

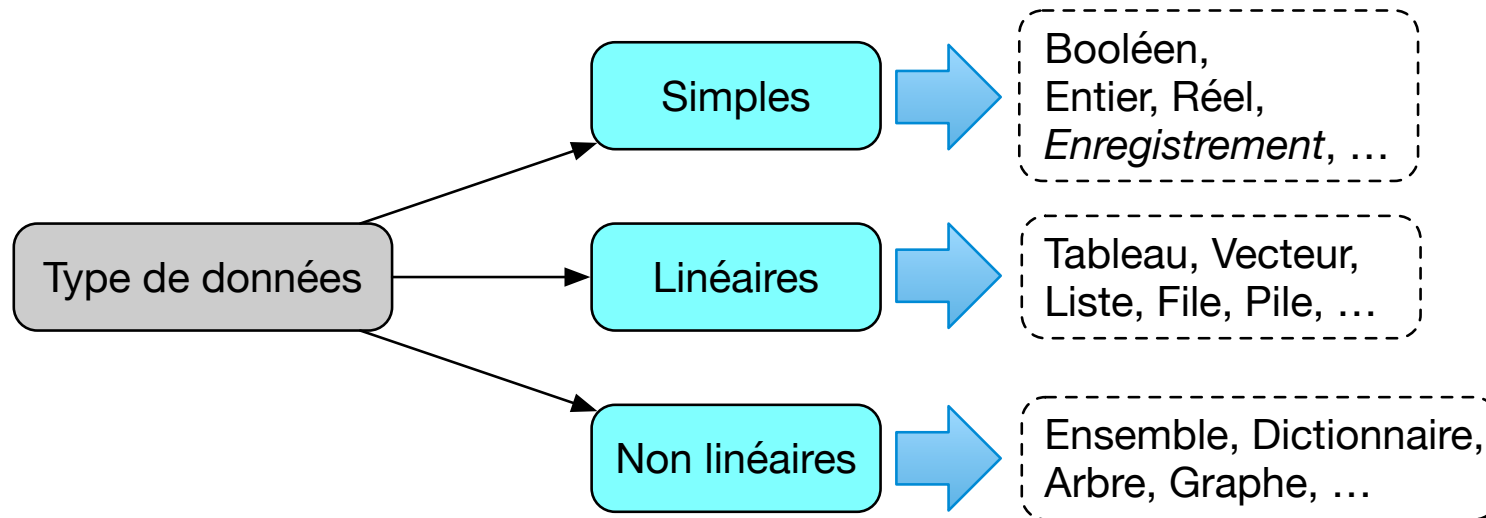
Structures De Données et Types de Données Abstraits

Sébastien Jean

IUT de Valence
Département Informatique

v1.0, 11 novembre 2025

Type de données



- **Identification d'une information** par un ensemble de valeurs (domaine) et d'opérations
 - **Simple** : une donnée est une **valeur unique** (*atomique*)
 - **Linéaires** : une donnée est une **collection de valeurs**, organisée **en séquence**
 - **Non linéaires** : une donnée est une **collection de valeurs**, organisée **sans lien séquentiel**

Le cas particulier des enregistrements

- Un **enregistrement** est un **type de données** qui peut être considéré comme simple ou non linéaire
- Il est **composé de plusieurs valeurs** appelées **champs** ou **membres**
 - Le **nombre de champs** est **fixe**
 - Les champs sont **nommés** et peuvent être de **type différent**
 - Les **opérations** se limitent à la **lecture et l'affectation des champs**
 - **Accès** aux champs via la **notation pointée** (`variable.champs`)
- Exemples :
 - Une **date** composée de 3 valeurs entières (jour, mois, année)
 - Un **point** composé de 2 valeurs réelles (abscisse, ordonnée)

