Algorithmique avancée

—o000o– TD3 / TP3 –o000o—

Begaie - debegaie On dit qu'une liste est "bégayée" lorsque chaque élément y apparaît deux fois de suite. Le but de cet exercice est de transformer une liste en une liste bégayée, et inversement de "débégayer" une liste en supprimant un élément sur deux.

1. Écrire la fonction *i*térative `begaie` qui, étant donnée une liste d'éléments, renvoie la liste où chaque élément est répété. Par exemple :
   `begaie (1 2 3 1 4) = (1 1 2 2 3 3 1 1 4 4)` et `begaie (None) = None`.

2. Donner maintenant une fonction *r*écursive `begaieRec` qui effectue le même traitement que la fonction `begaie` précédente.

3. Écrire maintenant la fonction *i*térative `debegaie` qui prend en argument une liste bégayée et retourne la liste privée de ses doublons. La fonction `debegaie` est telle que `debegaie(begaie(L)) = L`.

4. Écrire enfin la fonction *r*écursive `debegaieRec` qui réalise le même traitement que la fonction `debegaie`.

De la suite dans les idées Le but de cet exercice est de construire des listes contenant les éléments d'une suite définie récursivement. Nous construirons les listes dans les deux sens possibles : si u_n désigne un terme de la suite, alors nous pouvons construire la liste $(u_n u_{n-1} ... u_0)$ ou la liste $(u_0 u_1 ... u_n)$.

1. Une suite géométrique u_n, $n \in N$ est définie récursivement par son premier terme u_0 et sa raison r par :

   - $u_n = u_0$ si $n = 0$
   - $u_n = r \times u_{n-1}$ sinon

   Écrire la fonction suivant qui, étant donnés deux nombres réels u_n et r, rend le terme suivant de u_n dans la suite géométrique de raison r. Par exemple :

   ```
   suivant (1, 4) = 4
   suivant (5, 2) = 10
   ```

2. Donner maintenant la fonction *i*térative `liste0aN(u0, r, n)` qui, étant donnés deux nombres u_0 et r ainsi qu'un entier naturel n, renvoie la liste $(u_0 u_1 ... u_n)$ où les u_i sont les éléments de la suite géométrique de raison r et de premier terme u_0.
   Par exemple :

```
liste0aN (1 2 5) = (1 2 4 8 16 32)
liste0aN (1 2 0) = (1)
liste0aN (3 1 4) = (3 3 3 3 3)
```

3. Proposer une version *r*écursive `liste0aNRec(u0, r, n)` de la fonction précédente.

4. Écrire la fonction *i*térative `listeNa0(u0, r, n)` qui, étant donnés deux nombres u_0 et r ainsi qu'un entier naturel n, renvoie la liste $(u_n...u_1u_0)$ où les u_i sont les éléments de la suite géométrique de raison r et de premier terme $u0$.
   Par exemple :

```
listeNa0 (1 2 5) = (32 16 8 4 2 1)
listeNa0 (1 2 0) = (1)
listeNa0 (3 1 4) = (3 3 3 3 3)
```

5. Proposer enfin une version *r*écursive `listeNa0Rec(u0, r, n)` de la fonction précédente.

Nombre d'occurrences du maximum On dispose d'une liste de nombres pour laquelle on souhaite pouvoir extraire le maximum (la plus grande valeur) ainsi que son nombre d'occurrences. Le but de l'exercice est de faire ce traitement en une seule passe sur la liste en utilisant une structure de couple pour contenir le résultat.

1. Écrire une fonction itérative `occmax(L)` qui prend en argument une liste L de nombres réels et retourne un couple contenant la valeur maximale atteinte dans la liste et son nombre d'occurrences. On considère que la liste n'est pas vide initialement.

2. Produire une solution récursive équivalente.

Implémentation de la fonction `zip` De très nombreux langages de programmations permettent de travailler sur des structures de listes chaînées. Le langage `Python` notamment permet de travailler nativement avec les listes et implémente la fonction `zip(l1, l2)` qui permet, à partir de 2 listes de même longueur de construire une liste contenant les couples de valeurs formés des éléments de même indice issu des 2 listes. Par exemple :

```
zip( (1, 2, 3), ("a", "b", "c")) = ((1 "a"), (2 "b"), (3 "c"))
```

1. Écrire la fonction itérative `zip(l1, l2)` qui permet d'obtenir le résultat désiré en suivant le formalisme des listes chaînées étudiées en cours. On supposera les listes en argument de même taille et non vides.

2. (Optionnel) Produire une solution récursive équivalente.

Liste croissante On considère dans cet exercice une liste contenant des valeurs comparables grâce à l'opérateur $\leq$.

1. Écrire la fonction itérative `croissante(L)` qui prend en argument une telle liste et retourne `True` si et seulement si la liste est triée par ordre croissant au sens large. On implémentera la fonction en considérant les cas de base suivants :
   - une liste vide est considérée comme triée par ordre croissant,
   - une liste avec un seul élément est croissante également ;
2. Proposer une solution récursive.

Suppression des occurrences d'un élément Nous nous intéressons maintenant à la suppression d'un élément quelconque dans une liste chaînée.

1. Écrire une fonction itérative `removeAll(L, elem)` qui prend en argument une liste chaînée `L` et une valeur `elem` et retourne la liste privée de ses éléments `elem`.
2. Proposer une solution récursive à ce problème.

Indications : pensez à différencier les cas suivants :

- la liste est vide initialement,
- la liste ne contient que l'élément à supprimer,
- la liste commence par les éléments à supprimer,
- la liste est mixte : composée pour une part de l'élément à supprimer et pour une autre part d'autres éléments à conserver.

