Structures de données

Cours de spécialité NSI de Terminale

D Pihoué

Lycée Camille Jullian Bordeaux

15 septembre 2023

Capacités attendues

1. Spécifier une structure de données par son interface.
2. Distinguer interface et implémentation.
3. Écrire plusieurs implémentations d'une même structure de données.
4. Choisir une structure de données adaptée à la situation à modéliser.
5. Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.

La résolution de certains problèmes nécessite l'implémentation de structures de données plus élaborées que celles fournies nativement par les langages de programmation.

La résolution de certains problèmes nécessite l'implémentation de structures de données plus élaborées que celles fournies nativement par les langages de programmation.

Elles sont conçues à partir des types fournis par les langages envisagés comme des briques élémentaires.

La résolution de certains problèmes nécessite l'implémentation de structures de données plus élaborées que celles fournies nativement par les langages de programmation.

Elles sont conçues à partir des types fournis par les langages envisagés comme des briques élémentaires.

On parle ainsi de **type abstrait de données**.

La résolution de certains problèmes nécessite l'implémentation de structures de données plus élaborées que celles fournies nativement par les langages de programmation.

Elles sont conçues à partir des types fournis par les langages envisagés comme des briques élémentaires.

On parle ainsi de **type abstrait de données**.

Ils sont **abstraits** car ils ne dépendent pas du langage de programmation.

Caractérisation

Un type abstrait de données est caractérisé par une **interface** de programmation constituée de l'ensemble des **opérations** réalisables avec les données de ce type abstrait.

Caractérisation

Un type abstrait de données est caractérisé par une **interface** de programmation constituée de l'ensemble des **opérations** réalisables avec les données de ce type abstrait.

On distingue quatre catégories d'opérations.

Définition (Catégories d'opérations)

- Les **constructeurs** pour créer une nouvelle structure de données.

Définition (Catégories d'opérations)

- Les **constructeurs** pour créer une nouvelle structure de données.
- Les **opérateurs** pour modifier une structure de données comme ajouter ou supprimer.

Définition (Catégories d'opérations)

- Les **constructeurs** pour créer une nouvelle structure de données.
- Les **opérateurs** pour modifier une structure de données comme ajouter ou supprimer.
- Les **accesseurs** pour donner de l'information sur la structure comme la valeur d'une donnée ou le nombre des éléments qui la composent.

Définition (Catégories d'opérations)

- Les **constructeurs** pour créer une nouvelle structure de données.
- Les **opérateurs** pour modifier une structure de données comme ajouter ou supprimer.
- Les **accesseurs** pour donner de l'information sur la structure comme la valeur d'une donnée ou le nombre des éléments qui la composent.
- Les **itérateurs** pour énumérer les éléments de la structure de données.

Précisons la définition de deux démarches systématiquement réalisées.

Précisons la définition de deux démarches systématiquement réalisées.

Définition

- **Spécifier** un type abstrait de données c'est définir son **interface**.

Précisons la définition de deux démarches systématiquement réalisées.

Définition

- **Spécifier** un type abstrait de données c'est définir son **interface**.
- **Implémenter** un type abstrait de données c'est fournir le code des **opérations** qui constituent son **interface**.

Précisons la définition de deux démarches systématiquement réalisées.

Définition

- **Spécifier** un type abstrait de données c'est définir son **interface**.
- **Implémenter** un type abstrait de données c'est fournir le code des **opérations** qui constituent son **interface**.

Plusieurs implémentations peuvent répondre à une même spécification.

Précisons la définition de deux démarches systématiquement réalisées.

Définition

- **Spécifier** un type abstrait de données c'est définir son **interface**.
- **Implémenter** un type abstrait de données c'est fournir le code des **opérations** qui constituent son **interface**.

Plusieurs implémentations peuvent répondre à une même spécification.
La personne qui utilise un type abstrait de données n'a besoin de connaître que son interface.

Principes

Distinguer interface et implémentation permet notamment de

- séparer la conception des tests de celle du code des opérations,
- modifier l'implémentation sans avoir à reprendre les codes de tous les programmes qui utilisent le type abstrait de données,
- utiliser un type abstrait de données en faisant **abstraction** de son implémentation.

Parmi les types abstraits de données on distingue les structures de données **linéaires**, **hiérarchiques** ou **relationnelles**.