Exemple d'enregistrement : Date

- Nom : Date
- Champs :
 - jour : entier
 - mois : entier
 - annee : entier

```
FONCTION est_valide (d : Date) : booléen
```

```
    SI d.jour < 0 OU d.mois < 0 ALORS  
        RETOURNER FAUX
```

```
    ...
```

```
FIN FONCTION
```

Type de données linéaires et non linéaires usuels

- N.B. : Lorsque l'on parle de **collections**, on fait une distinction entre :
 - **Statique** : le nombre de valeurs est fixe
 - **Dynamique** : le nombre de valeurs évolue
- **Chaîne de caractères**
 - Sequence ordonnée de caractères (dynamique)
- **Tableau, Vecteur**
 - Collection de valeurs repérées par leur position, statique (tableau) ou dynamique (vecteur)

Type de données linéaires et non linéaires usuels (suite)

- Liste, File, Pile
 - Groupe de valeurs organisées par une relation prédécesseur/successeur, avec notion de début et fin
 - Politique d'accès
 - Ajout/retrait n'importe où pour une liste
 - Ajout au début, retrait au début pour une pile (*LIFO : Last In First Out*)
 - Ajout en fin, retrait au début pour une file (*FIFO : First In First Out*)

Type de données linéaires et non linéaires usuels (suite)

- Dictionnaire

- Groupe de paires clé/valeur, où les clés sont uniques

- Ensemble

- Groupe de valeurs sans duplicata, sans notion de position

- Arbre

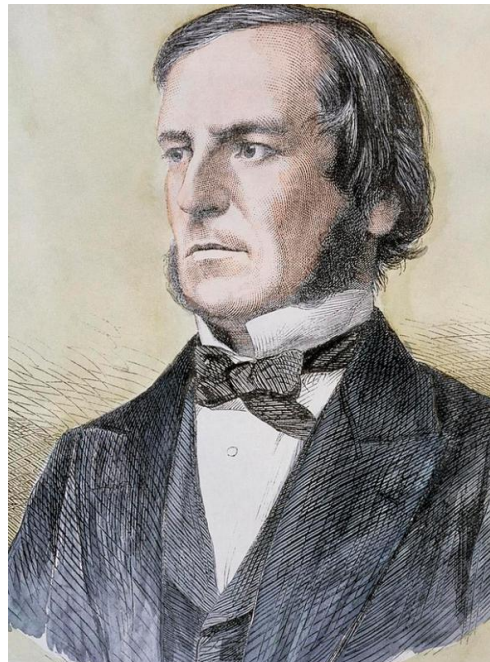
- Groupe de noeuds où chaque noeud contient une valeur et est associé à d'autres noeuds par une relation parent/enfant

- Graphe

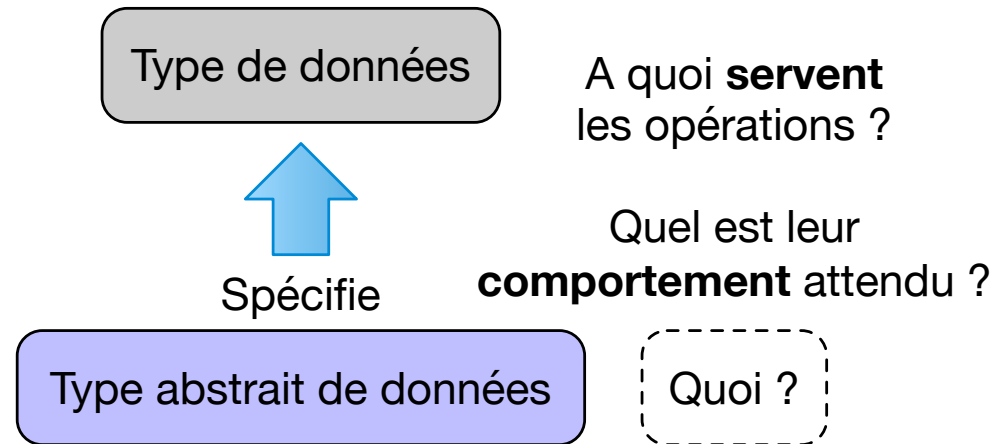
- Groupe de noeuds où chaque noeud contient une valeur et est associé à d'autres noeuds par une relation dirigée et valuée

