux à deux}*

Exemple :

- si la séquence **R** est [2, 25, 1], représentée avec ces valeurs dans **TR** et de longueur **LR** = 3, et si la séquence **S** est [**3**, **10**, 2, **7**, 25, **3**, **10**, **7**, 1], représentée avec ces valeurs dans **TS** et de longueur **LS** = **9**, alors :
  TousPrésentUneSeuleFois (TR, LR, TS, LS) = vrai
  Car chacun des éléments de **R** n'apparaît qu'une seule fois dans **S**.
- si la séquence **R** est [10, 3, 2], représentée avec ces valeurs dans **TR** et de longueur **LR** = 3, et si la séquence **S** est [**3**, **10**, 2, **7**, 25, **3**, **10**, **7**, 1], représentée avec ces valeurs dans **TS** et de longueur **LS** = **9**, alors :
  TousPrésentUneSeuleFois (TR, LR, TS, LS) = faux
  Car 10 et 3 (de **R**) apparaissent 2 fois dans S.

— **Donner une réalisation** de la fonction **PrésentUneSeuleFois**

— **Donner une réalisation** de la fonction **TousPrésentsUneSeuleFois**.

2. Inversion de l'ordre entre deux éléments consécutifs d'une liste chaînée

Étant donnés un entier **X** et une séquence d'entiers **S**, on veut inverser l'ordre entre le **premier élément de valeur X** dans **S** et son **successeur dans S**. La séquence **S** est inchangée si **X** n'appartient pas à **S**, ou si **X** n'a pas de successeur dans **S**.

Exemple : si **X**=20 et **S** = [10, **20, 30**, 40, 20, 50], à l'état final S = [10, **30, 20**, 40, 20, 50].
On a échangé la 1e valeur 20 de S avec son successeur (30).

La séquence **S** est donnée sous forme d'une liste chaînée (représentation standard) :

```
adCelE : type pointeur de CelE
CelE : type <E : entier ; Suc : adCelE>
T : adCelE                                   {tête de la liste traitée}
```

Les algorithmes suivants **doivent** procéder par **modification du chaînage**.

Q2 [8 points]

(i) Le premier élément de la liste a la valeur X

On étudie dans un premier temps le cas où **X** est la valeur du premier élément et où la liste a au moins deux éléments.

— **Dessiner le principe de modification du chaînage** dans ce cas (état initial : flèches pleines ; état final : flèches en pointillés) : faire apparaitre deux variables nommées **T** et **D** : la valeur de **T** est l'adresse du premier élément de la liste, la valeur de **D** est l'adresse du deuxième élément de la liste.

— **Donner l'algorithme** qui réalise la modification des liens de chainage entre **T** et **D**.

(ii) Le premier élément de valeur X n'est pas en tête et a un successeur

On étudie maintenant le cas où le premier élément de valeur **X** se situe ailleurs qu'en tête et a un successeur.

— **Dessiner le principe de modification** des liens de chaînage dans ce cas : faire apparaître deux variables nommées **AC** et **AP** : la valeur de **AC** est l'adresse du premier élément de valeur **X** et la valeur de **AP** est l'adresse du prédécesseur du premier élément de valeur **X**.

— **Donner l'algorithme** qui réalise la modification des liens de chainage entre **AP** et **AC**.

(iii) Algorithme complet

On traite maintenant l'algorithme complet de notre problème et on spécifie l'action suivante :

```
InverserOrdre : action (donnée X : entier, donnée-résultat T : adCelE)
    {Inverse, dans la liste de tête T, l'ordre entre le premier élément de valeur X et son
    successeur. Si X n'appartient pas à la liste ou si X n'a pas de successeur, la liste est
    inchangée. L'algorithme procède par modification du chaînage.}
```

— **Donner une réalisation** de l'action **InverserOrdre**, en **incluant tous les cas (dont les 2 premiers étudiés en (i) et (ii))**. Dans cette réalisation, **on ne recopiera pas** les réalisations données en **Q2i** et **Q2ii**, mais **on y fera référence** par les mentions "*Inverser comme en Q2i*" ou "*Inverser comme en Q2ii"*.