Parmi les types abstraits de données on distingue les structures de données **linéaires**, **hiérarchiques** ou **relationnelles**.

On regroupe dans la première catégorie les structures de **Liste**, de **Pile**, de **File** et de **Dictionnaire**.

Parmi les types abstraits de données on distingue les structures de données **linéaires**, **hiérarchiques** ou **relationnelles**.

On regroupe dans la première catégorie les structures de **Liste**, de **Pile**, de **File** et de **Dictionnaire**.

Les **arbres** sont des exemples de structures **hiérarchiques** tandis que les **graphes** en sont pour les structures **relationnelles**.

Définition

Une **pile** est une structure de données linéaire pour laquelle l'ajout et le retrait d'un élément obéit à la discipline **LIFO** pour *Last In, First Out*.

Définition

Une **pile** est une structure de données linéaire pour laquelle l'ajout et le retrait d'un élément obéit à la discipline **LIFO** pour *Last In, First Out*.

- On peut associer cette structure à une pile d'assiettes, l'assiette retirée en haut de la pile est la dernière à avoir été déposée.
- Une situation assez classique associée à la structure de pile est un historique de navigation sur le WEB.

L'interface d'une implémentation de la structure de **pile** offre au moins les cinq opérations suivantes.

Interface minimale

`creer_pile`	Renvoie une pile vide.
`est_vide`	Renvoie `True` si cette pile est vide ou `False` sinon.
`empiler(x)`	Ajoute l'élément `x` à une pile.
`depiler()`	Renvoie la valeur du dernier élément à avoir été empilé et enlève celui-ci de la pile.
`consulter()`	Renvoie la valeur du dernier élément à avoir été empilé mais sans le retirer.

Définition

Une **file** est une structure de données linéaire pour laquelle l'ajout et le retrait d'un élément obéit à la discipline **FIFO** pour *First In, First Out*.

Définition

Une **file** est une structure de données linéaire pour laquelle l'ajout et le retrait d'un élément obéit à la discipline **FIFO** pour *First In, First Out*.

- Une file d'attente à un guichet est, normalement, la situation classique que l'on peut associer à cette structure.
- On peut aussi penser à la liste des travaux en attente d'impression sur un photocopieur.

L'interface d'une implémentation de la structure de **file** offre au moins les cinq opérations suivantes.

Interface minimale

`creer_file`	Renvoie une file vide.
`est_vide`	Renvoie `True` si cette file est vide ou `False` sinon.
`enfiler(x)`	Ajoute l'élément `x` à une file.
`defiler()`	Renvoie la valeur du dernier élément à avoir été enfilé et enlève celui-ci de la file.
`consulter()`	Renvoie la valeur du dernier élément à avoir été enfilé mais sans le retirer.

Définition

Une Liste est une séquence ordonnée d'éléments.

Définition

Une Liste est une séquence ordonnée d'éléments.

Avec cette définition, une Liste ne semble pas très différente d'une Pile ou d'une File.

Définition

Une Liste est une séquence ordonnée d'éléments.

Avec cette définition, une Liste ne semble pas très différente d'une Pile ou d'une File.

Cependant, cette structure de données permet davantage d'opérations et ainsi Pile ou File peut être pensée comme Liste particulière.

Définition

Une Liste est une séquence ordonnée d'éléments.

Avec cette définition, une Liste ne semble pas très différente d'une Pile ou d'une File.

Cependant, cette structure de données permet davantage d'opérations et ainsi Pile ou File peut être pensée comme Liste particulière.

Une interface est donnée par le tableau qui suit où la lettre `l` désigne une liste et la lettre `e` un élément.

Interface

`creer_liste()`	Un **constructeur**, cette fonction renvoie une liste vide.
`est_liste_vide(l)`	Un **accesseur**, cette fonction renvoie `Vrai` si la liste `l` est vide ou `Faux` sinon.
`inserer(e, l)`	Un **opérateur**, cette fonction renvoie une nouvelle liste en insérant l'élément `e` en tête de la liste `l`.
`tete(l)`	Un **accesseur**, cette fonction renvoie l'élément en tête de la liste `l` si elle n'est pas vide.
`reste(l)`	Un **accesseur**, cette fonction renvoie une nouvelle liste constituée de la liste `l` privée de son premier élément.

