EISTI
Ecole Internationale des Sciences du Traitement de l'Information

Algorithmique et Conteneurs

Approche mathématique d'un type : Type abstrait

Spécifications axiomatiques d'un type abstrait

Type abstrait et structure de données

- Un type abstrait est un cahier de charges qu'une structure de données, un type ou une classe doit implémenter
- Un type abstrait est indépendant de tout langage de programmation
- Une structure de données dépend :
  - Du programmeur
  - Du langage de programmation retenu
- Pour un type abstrait, on utilisera le terme d'objet à la place du terme variable

La structure d'un type abstrait

- Définition d'un type abstrait
  - par les méthodes sur le type
  - indépendante de l'implémentation en mémoire
    - Données masquées
    - Implémentation définie à la programmation
- Trois types de méthodes
  - Méthodes de création et d'initialisation
  - Méthodes d'accès à tout ou partie de l'objet
  - Méthodes de mise à jour de l'objet

Les méthodes d'un type

- Signature ou prototypage d'une méthode
  - Nature : procédure ou fonction
  - Nommage : nom donné à la méthode
  - Paramètres : liste ordonnée et typée des paramètres
  - Retour : pour une fonction, le type du retour
- Précondition d'une méthode
  - Ensemble de conditions requises pour que la méthode se déroule normalement
  - Définition sous forme d'une proposition logique
- Postcondition d'une méthode
  - Etat de l'objet après l'exécution de la méthode
  - Définition sous forme d'une proposition logique ou d'un algorithme

Les natures de méthode

- Méthodes de base
  - Dépendance directe avec les données masquées
  - Méthodes nécessaires et suffisantes à la définition du type
  - Définition à l'aide d'axiomes
- Méthodes d'extension
  - Indépendance avec les données masquées
  - Définition à l'aide d'un algorithme
  - L'algorithme peut faire appel à d'autres méthodes

La vie d'un objet

1. Déclaration (*): l'objet est déclaré mais n'est pas initialisé
2. (Alloué et) initialisé (*): l'objet est (réservé en mémoire vive et) initialisé (à une bonne valeur)
3. Mise à jour : l'objet est mis à jour de 0 à plusieurs fois lors de l'exécution de l'algorithme.
4. Accès : l'objet est accédé de 1 à plusieurs fois entre son initialisation et ses différentes mises à jour

(*) : Dans les langages modernes, on déclare des références d'objets qui ne réfèrent rien. Puis on alloue réellement l'objet qui est alors repérée par la référence.

Restriction aux méthodes de base

- Une méthode d'extension se définit à l'aide de méthodes de base et de méthodes d'extension.
- Par récurrence, on peut donc montrer que la définition d'une méthode d'extension ne dépend que de la définition des méthodes de base.
- Un type abstrait est donc défini sans ambiguïté si les méthodes de base sont définies sans ambiguïté.
- Que signifie : **Défini sans ambiguïté**
  - **Consistance et complétude**

Consistance et complétude

- Consistance
  - Les axiomes ne sont pas contradictoires
  - Plusieurs appels d'une même méthode d'accès avec les mêmes arguments donnent le même résultat
- Complétude
  - Préalable : un type abstrait qui n'a pas de méthode d'accès n'a pas de sens
  - Après la création d'un objet et l'application de 0 à plusieurs méthodes de mise à jour à cet objet, on doit connaître le résultat de l'exécution de toute méthode d'accès à cet objet

Exemple : Type abstrait Vecteur

- Cahier de charges en langage naturel
  - Un vecteur est une suite finie d'éléments de même type.
  - Le plus petit indice et le plus grand indice sont définis à la construction. On les nomme respectivement borne inférieure et borne supérieure.
  - A la construction, tous les éléments de la suite sont indéfinis
  - On peut accéder à tout élément du vecteur directement via son indice si cet élément est défini.
  - On peut affecter tout élément du vecteur directement via son indice
  - On peut récupérer les bornes inférieures et les bornes supérieures du vecteur

TA Vecteur : Liste des opérations

- Cahier de charges à l'aide d'un type abstrait
  - Utilise Booleen et Element
  - Méthode de construction et initialisation
    - fonction creerVecteur(bi : Entier, bs : Entier) : Vecteur
  - Méthodes d'accès
    - fonction borneInf(v : Vecteur) : Entier
    - fonction borneSup(v : Vecteur) : Entier
    - fonction ieme(v : Vecteur, i : Entier) : Element
    - fonction estInitialise(v : Vecteur, i : Entier) : Booleen
  - Méthode de mise à jour

    Remarque : Pour ces méthodes, on renvoie l'objet mis à jour
    - fonction modifieme(ES v : Vecteur, i : Entier, e : Element) : Vecteur

TA Vecteur : Préconditions

- définie(creerVecteur(bi, bs)) $\Rightarrow$ bi $\leq$ bs
- définie(ieme(v, i)) $\Rightarrow$ borneInf(v) $\leq$ i $\leq$ borneSup(v)
- définie(modifieme(v, i,e)) $\Rightarrow$ borneInf(v) $\leq$ i $\leq$ borneSup(v)
- définie(estInitialise(v, i)) $\Rightarrow$ borneInf(v) $\leq$ i $\leq$ borneSup(v)

TA Vecteur : Axiomes ou Postconditions

- borneInf(creerVecteur(bi, bs)) = bi
- borneSup(creerVecteur(bi, bs)) = bs
- ieme(modifieme(v, i,e),i) = e
- v1 ← copie(v) et i <> i'⇒

  ieme(modifieme(v, i,e),i') = ieme(v1,i') et

  estInitialise(modifieme(v, i,e),i') = estInitialise(ieme(v1,i')) et

  borneInf(v1) = borneInf(modifieme(v, i,e)) et

  borneSup(v1) = bonreSup(modifieme(v, i,e))
- estInitialise(modifieme(v, i,e),i)
- bi ≤ i ≤ bs ⇒ non estInitialise(creerVecteur(bi, bs),i)

TA Vecteur : une implémentation en C

Le fichier vecteur.h

```
// Les types utilisés par Vecteur
typedef enum { faux, vrai } Booleen;
typedef void * Element;

typedef struct {
      int bi, bs;
      void ** pptab;
} SVecteur;

typedef SVecteur * Vecteur;
```

```
Vecteur creerVecteur(int parbi, int parbs);
Element ieme(Vecteur pvect, int i);
Vecteur modifieme (Vecteur pvect, int i,
Element pe);
Booleen estInitialise(Vecteur pvect, int i);
int borneInf(Vecteur pvect);
int borneSup(Vecteur pvect);
```

TA Vecteur : une implémentation en Java

Le fichier vecteur.java

```
Class Vecteur {
    // les attributs privés donc masqués
    private int bi,bs;
    private Object  tab[];

    // Méthode de construction
    public  static Vecteur creerVecteur(int parbi, int parbs) { … };
    // Méthodes d'accès
    public Object ieme(int i) { … };
    public boolean estInitialise(int i) { … };
    public int borneInf() { … };
    public int borneSup() { … };
    // Méthode de mise à jour
    public Vecteur modifieme (int i, Element pe) { … return this;};
    }
```