3. À propos de listes de listes (listes chaînées)

On considère des séquences de séquences d'entiers comme, **par exemple,** la suivante :

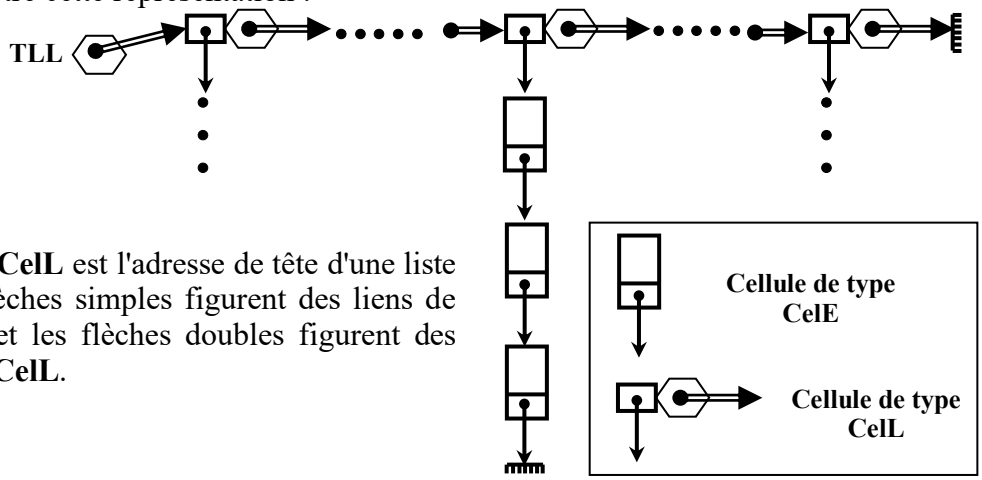
[[9, 5, 8], [], [0, -7, 5, 2], [2], [8,5]]

Ces séquences sont représentées par des listes chaînées de listes chaînées d'entiers :

```
{cellules pour listes d'entiers}
    adCelE : type pointeur de CelE
    CelE : type <E : entier, ESuc : adCelE >
```

{cellules pour listes de listes d'entiers}
adCelL : type pointeur de CelL
CelL : type <T : adCelE , LSuc : adCelL>

La figure suivante illustre cette représentation :



TLL de type **adCelL** est l'adresse de tête d'une liste de listes ; les flèches simples figurent des liens de type **adCelE** , et les flèches doubles figurent des liens de type **adCelL**.

Q3 [8 points]

(i) Nombre de listes de poids X

Le *poids* d'une liste d'entiers est la somme de ses éléments. On spécifie les fonctions suivantes :

Poids : fonction (T : adCelE) → entier ≥ 0
*{**Poids** de la liste donnée par son adresse de tête T. Le poids de la liste vide est 0.}*

NbDePoids : fonction (X : entier, TLL : adCelL) → entier ≥ 0
*{**Nombre de listes de poids** X dans la liste de listes donnée par son adresse de tête TLL.}*

Exemple : si la liste de tête **TLL** représente **[[9, -5, 8], [], [0, -7, 5, 2], [2], [8, 5]]** alors :
NbDePoids (13, TLL) = 1 car la dernière sous-liste est de poids 13.
NbDePoids (0, TLL) = 2 car la 2e et la 3e sous-liste sont de poids 0.
NbDePoids (-1, TTL) = 0 car aucune des sous-listes n'est de poids -1.

— **Donner une réalisation** de la fonction **Poids**.
— **Donner une réalisation** de la fonction **NbDePoids** en utilisant la fonction **Poids**.

(ii) Mise à plat d'une liste de listes d'entiers

La *mise à plat* d'une séquence **SS** de séquences d'entiers est une séquence **S** d'entiers formée par la concaténation des séquences d'entiers de la séquence **SS** donnée.

Exemple : la mise à plat de la séquence **[[9, 5, 8], [], [0, -7, 5, 2], [2], [8,5]]**
est la séquence **[9, 5, 8, 0, -7, 5, 2, 2, 8, 5]**.

