Corrigé du TD 3

Exercice 1.

Question 1. *Rappeler la structure* `noeud` *et le type* `liste` *utilisés pour représenter une suite finie d'entiers par une liste simplement chaînée.*

```
Structure noeud
    valeur : entier
    suivant : pointeur sur noeud
type liste = pointeur sur noeud
```

Question 2. *Écrire une procédure qui prend en argument une liste* `L` *et retourne une copie de* `L`*.*

Voici une solution itérative. On stocke la copie en cours de construction dans la liste `N`. On utilise deux pointeurs auxiliaires : un pointeur `Q` pour parcourir la liste à copier `L`, et un pointeur `P` pour parcourir parallèlement la nouvelle liste `N`.

```
copieListeIt(L : liste) : liste
    si L == None alors
        retourner None
    N : liste ; N = Nouveau(noeud)
    N->valeur = L->valeur
    P : pointeur sur noeud ; P = N
    # boucle de parcours de L :
    Q : pointeur sur noeud ; Q = L
    tant que Q->suivant <> None faire
        # on rallonge la liste N :
        P->suivant = Nouveau(noeud)
        P->suivant->valeur = Q->suivant->valeur
        # on décale les deux pointeurs :
        P = P->suivant
        Q = Q->suivant
    # on marque la fin de N :
    P->suivant = None
    retourner N
```

Et voici la version récursive :

```
copieListeRec(L : liste) : liste
    si L == None alors
        retourner None
    N : liste ; N = Nouveau(noeud)
    N->valeur = L->valeur
    N->suivant = copieListeRec(L->suivant)
    retourner N
```

Question 3. *Écrire une procédure récursive qui prend en entrée une liste* `L` *et un entier* `x` *et retourne la liste* `L` *privée de toutes les occurrences de* `x`*.*

Cette question et la suivante sont plus difficiles...

```
supprimeListeRec(L : liste, x : entier) : liste
    si L == None alors
        retourner None
    si L->valeur <> x alors
        L->suivant = supprimeListeRec(L->suivant,x)
    sinon
        L = supprimeListeRec(L->suivant,x)
    retourner L
```

Question 4. *Donner une version itérative de cette procédure.*

```
supprimeListeIt(L : liste, x : entier) : liste
    si L == None alors
        retourner None
    si L->valeur == x alors
        L = L->suivant
    P : pointeur sur noeud ; P = L
    tant que P->suivant <> None faire
        si P->suivant->valeur == x alors
            P->suivant = P->suivant->suivant
        si P->suivant <> None alors
            P = P->suivant
    retourner L
```

Exercice 2. *On considère des listes d'entiers triés par ordre croissant (au sens large). On utilise des listes simplement chaînées. On souhaite fusionner deux listes triées* `L1` *et* `L2` *en une liste* `L`.

L'idée est de parcourir les deux listes `L1`, `L2` « simultanément » et d'ajouter à la liste fusionnée le plus petit des deux éléments « en cours » dans `L1` et `L2`.

Dans le version itérative ci-dessous on utilise trois pointeurs : `P` pour parcourir la liste `N` en cours de création, et `P1` et `P2` pour parcourir `L1` et `L2` (même si ce n'est pas strictement nécessaire : on pourrait déplacer directement les pointeurs `L1` et `L2`). La boucle se termine quand `P1` a atteint la fin de `L1` **et** `P2` a atteint la fin de `L2` — autrement dit, on continue à boucler tant que `P1` n'a pas atteint la fin de `L1` **ou** `P2` n'a pas atteint la fin de `L2`

On ajoute la valeur `P2->valeur` issue de la liste `L2` à la nouvelle liste dans le cas où `P2->valeur` est inférieur ou égal à `P1->valeur`, ce qui suppose qu'on n'a atteint ni la fin de `L1` ni celle de `L2`. On ajoute également `P2->valeur` quand on a atteint la fin de `L1`. Lorsqu'on a ajouté `P2->valeur` à `N` on peut déplacer le pointeur `P2` « vers la droite ». Si on est dans l'autre cas, on ajoute `P1->valeur` à `N` puis on déplace `P1`.

```
fusionListesTrieesIt(L1 : liste, L2 : liste) : liste
    si L1 == None et L2 == None alors
        retourner None
    N : liste ; N = Nouveau(noeud)
    P : pointeur sur noeud ; P = N
    P1 : pointeur sur noeud ; P1 = L1
    P2 : ponteur sur noeud ; P2 = L2
    tant que P1 <> None ou P2 <> None:
        si (P1 <> None et P2<>None et P2->valeur <= P1->valeur) ou P1 == None alors
            P->valeur = P2->valeur
            P->suivant = Nouveau(noeud)
            P2 = P2->suivant
        sinon
            P->valeur = P1->valeur
            P->suivant = Nouveau(noeud)
            P1 = P1->suivant
        P = P->suivant
    retourner N
```

