Type Concret – Type Abstrait

Type concret

Définition : Un type concret est une structure (ou un type) de données spécifiée par la manière dont les objets sont physiquement stockés dans la mémoire de l'ordinateur.

Par exemple :

- un entier
- un tableau,
- un réel, etc.

Type abstrait

Définition: Un type abstrait est un triplet composé :

- d'un nom,
- d'un ensemble de valeurs,
- d'un ensemble d'opérations (souvent appelé primitives) définies sur ces valeurs.

D'un point de vue complexité, on considère que les primitives (à part celle d'initialisation si elle existe) ont une complexité en temps et en mémoire en O(1).

Les primitives devront préciser le comportement de la structure (ce qui se passe), et la signature (paramètres d'entrée et valeurs ou effet en sortie), et parfois des pré-conditions.

Exemple 1 : Containeur

Définition : Un containeur est un type abstrait permettant de représenter des collections d'objets ainsi que les opérations sur ces objets.

Les collections que l'on veut représenter peuvent être ordonnées ou non, numériques ou non.

L'ordre est parfois fourni par un évènement extérieur.

Les collections d'objets peuvent parfois contenir des éléments identiques.

Exemple 1 : Containeur

Primitives :

- Accès

  valeur : containeur → objet

- Modification

  creerContaineur: containeur → vide
  ajouter : containeur X objet → vide
  supprimer : containeur X objet → vide
  detruireContaineur : containeur → vide

Exemple 2 : Nombre complexe

Les nombres complexes ne sont pas des types de bases. On peut les définir comme un type abstrait :

- nom : *nombreComplexe*
- ensemble de valeur : *réel×réel*
- primitives :

Lesquelles ?

Exemple 2: Nombre complexe

- nom : *nombreComplexe*
- ensemble de valeur : *réel×réel*
- primitives :
  - multiplication :

    (nombreComplexe × nombreComplexe) → nombreComplexe
  - addition :

    (nombreComplexe × nombreComplexe) → nombreComplexe
  - module :

    nombreComplexe → réel
  - etc

Implémentation

Définition: L'implémentation consiste à choisir une structure de données et les algorithmes associés pour réaliser un type abstrait

La structure de données utilisée pour l'implémentation peut elle-même être un type abstrait.

L'implémentation doit respecter la complexité des primitives à part celle d'initialisation (celle-ci ne s'exécutera qu'une fois).

Implémentation : nombres complexes

Le type abstrait `nombreComplexe` peut être implémenté de la manière suivante :

```
nombreComplexe=structure
    r:réel;
    i:réel;
Finstructure

var c : nombreComplexe;
```

nombre Complexe	
Nom	Type
r	réel
i	réel

Nom variable	Type
c	nombreComplexe
c.r	réel
c.i	réel

Implémentation : nombres complexes

Le type abstrait `nombreComplexe` peut être implémenté de la manière suivante :

```
nombreComplexe=structure
      r:réel;
      i:réel;
Finstructure

fonction C_mul(val a,b:nombreComplexe):
nombreComplexe;
  var c:nombreComplexe;
  debut
     c.r=a.r*b.r-a.i*b.i;
     c.i=a.r*b.i+a.i*b.r;
     retourner(c)
  fin
```

Implémentation : nombres complexes

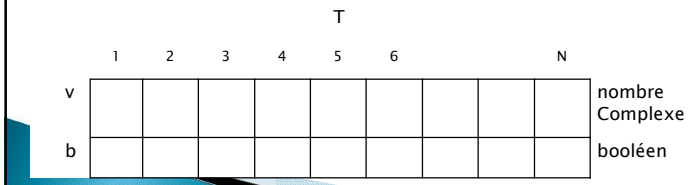
```
fonction C_add(val a,b:nombreComplexe):nombreComplexe;
  var c:nombreComplexe;
  debut
    c.r=a.r+b.r;
    c.i=a.i+b.i;
    retourner(c)
  fin

fonction C_mod(val a:nombreComplexe):réel;
  debut
    retourner(sqrt(a.r*a.r+a.i*a.i))
  fin
```

Implémentation : container

Un containeur de `nombreComplexe` peut être implémenté par un tableau de `nombreComplexe`.

```
containeur d'objet=tableau[1..N]de structure
      v: nombreComplexe;
      b: booleen;
Finstructure
```



Implémentation : container

Primitive de Modification

```
fonction creerContaineur(ref C:containeur de
                nombreComplexe):vide;
  var i:entier;
  debut
    pour i allant de 1 à N faire
      C[i].b=faux;
    finPour;
  fin
```

C

	1	2	3	4	5	…		N	
v									nombreComplexe
b	F	F	F	F	F	F…F	F	F	booléen

Complexité : ?

Implémentation : container

Primitive de Modification :

```
fonction ajouter(ref C: containeur de
            nombreComplexe;val x:nombreComplexe):vide
  var i:entier;
  debut
    i=1;
    tant que i<=N et C[i].b faire
      i=i+1;
    fintantque
    si i<=N alors
      C[i].v=x;
      C[i].b=vrai
    finsi
  fin
```

C

	1	2	3	4	5	…		N	
v	c1	c21	c5	c2					nombreComplexe
b	V	V	V	V	F	F…F	F	F	booléen

Complexité : ?

Implémentation : container

Primitive de Modification

```
fonction supprimer(ref C: containeur de
          nombreComplexe;val x:nombreComplexe):vide
  var i:entier;
  debut
    i=1;
    tant que i<=N et (C[i].v!=x or C[i]==faux) faire
      i=i+1;
    fintantque
    si i<=N alors
      C[i].b=faux
    finsi
  fin
```

C

	1	2	3	4	5…		N	
v	c1	c21	c5	c2				nombreComplexe
b	V	V	F	V	F…F	F	F	booléen

Complexité : ?

Implémentation : container

Primitive de Modification

```
fonction detruireContaineur(ref C : containeur
de nombreComplexe):vide
  debut
    pour i allant de 1 à N faire
      C[i].b=faux;
    finPour;
  fin
```

Complexité : ?

Implémentation : container

Primitives d'accès

```
fonction valeur(ref C : containeur de
nombreComplexe):nombreComplexe;
/* retourne le 1er nombre complexe présent/*
  var i:entier;
  debut
    i=1;
    tant que i<=n et !C[i].b faire
      i=i+1;
    fintantque
    si i<=n alors
      retourner(C[i].v)
    sinon
      retourner(NULL)
    finsi
  fin
```

2 conditions d'arrêt

Test en sortie de boucle

Complexité : ?

Conclusion

- Sans plus de précisions un container ne permet pas d'implémenter des primitives en O(1)
- En ajoutant des spécifications on peut implémenter des primitives en O(1) :
  - Listes chainées
  - Files
  - Piles
  - etc

Conclusion

- On peut voir le programmeur qui développe une bibliothèque informatique comme le concepteur du type abstrait :
  - Il dévoile dans la documentation ce qu'on peut attendre de la structure (comportement, stockage, accès, etc.) sans dévoiler comment elle est physiquement construite.
- Alors que le programmeur qui importe une bibliothèque dans son code est l'utilisateur du type abstrait :
  - il voit la documentation comme une garantie de performances pour certaines opérations.
- Vous serez à tour de rôle concepteur et/ou utilisateur.