Interface

`afficher(l)`	Un **accesseur**, cette fonction renvoie une chaîne de caractères qui représente la liste du premier au dernier élément.
`enumerer(l)`	Un **itérateur**, cette fonction renvoie les éléments de la liste `l` comme variable d'un type Python énumérable avec la construction `in`.
`taille(l)`	Un **accesseur**, cette fonction renvoie le nombre d'éléments composant la liste.
`ieme_element(i, l)`	Un **accesseur**, cette fonction renvoie la valeur du i- élément de la liste `l` s'il existe.

Interface

`ajouter(e, l)`	Un **opérateur**, cette fonction renvoie une nouvelle liste en ajoutant l'élément `e` à la fin de la liste `l`.
`inserer_ieme(e, i, l)`	Un **opérateur**, cette fonction renvoie une nouvelle liste en insérant `e` comme i-ième élément de la liste `l` si l'indice est valide.
`substituer(e, i, l)`	Un **opérateur**, cette fonction renvoie une nouvelle liste en substituant `e` au i-ième élément de la liste `l` si l'indice est valide.
`supprimer(i, l)`	Un **opérateur**, cette fonction renvoie une nouvelle liste en supprimant l'élement d'indice `i` de la liste `l` si l'indice est valide.

Définition

Un **Dictionnaire** est un ensemble non ordonné d'associations de valeurs à des clés.

Définition

Un **Dictionnaire** est un ensemble non ordonné d'associations de valeurs à des clés.

Il s'agit d'une définition proche de celle d'un dictionnaire d'une langue qui possède des mots pour clés et leurs définitions pour valeurs.

Définition

Un **Dictionnaire** est un ensemble non ordonné d'associations de valeurs à des clés.

Il s'agit d'une définition proche de celle d'un dictionnaire d'une langue qui possède des mots pour clés et leurs définitions pour valeurs.

Une interface est donnée par le tableau qui suit ou la lettre `d` désigne un dictionnaire.

Interface minimale

`creer_dico()`	Un **constructeur**, cette fonction renvoie un dictionnaire vide.
`ajouter(cle, val, d)`	Un **opérateur**, cette fonction ajoute l'association `cle-val` au dictionnaire `d`.
`valeur(cle, d)`	Un **accesseur**, cette fonction renvoie la valeur associée à la clef `cle` dans le dictionnaire `d` si elle existe.
`est_cle(cle, d)`	Un **accesseur**, cette fonction renvoie `True` si `cle` est une clef du dictionnaire `d` et `False` sinon.

Interface minimale

`clefs(d)`	Un **itérateur**, cette fonction renvoie une variable d'un type Python qui permet d'énumérer les clefs du dictionnaire `d` avec la construction `in`.
`substituer(cle, val, d)`	Un **opérateur**, cette fonction substitue la valeur `val` à celle de l'association avec `cle` dans le dictionnaire `d` si c'est une clef.
`supprimer(cle, d)`	Un **opérateur**, cette fonction supprime la clef `cle`, et donc l'association, du dictionnaire `d` si c'est une clef.

Une implémentation simple est réalisable avec un tableau dynamique dont les éléments sont des valeurs de type `tuple` à deux éléments `(cle, val)`.

Une implémentation simple est réalisable avec un tableau dynamique dont les éléments sont des valeurs de type `tuple` à deux éléments `(cle, val)`.

Une autre implémentation consiste à utiliser un tableau en transformant la valeur de la clef en un nombre entier par une **fonction de hachage**. Cette méthode permet un traitement plus efficace mais il est nécessaire d'éviter les phénomènes de **collision**, c'est-à-dire le fait d'avoir le même **haché** pour deux clefs distinctes.

Une implémentation simple est réalisable avec un tableau dynamique dont les éléments sont des valeurs de type `tuple` à deux éléments `(cle, val)`.

Une autre implémentation consiste à utiliser un tableau en transformant la valeur de la clef en un nombre entier par une **fonction de hachage**. Cette méthode permet un traitement plus efficace mais il est nécessaire d'éviter les phénomènes de **collision**, c'est-à-dire le fait d'avoir le même **haché** pour deux clefs distinctes.

Cette implémentation est présentée en diaporama exemple.