On étudie une action de mise à plat par **modification des liens de chaînage** :
MettreAplat : action (donnée-résultat TLL : adCelL ; résultat TE : adCelE)
{Construit la mise à plat de la liste de listes d'entiers de tête TLL. L'algorithme procède par modification des liens de chaînage. À l'état final, TE est l'adresse de tête de la liste d'entiers construite, TLL vaut Nil , les cellules de type adCelE ont été déplacées de TLL vers TE et toutes les cellules de type adCelL ont été libérées.}

— **Donner une réalisation** de l'action **MettreAplat**.

M2 CCI - Algorithmique

Des exemples de solutions **décembre 2025**

1. . À propos de séquences d'entiers

Q1 "Présent une seule fois" 2 points

version 1 : on dénombre les exemplaires de E (schéma de parcours)

```
PrésentUneSeuleFois(E, T, L) :
    NbE : entier sur [1…Lmax]                    {nombre d'exemplaires de E}
    NbE ← 0
    pour  i allant de 1 a L :
        si E = Ti alors NbE ← NbE + 1
  retour : NbE=1          {remarque : on peut arrêter l'itération dès que NbE = 2.}
```

version 2 : on cherche E dans S et si on le trouve, on vérifie qu'il n'apparaît plus ensuite (schéma de recherche)

```
PrésentUneSeuleFois(E, T, L) :
    OK : booléen                                 {pour élaborer le résultat}
    i : entier sur [1…Lmax+1]
    i ← 1
    tant que i ≠ 1 + L et puis E ≠ Ti
        i ← i + 1
    si i = 1 + L alors OK ← faux
    sinon     i ← i+1
              tant que i ≠ 1 + L et puis E ≠ Ti
                  i ← i + 1
              OK ← i = 1+L
  retour : OK
```

version 3 : E est présent une seule fois ⇔ E est présent au moins une fois et E n'est pas présent deux fois" (schéma de parcours)

```
PrésentUneSeuleFois(E, T, L) :
    Présent_au_moins_une_fois, Pas_présent_deux _fois : booléen
    Présent_au_moins_une_fois ← faux
    Pas_présent_deux _fois ← vrai
    pour i allant de 1 a L :
          si E = Ti alors
              si Présent_au_moins_une_fois alors Pas_présent_deux_fois ← faux
              sinon Présent_au_moins_une_fois ← vrai
  retour : Présent_au_moins_une_fois et Pas_présent_deux_fois
```

"Tous présents une seule fois" 2 points

{On utilise PrésentUneSeuleFois, dans un schéma de parcours, ou dans un schéma de recherche.}

version 1 : parcours

```
TousPrésentsUneSeuleFois(TR, LR, TS, LS) :
    OK : booléen                                 {pour élaborer le résultat}
    OK ← vrai
    pour i allant de 1 a LR :
        OK ← OK et PrésentUneSeuleFois(TRi, TS, LS)
  retour : OK
```

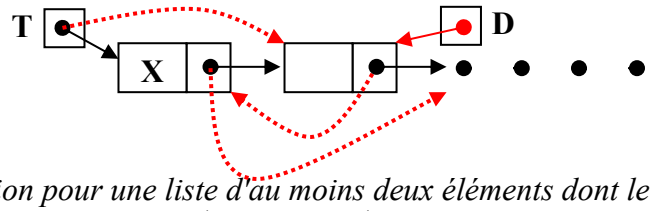
version 2 : recherche

```
TousPrésentsUneSeuleFois(TR, LR, TS, LS) :
    i : entier sur [1…Lmax+1]                                  {pour la recherche}
    i ← 1
    tant que i ≠ 1 + LR et puis PrésentUneSeuleFois(TRi, TS, LS) :
        i ← i+1
  retour : i = 1+LR
```

2. Inversion de l'ordre entre deux éléments consécutifs d'une liste chaînée

Q2

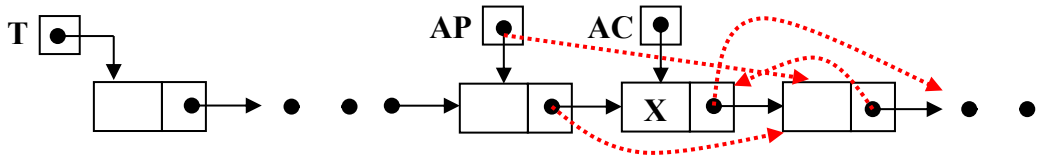
(i) Le premier élément de la liste a la valeur X et a un successeur 2 points