Type de données : Booléen

- Domaine :
 - {VRAI, FAUX}
- Opérations :
 - ET, OU, NON (cf. algèbre de Boole)

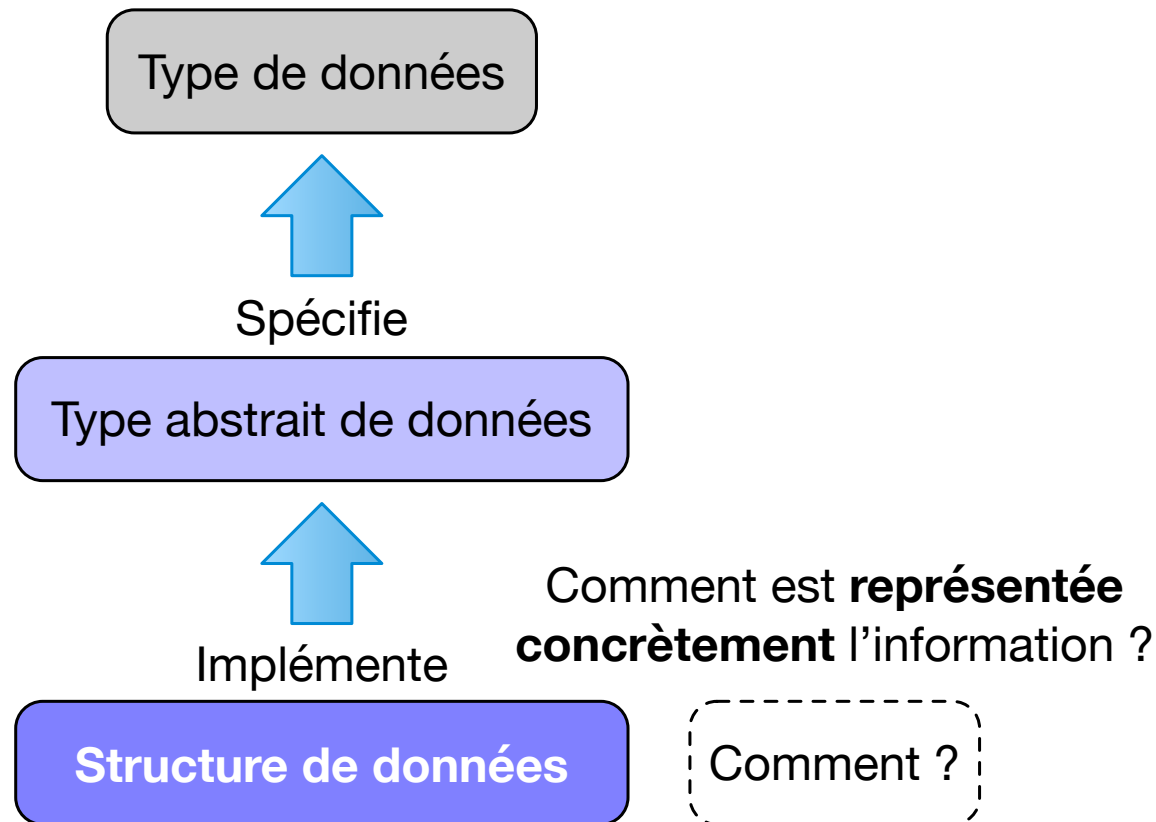


Types de Données Abstrait (TDA)



