STRUCTURES DE DONNEES

LISTES, PILES et FILES

L'écriture sur des exemples simples de plusieurs implémentations d'une même structure de données permet de faire émerger les notions d'**interface** et d'**implémentation**, ou encore de **structure de données abstraites**.

Le paradigme de la programmation objet peut être utilisé pour réaliser des implémentations effectives des structures de données, même si ce n'est pas la seule façon de procéder.

1. Généralités

Les algorithmes opèrent sur des **données** qui peuvent être de **différentes natures**. Un algorithme doit donc être autant que possible **indépendant d'une implémentation particulière**, c'est-à-dire qu'il doit pouvoir s'appliquer quelle que soit la représentation de données.

Les données sont dites alors **abstraites** ; pour décrire ces données on définit un ensemble d'opérations que l'on pourra leur appliquer, sans oublier les propriétés liées à ces opérations.

D'où la qualification de **type abstrait de données**.

Cette notion de type abstrait de données va nous permettre de définir des types de données non « primitifs », c'est à dire non disponibles dans les langages de programmation courants.

En classe de première nous avons vu différentes façons de représenter des données. Par exemple, les entiers (**int**), les réels (**float**), les chaines de caractères (**str**) et les booléens (**bool**). Ce sont ces types de données définies dans les langages de programmation qui sont qualifiés de « primitifs ».

2. Notion d'interface

Une structure de données possède un ensemble de procédures appelées « **routines** » permettant, de lire, d'ajouter, d'effacer, d'accéder aux données.
C'est cet ensemble de routines que l'on appelle « **interface** »
Une interface est généralement constituée de 4 routines élémentaires dites **CRUD** :

- **C**reate : ajout de donnés.
- **R**ead : lecture de donnée.
- **U**pdate : modification d'une donnée.
- **D**elete : suppression d'une donnée.

3. Les listes

3.1 Qu'est qu'une liste ?

Nous savons tous intuitivement ce qu'est une liste. Une liste est une collection finie d'éléments qui se suivent. C'est donc une structure de données **séquentielle** ou **linéaire**.

Une liste peut contenir un nombre quelconque d'éléments y compris « null » (la liste vide).

Prenons une liste comme par exemple : **ℓ1 = [3,1,4]**

C'est une liste à trois éléments (ou de longueur trois) dont le premier est 3, le deuxième 1, et le dernier 4.

Une autre façon de décrire cette liste consiste à dire que la liste **ℓ1** possède **un premier élément** « 3 » qu'on nommera élément de **tête**, et que vient après cet élément de tête la liste ℓ2 = [1,4] des éléments qui suivent, cette 2ème liste ℓ2 est appelée **reste**.

Ce qu'on vient de dire de la liste ℓ1 peut être répété pour la liste ℓ2 qui est donc constituée d'un élément de **tête** « 1 » et d'un **reste** : ℓ3 = [4].

À nouveau on peut répéter le même discours pour la liste ℓ3 qui est donc constituée d'un élément de **tête** « 4 » et d'un **reste** : ℓ4 = [ ].

La liste ℓ4 étant vide, elle ne possède pas d'élément de tête, et ne peut donc pas être décomposée comme nous venons de le faire à trois reprises.

Si on convient d'utiliser la notation (x,ℓ) pour désigner le couple constitué de l'élément x de tête, et du reste ℓ d'une liste, on peut alors écrire : **ℓ1 = (3, (1, (4, [ ])))**

On conçoit aisément que ce qui vient d'être fait pour notre exemple de liste ℓ1 peut être reproduit pour n'importe quelle liste.

On peut conclure cette approche en donnant une définition **abstraite** et **formelle** des listes d'éléments appartenant tous à un ensemble E.

Une liste d'éléments d'un ensemble E est soit la liste vide, soit un couple (x,ℓ) constitué d'un élément x ∈ E et d'une liste ℓ d'éléments de E.

Il ressort de cette définition que les listes peuvent être vues comme des structures de **données récursives**.

Remarque :
En classe de première, nous avons manipulé des « listes » en python. Il s'agissait en fait d'un **tableau dynamique**, le type liste de python ne correspond pas à la structure de données liste que nous voyons dans ce cours.