Un classique : la suite de Fibonacci (en TP) La suite de Fibonacci est définie comme suit :

- $f_0 = 1$
- $f_1 = 1$
- $f_n = f_{n-1} + f_{n-2}$ si $n > 1$

1. Écrire la définition de la fonction itérative `fibonacci(n)` qui, étant donné un entier naturel n renvoie la liste $(f_n f_{n-1} f_{n-2} ... f_1 f_0)$ où les f_i sont les termes de la suite de Fibonacci. Par exemple :

```
liste-fibo (0) = (1)
liste-fibo (1) = (1 1)
liste-fibo (4) = (5 3 2 1 1)
liste-fibo (5) = (8 5 3 2 1 1)
```

2. Produire une version récursive de cette fonction

Interclassements (en TP) On s'intéresse dans cet exercice à deux méthodes d'interclassement de listes. L'interclassement vise à parcourir deux listes en parallèle et, à chaque itération, à insérer un élément venant de l'une ou de l'autre liste, dans la liste résultat. Le critère utilisé pour choisir la liste qui fournit l'élément à insérer suivant définit le type d'interclassement.

1. On s'intéresse tout d'abord à un interclassement strict qui insère alternativement un élément de la première liste puis un élément de la seconde liste. Dans le cas où une liste est plus courte que l'autre, les éléments restants de la plus longue sont insérés à la fin de la liste résultat sans interclassement. Plus précisément, si une liste est vide dès le début, la seconde est retournée. Si les deux listes sont vides, alors la liste vide (`None`) est retournée. Par exemple :

```
interStrict((1, 2, 3), (4, 5, 6, 7)) =
        (1, 4, 2, 5, 3, 6, 7)
interStrict(None, (4, 5, 6, 7)) = (4, 5, 6, 7)
interStrict(None, None) = None
```

   Écrire la fonction itérative `interStrict(L1, L2)` qui retourne un tel interclassement strict entre les listes `L1` et `L2`. Proposer ensuite une version récursive de cette fonction

2. On considère maintenant deux listes de nombres `L1` et `L2` qui sont triées par ordre croissant. Écrire une fonction d'interclassement `interCroissant(L1, L2)` qui insère à chaque itération l'élément le plus petit qui se trouve en tête de liste soit de `L1` soit de `L2`. Un tel interclassement produit une liste fusionnée elle-même triée par ordre croissant. Proposer une solution itérative puis une solution récursive à ce problème.

Réalisation d'une calculatrice (en TP) Nous nous intéressons à la conception d'une calculatrice qui repose sur une représentation des opérations à l'aide d'une pile. De façon à mener à bien ce programme, il va donc falloir :

- implémenter les fonctions de la barrière d'abstraction de la structure pile de données (push, pop),
- comprendre la notation postfixée pour la saisie des opérations

De manière traditionnelle, les opérations sont notées de manière infixée avec, dans le cas d'un opérateur binaire, le placement de l'opérateur entre les 2 opérandes. On obtient la notation suivante :

$$\text{opérande}_1 \ \textbf{opérateur} \ \text{opérande}_2$$

comme par exemple : $3 + 4$, $2.5 \ / \ 3.14 \ldots$

En notation postfixée, l'opérateur succède à son (dans le cas d'un opérateur unaire) ou ses

(dans le cas d'un opérateur binaire) opérande(s). Ainsi : "3+4" s'écrit en postfixe "3 4 +".

L'intérêt d'une telle représentation est de pouvoir ensuite utiliser une pile pour réaliser les calculs en suivant le schéma de résolution suivant :

- si la pile ne contient qu'une valeur réelle, il s'agit du résultat du calcul,
- sinon on empile les valeurs réelles lorsqu'elles sont lues en entrée,
- si on lit un caractère correspondant à un opérateur binaire on dépile les deux dernières valeurs réelles, on effectue le calcul et on empile le résultat,
- si on lit un caractère correspondant à un opérateur unaire, comme l'opérateur de négation, on dépile la dernière valeur, on applique l'opérateur et on empile le résultat.

Attention : la division est une loi non commutative, c'est-à-dire que $x/y \neq y/x$. Il faut donc garder à l'esprit qu'en écrivant "x y /" en postifixé, l'opération souhaitée est $\frac{x}{y}$ mais que du fait de l'utilisation d'une pile, la valeur de y sera dépilée avant celle de x (pile = dernier arrivé, premier sorti).
Dans le cadre de notre programme, les opérateurs suivants doivent être reconnus :

- `+` : addition, `-` : soustraction, `*` : multiplication, `/` division et `n` : négation

Similairement, on veillera à bien gérer les erreurs :

- soit de division par 0,
- soit de dépilement de valeur inexistante : par exemple, essayer de dépiler les deux opérandes d'un opérateur `+` alors qu'il n'y en a qu'un seul dans la pile.

1. Indiquer la traduction en notation postfixée des expressions suivantes exprimées en notation infixée (les parenthèses ne sont là que pour vous aider à comprendre la priorité des opérations) :
   - (2 + 4) * n 5
   - 9 / (3.5 + 6)
   - (3 - 5) / (3 + 5)
2. Implémenter une calculatrice simple en notation postfixée sur la base d'une pile gérée par votre barrière d'abstraction.

Vous pourrez utiliser la fonction Python `input(chaine)` qui permet d'afficher l'invite de saisie contenue dans `chaine` et de récupérer la valeur saisie dans une variable de votre choix. Vous pourrez aussi avoir besoin de la fonction `float(c)` qui traduit une chaine en son nombre réel correspondant lorsque cela est possible.

Enfin, vous pouvez définir autant de fonctions qui vous semblent nécessaires pour réaliser cette calculatrice.