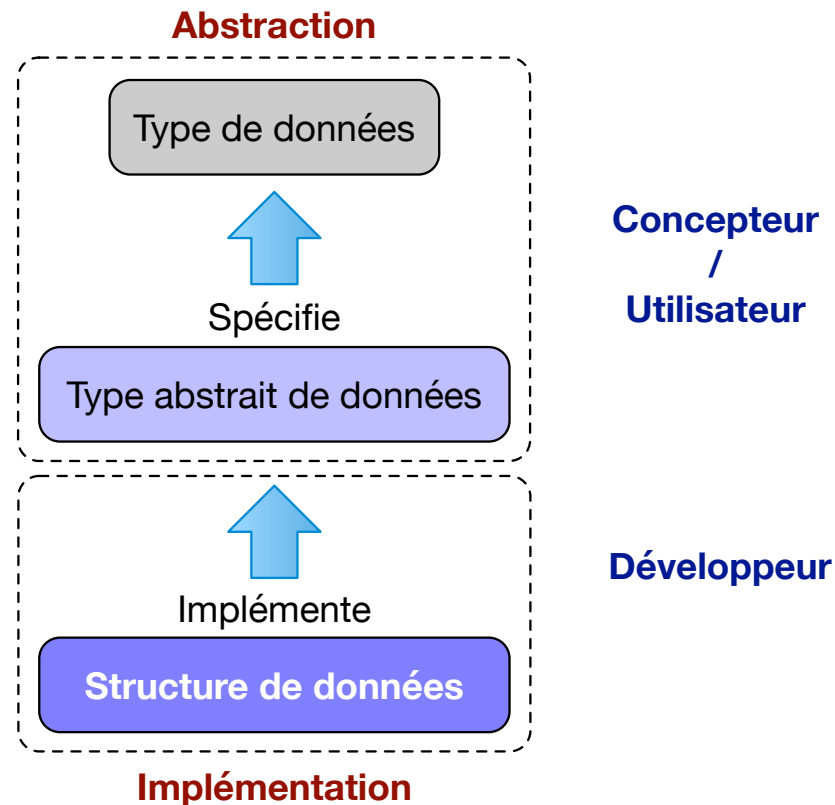
- **Spécification mathématique** d'un **type de données**
 - Nom
 - Dépendances
 - Opérations
 - Pré-conditions
 - Axiomes

Structure De Données (SDD)



- **Implémentation concrète** (langage / architecture) d'un TDA
 - N.B. : certains TDA simples (ex. booléens, entiers, flottants) se projettent sur des *types machine*

Barrière de l'abstraction



- Les TDA **masquent la réalité de l'implémentation** aux **utilisateurs**
- Pour un **même TDA**, **plusieurs SDD** peuvent être définies
 - Complexité différente pour les opérations par ex.
 - **Algorithmes** écrits **indépendamment** d'une **implémentation**

TDA : opérations

- 3 catégories :
 - **Constructeurs** : créent une donnée parmi les possibles
 - **Transformateurs**
 - **Producteurs** : produisent une nouvelle donnée à partir d'une ancienne
 - **Mutateurs** : modifient une donnée
 - **Observateurs** : donnent des informations sur la donnée
- **Syntaxe** de la description d'une opération
 - $\text{nom} : \text{type operand1} \times \dots \times \text{type operandN} \rightarrow \text{type résultat}$

TDA : Booléen

- Nom : Booléen
- Dépendances : (*aucune*)
- Opérations :
 - Constructeurs :
 - vrai : \rightarrow Booléen
 - faux : \rightarrow Booléen
 - Transformateurs :
 - non : Booléen \rightarrow Booléen
 - et : Booléen \times Booléen \rightarrow Booléen
 - ou : Booléen \times Booléen \rightarrow Booléen
- A suivre ...