3.2 Types d'opérations possibles sur une liste existante.

- INSERER(L, x, i)
  Cette opération repère la donnée de rang (i-1), décale de 1 rang vers la droite toutes les données situées derrière la donnée de rang (i-1), et insère la donnée « x » au rang i.

- SUPPRIMER(L, i)
  Cette opération repère la donnée de rang (i), puis décale de 1 rang vers la gauche tous les éléments situés derrière cette donnée de rang (i)

3.3 Autres opérations.

Voici d'autres opérations possibles sur une liste. En vous aidant de ce qui a été fait ci-dessus, expliquez ce que font ces opérations.

- CREER-LISTE_VIDE()

- RECHERCHER(L, e)

- LIRE(L,i)

- MODIFIER(L,i,e)

- LONGUEUR(L)

3.4 Exemple d'application.

Soit la suite d'instructions suivantes :

```
L=CREER_LISTE_VIDE()
INSERER(L,'A', 1)
INSERER(L,'O', 2)
INSERER(L,'B', 1)
INSERER(L,'V', 3)
INSERER(L,'R', 2)
```

Ecrire le résultat de la liste obtenue à chacune de ces instructions.

Instructions	Résultat
L=CREER_LISTE_VIDE()	
INSERER(L,'A', 1)	
INSERER(L,'O', 2)	
INSERER(L,'B', 1)	
INSERER(L,'V', 3)	
INSERER(L,'R', 2)	

Que contient la liste créée à la fin de ces instructions ?

3.5 Représentation d'une liste avec un tableau

3.5.1 Exemple.

Pour concevoir une liste, on peut utiliser un tableau dont chaque élément est identifié par son indice.
Nous allons créer une liste de n éléments avec un tableau pouvant contenir (n+1) éléments.

La première case du tableau indique le nombre d'éléments de la liste.
Les autres cases du tableau contiennent les éléments de la liste.

Exemple : Voici une liste créée sous la forme d'un tableau.

3	N	S	I		
0	1	2	3	4	5

Proposez 2 suites d'instructions permettant de créer cette liste en utilisant l'instruction INSERER(L,x,i).
Vous présenterez votre réponse sous la forme d'un tableau comme nous l'avons fait dans la question 3.4, avec une 1ère colonne qui indique les instructions, et une 2ème colonne qui indique le résultat attendu après l'instruction.

1ère méthode :

Instructions	Résultat

2ème méthode

Instructions	Résultat

Cela montre que l'on peut aboutir au même résultat de plusieurs manières, mais ces différentes façons ne sont pas équivalentes au niveau de l'accès mémoire.

3.5.2 Pseudo-code pour insérer et supprimer les éléments d'une liste.

Voici le pseudo-code qui permet d'insérer et de supprimer un élément d'une liste.

```
Fonction INSERER(L, x, i) :
    Si (L[0] == longueur(L)-1 OU  (i-1 > L[0]) :
        Ecrire "La liste est déjà pleine ou alors le rang n'est pas correct !"
        Retourner Faux
    Sinon :
        Pour k allant de (L[0]+1) à (i) par pas de (-1)
            L[k] = L[k-1]
        L[i] = x
    L[0] = L[0] + 1
    Retourner Vrai
```

```
Fonction SUPPRIMER(L, i) :
    Si (L[0] != 0)  ET  (i < =L[0]) :
        Pour k allant de i à (L[0]) par pas de 1
            L[k] = L[k+1]
        L[0] = L[0]-1
        Retourner Vrai
    Sinon :
        Ecrire "La liste est déjà vide ou alors le rang n'est pas correct !"
        Retourner Faux
```

3.5.3 Implémentation en python.

Ecrire en python les deux fonctions ci-dessus.

Puis ajouter le programme principal ci-dessous et analyser son exécution.

```
Nb_Element = 5
Liste=[None]*(Nb_Element+1)
print(Liste)
Liste[0]= 0
print(Liste)
INSERER(Liste, 'N', 1)
print(Liste)
INSERER(Liste, 'I', 2)
print(Liste)
INSERER(Liste, 'I', 3)
print(Liste)
SUPPRIMER(Liste,2)
print(Liste)
INSERER(Liste,'S',2)
print(Liste)
INSERER(Liste, 2020, 4)
print(Liste)
INSERER(Liste, 2021, 5)
print(Liste)
```