STRUCTURE de DONNEES ABSTRAITES

Rappels :

- En 1ère on a découvert les types de données primitifs : les entiers, les flottants, les booléens, le chaines de caractère.
- Puis ensuite on a vu des types construits qui sont des « conteneurs » des données de type primitifs : les listes, les tuples, les dictionnaires

Définitions :

- **Une structure de données** est une manière de stocker, d'accéder à,..,et de manipuler des données
- **L'interface** de la structure de données décrit de quelle manière on peut la manipuler. Elle spécifie la **nature des données** ainsi que **l'ensemble des opérations permises sur la structure**.
- **L'implémentation** :de la structure de données, au contraire, contient le code de ces méthodes (comment fait Python). Il n'est pas nécessaire de connaître l'implémentation pour manipuler la structure de données
- **Un type** abstrait décrit essentiellement l'interface, indépendamment du langage de programmation.

On parle de structure de données **abstraites** ou de type **abstrait** de données car au niveau de l'interface les données, leurs liens et les opérations permises sont caractérisées mais on ne sait pas (et on ne veut pas savoir) comment c'est fait concrètement (implémentation).

Les opérations usuelles d'une Structure de Données Abstraites (SDA) sont :

- **Créer** une donnée éventuellement vide, en utilisant ce qu'on appelle un constructeur.
- **Accéder** à un élément :
  - soit ceux directement repérables par la structure (le premier d'une séquence, ou associé à une clé donnée, ...)
  - soit à partir d'un élément préalablement repéré (successeur d'un élément donné, ...)
- **Ajouter** un élément, en précisant comment il s'intègre dans l'organisation globale de la collection.
- **Retirer** un élément, en précisant comment ceux qui lui étaient liés se réorganisent.
- Éventuellement, des opérations plus **avancées** (rechercher un élément, trier la collection, fusionner deux collections, ...)

Chaque opération doit être bien spécifiée (entrée, sorties, précondition) : c'est ce que l'on fait dans l'interface.

Les structures de données abstraites que nous étudierons cette année peuvent être classées selon la nature de l'organisation de la collection de données :

- Structures **linéaires** (ou séquentielles) : il y a un premier élément et un dernier ; chaque élément a un prédécesseur (sauf le premier) et un successeur (sauf le dernier). Exemples : *liste, file, pile*.
- Structures **associatives** : les éléments sont repérés par une clé ; ils n'ont pas de lien entre eux. Exemples : *dictionnaires, ensembles.*
- Structures **hiérarchiques** : il y a un (parfois plusieurs) élément racine ; chaque élément dépend d'un antécédent (sauf la/les racine/s) et a des descendants (sauf les feuilles). Exemple : *arbre.*
- Structures **relationnelles** : chaque élément est en relation directe avec des voisins, ou bien a des prédécesseurs et des successeurs. Exemple : *graphe.*

Ces structures de données sont parfois implémentées nativement dans les langages de programmation comme type de données mais ce n'est pas toujours le cas. En Python, les listes (type list), les dictionnaires (type dict) et les ensembles (type set) le sont mais pas les autres.

Un exemple possible de type abstrait

Type Abstrait Liste

Type abstrait : Liste

Données : éléments de type T

Opérations

CREER_LISTE_VIDE() qui retourne un objet de type Liste

La liste existe et elle est vide.

INSERER(L, e, *i*)

L'élément e est inséré à la position i dans la liste L.

SUPPRIMER(L, *i*)

L'élément situé à la position i est supprimé de la liste L.

RECHERCHER(L, e) qui retourne un objet de type Entier

L'élément e est cherché dans la liste L et on retourne son index (sa position).

LIRE(L, *i*) qui retourne un objet de type T

L'élément situé à la position i dans la liste L est retourné.

MODIFIER(L, *i*, e)

L'élément situé à la position i dans la liste L est écrasé par le nouvel élément e.

LONGUEUR(L) qui retourne un objet de type Entier

Le nombre d'éléments présents dans la liste L est retourné.

Conditions

LIRE(L, i) est défini si et seulement si $1 \leq i \leq LONGUEUR(L)$

MODIFIER(L, i, e) est défini si et seulement si $1 \leq i \leq LONGUEUR(L)$

SUPPRIMER(L, i) est défini si et seulement si $1 \leq i \leq LONGUEUR(L)+1$

INSERER(L, e, i) est défini si et seulement si $1 \leq i \leq LONGUEUR(L)$

INTERFACE

Exemple d'application de ce type abstrait

Prenons la suite d'instructions suivantes :

```
L = CREER_LISTE_VIDE()
INSERER(L, 'A', 1)
INSERER(L, 'O', 2)
INSERER(L, 'B', 1)
INSERER(L, 'V', 4)
INSERER(L, 'R', 2)
```

On voit assez aisément qu'à la fin, la liste L contient 5 éléments de type « caractère » et L = ('B', 'R', 'A', 'V', 'O').