TDA : pré-conditions et axiomes

- **Pré-conditions**

- Conditions devant être satisfaites par les arguments des opérations

- **Axiomes**

- Propositions logiques vraies décrivant le comportement d'une opération

TDA : Booléen (suite)

- Pré-conditions : (*aucune*)

- Axiomes :

- $\text{non}(\text{vrai}()) = \text{faux}()$
- $\text{non}(\text{faux}()) = \text{vrai}()$
- $\text{et}(a, b) = \text{et}(b, a)$
- $\text{ou}(a, b) = \text{ou}(b, a)$
- $\text{et}(\text{vrai}(), a) = a$
- $\text{et}(\text{faux}(), a) = \text{faux}()$
- $\text{ou}(\text{vrai}(), a) = \text{vrai}()$
- $\text{ou}(a, \text{faux}()) = a$

TDA et types machines

- Dans notre *pseudo-code*, les booléens, les entiers, les flottants et les caractères sont considérés comme des **types préexistants** et "**primitifs**"
 - On a fait l'hypothèse (parce que c'est presque toujours le cas) qu'ils se projettent sur des types "machines", et que les **opérations deviennent des opérateurs**
- Un **booléen** a une **représentation immédiate en mémoire** (sur très peu de place), l'opération **et** à 2 paramètres est remplacée par l'opérateur **ET** à 2 opérandes
- ...

Type de données : Chaîne de caractères

- Type de données **linéaire**, **dynamique**
- **Domaine :**
 - Toutes les **séquences de caractères affichables** (chiffres, lettres, symboles)
- **Opérations :**
 - **ajouter un caractère (affichable) en fin de chaîne**
 - **connaitre le nombre de caractères**
 - **retrouver un caractère à une position donnée**
 - **concaténer 2 chaînes**
 - **savoir si 2 chaînes sont égales**
 - ...

TDA : Chaîne de caractères

- Nom : **Chaîne**
- Dépendances : *Caractère, Entier, Booléen*
- Opérations :
 - Constructeurs :
 - **creer** : \rightarrow **Chaîne** (vide)
 - Transformateurs :
 - **ajouter** : $\text{Caractère} \times \text{Chaîne} \rightarrow \text{Chaîne}$
 - **concatener** : $\text{Chaîne} \times \text{Chaîne} \rightarrow \text{Chaîne}$
 - Observateurs :
 - **taille** : $\text{Chaîne} \rightarrow \text{Entier}$
 - **egal** : $\text{Chaîne} \times \text{Chaîne} \rightarrow \text{Booléen}$
 - **caractere** : $\text{Entier} \times \text{Chaîne} \rightarrow \text{Caractère}$
- *A suivre ...*

TDA : Chaîne (suite)

- Pré-conditions :

- $\text{caractere}(i, c) \rightarrow 1 \leq i \leq \text{taille}(c)$

- Axiomes :

- $\text{taille}(\text{creer}()) = 0$
 - $\text{taille}(\text{ajouter}(k, s)) = \text{taille}(s) + 1$
 - $\text{taille}(\text{concatener}(s1, s2)) = \text{taille}(s1) + \text{taille}(s2)$
 - $\text{caractere}(1, \text{ajouter}(k, \text{creer}())) = k$
 - $\text{caractere}(i, \text{ajouter}(k, s)) = k$ si $\text{taille}(s) = i - 1$
 - $\text{caractere}(i, \text{ajouter}(k, s)) = \text{caractere}(i, s)$ si $\text{taille}(s) > i - 1$
 - $\text{egal}(s1, s2) = \text{faux}()$ si $\text{taille}(s1) \neq \text{taille}(s2)$
 - $\text{egal}(\text{creer}(), \text{creer}()) = \text{vrai}()$
 - $\text{egal}(\text{ajouter}(k, s1), \text{ajouter}(k, s2)) = \text{egal}(s1, s2)$

TDA : Chaîne (fin)

- Certaines **opérations** sont **dépendantes de l'implémentation**
 - Elles nécessitent de connaître la **structure de données** pour être écrites
- D'autres **opérations** sont **indépendantes de l'implémentation**
 - Elles peuvent être **écrites sans connaître l'implémentation**, mais simplement en **s'appuyant sur les opérations dépendantes de l'implémentation**



- Quid des opérations du type Chaîne ?

Fin !