{pré-condition pour une liste d'au moins deux éléments dont le premier vaut X :
T ≠ Nil et puis (T↑.E = X et T↑.Suc ≠Nil)}

```
D : adCelE                                                     {adresse du deuxième}
D ← T↑.Suc
T↑.Suc ← D↑.Suc
D↑.Suc ← T
T ← D
```

(ii) Le premier élément de valeur X n'est pas en tête et a un successeur 2 points




```
AP ; AC : adCelE

AP↑.Suc ← AC↑.Suc
AP ← AC↑.Suc
AC↑.Suc ← AP↑.Suc
AP↑.Suc ← AC
```

(iii) Algorithme complet 4 points

```
InverserOrdre(X, L)
    AP, AC : adCelE              {pour la recherche : élément courant et son prédécesseur}
    {recherche du premier élément de valeur X, s'il existe, en maintenant le prédécesseur}
    AP ← Nil
    AC ← T
    tant que AC ≠ Nil et puis AC↑.E ≠ X
        AP ← AC
        AC ← AC↑.Suc
    si AC ≠ Nil alors            {X appartient à la liste}
        si AC↑.Suc ≠ Nil alors {X a un successeur}
            si AP = Nil alors Inverser comme en Q2i                {X est en tête}
            sinon Inverser comme en Q2ii                           {X n'est pas en tête}
```

3. À propos de listes de listes

Q3

(i) Nombre de listes de poids X 2 points Poids, 2 points NbDePoids

```
Poids (T) :
    ACE : adCelE                          {pour le parcours de la liste d'entiers courante}
    PC : entier ≥ 0                       {poids courant}
    ACE ←T
    PC ← 0
    tant que ACE ≠ Nil
        PC ← PC + ACE↑.E
        ACE ← ACE↑.ESuc
  retour : PC

  NbDePoids(X, TLL) :
  {Dans un parcours de la liste de listes, on compte le nombre de listes non vides de poids X
  (NbL). Le poids d'une liste est calculé par un simple parcours.}
    NbL : entier ≥ 0                      {nombre courant de listes de poids X}
    ACL : adCelL                          {pour le parcours de la liste de listes}
    ACL ← TLL
    NbL ← 0
    tant que ACL ≠ Nil
        si Poids(ACL↑.T)=X alors
            NbL ← NbL+1
        ACL ← ACL↑.LSuc
  retour : NbL
```

(ii) Mise à plat d'une liste de listes d'entiers 4 points

Le résultat est la concaténation des listes d'entiers non vides de la liste de listes donnée. Dans un *parcours* de cette liste de listes, le premier élément de la liste courante est connecté en queue du résultat partiel (adresse **Q**) et on met à jour cette adresse de queue (*recherche* du dernier de la liste courante). Les cellules de type **adCelL** sont libérées au fur et à mesure de ce parcours.

Pour ne pas traiter la première liste à part on munit temporairement le résultat d'un fictif de tête.

```
MettreAplat (TLL, TE) :
    F, Q : adCelE                 {adresse du fictif de tête et adresse de queue du résultat.}
    ACL : adCelL                  {pour la libération des cellules de type adCelL.}

    Allouer(F)                    {créer liste vide TE (avec fictif et queue)}
    Q ← F
    tant que TLL ≠ Nil
        si TLL↑.T ≠ Nil alors     {concaténation de la liste courante au résultat partiel}
            Q↑.ESuc ← TLL↑.T      {connexion en queue de TE}
            Q ← TLL↑.T            {mise à jour de la queue de TE}
            tant que Q↑.ESuc ≠ Nil :   { par recherche de la queue de la sous-liste}
                Q ← Q↑.ESuc
        ACL ← TLL                 {suppression en tête de TLL}
        TLL ← ACL↑.LSuc
        Libérer(ACL)
    Q↑.ESuc ← Nil
    TE ← F↑.ESuc                  {suppression du fictif et création de TE}
    Libérer(F)
```