Voici une version récursive.

```
fusionListesTrieesRec(L1 : liste, L2 : liste) : liste
    si L1 == None alors
        retourner copieListeRec(L2)
    si L2 == None alors
        retourner copieListeRec(L1)
    N : liste ; N = Nouveau(noeud)
    si L1->valeur <= L2->valeur alors
        N->valeur = L1->valeur
        N->suivant = fusionListesTrieesRec(L1->suivant, L2)
    sinon
        N->valeur = L2->valeur
        N->suivant = fusionListesTrieesRec(L1, L2->suivant)
    retourner N
```

À chaque appel récursif, ou à chaque exécution de la boucle, on crée un nœud de la nouvelle liste, et on compare au plus deux valeurs des listes passées en argument. Le coût maximal est égal à la longueur de la liste fusionnée, c'est-à-dire la somme des longueur des deux listes passées en argument.

Exercice 3.

Dans cet exercice, on code des entiers naturels strictement positifs comme des produits de puissances de facteurs premiers. Plus précisément, chaque entier est représenté par une structure de liste simplement chaînée où chaque nœud contient un facteur premier de cet entier et sa puissance. On supposera que la liste est ordonnée de façon croissante suivant les facteurs premiers.

Question 1. *Définir la structure* `noeudFacteur` *adéquate.*

```
Structure noeudFacteur
    facteur : entier
    puissance : entier
    suivant : pointeur sur noeudFacteur

type listeFacteurs = pointeur sur noeudFacteur
```

Question 2. *Écrire une procédure qui prend en entrée un entier codé par sa liste de facteurs et qui retourne la valeur de cet entier.*

Voici une solution itérative suivie d'une solution récursive. On utilise l'opérateur `a^b` qui calcule la puissance a^b. Dans la version itérative on pourrait utiliser directement le pointeur `L` (au lieu de `P`) pour parcourir la liste.

```
valeurIt(L : listeFacteurs) : entier
    resultat : entier ; resultat = 1
    P : pointeur sur noeudFacteur ; P = L
    tant que P <> None faire
        resultat = resultat * (P->facteur ^ P->puissance)
        P = P->suivant
    retourner resultat

valeurRec(L : listeFacteurs) : entier
    si L == None alors retourner 1
    retourner (L->facteur ^ L->puissance) * valeurRec(L->suivant)
```

Question 3. *Écrire une procédure qui retourne le plus grand facteur premier qui divise l'entier codé par* `N`*.*

Le plus grand facteur est le dernier de la liste, puisque les facteurs sont triés par ordre croissant. On le trouve simplement en parcourant la liste « presque » jusqu'au bout — il ne faut pas aller jusqu'à `L == None` sinon on ne peut plus accéder à `L->facteur`.

```
plusGrandFacteurRec(L : listeFacteurs) : entier
    si L == None alors retourner None
    si L->suivant == None alors retourner L->facteur
    retourner plusGrandFacteurRec(L->suivant)

plusGrandFacteurIt(L : listeFacteurs) : entier
    si L == None alors retourner None
    tant que L->suivant <> None faire
        L = L->suivant
    retourner L->facteur
```

Question 4. *Écrire une procédure qui retourne la* `listeFacteurs` *du produit des deux entiers codés par* `N1` *et* `N2`*.*

On s'inspire de la « fusion triée » de l'exercice 2. Dans le cas où un facteur premier figure à la fois dans `L1` et `L2` on additionne les puissances correspondantes.

```
produitRec(L1 : listeFacteurs, L2 : listeFacteurs) : listeFacteurs
    si L1 == None et L2 == None alors retourner None
    N : listeFacteurs ; N = Nouveau(noeudFacteur)
    si L1 == None ou (L2 <> None et L1->facteur > L2->facteur) alors
        N->facteur = L2->facteur
        N->puissance = L2->puissance
```

```
        N->suivant = produitRec(L1, L2->suivant)
    si L2 == None ou (L1 <> None et L1->facteur < L2->facteur) alors
        N->facteur = L1->facteur
        N->puissance = L1->puissance
        N->suivant = produitRec(L1->suivant, L2)
    si L1 <> None et L2 <> None et L1->facteur == L2->facteur alors
        N->facteur = L1->facteur
        N->puissance = L1->puissance + L2->puissance
        N->suivant = produitRec(L1->suivant, L2->suivant)
    retourner N
```

Exercice 4. Exercice non traité en TD.