# Structure

## I3 - Algorithmique et programmation

Nicolas Delestre, Nicolas Malandain

# Plan

1. Pourquoi les structures ?
2. Définition des structures
3. Accès aux attributs
4. Bonnes pratiques
5. Un exemple complet

Architecture des Systèmes d'Information

# Pourquoi les structures ? 1 / 3

- Imaginons que l'on veuille afficher les notes d'une promotion par ordre croissant avec les noms et prénoms de chaque étudiant
  - On va donc utiliser trois tableaux (pour stocker les noms, les prénoms et les notes)

| prenoms | noms | notes |
|---|---|---|
| Gary | Citacaisse | 12 |
| Alain | Terieur | 4 |
| Sacha | Touille | 15 |
| Thérèse | Ultasonnul | 10 |

**Problème**

Lorsque l'on va trier le tableau des notes il faut aussi modifier les tableaux "noms" et "prenoms"

# Pourquoi les structures ? 2 / 3

## Avec :

**Type** Notes = **Tableau**[1..MAX] **de Reel**
**Type** Noms = **Tableau**[1..MAX] **de Chaine de caracteres**
**Type** Prenoms = **Tableau**[1..MAX] **de Chaine de caracteres**

## La procédure de tri va donc ressembler à :

**procédure** trierNotesParMinimumSuccessif **(E/S** lesNotes : Notes, lesNoms : Noms, lesPrenoms : Prenoms ; **E** nbEtudiants : Entier**)**

```
    Déclaration   i,pos : Naturel
debut
    pour i ←1 à nbEtudiants-1 faire
        pos ← indiceDuMinimum(lesNotes,i,nbEtudiants)
        echangerReel(lesNotes[i],lesNotes[pos])
        echangerChaine(lesNoms[i],lesNoms[pos])
        echangerChaine(lesPrenoms[i],lesPrenoms[pos])
    finpour
fin
```

Architecture des Systèmes d'Information

# Pourquoi les structures ? 3 / 3

- Mais :
  - Cela multiplie le risque d'erreur
  - Cela oblige à développer autant de procédure *echanger* qu'il y a d'éléments qui caractérisent l'étudiant
- Il serait plus intéressant qu'un étudiant soit caractérisé par un type (nommé par exemple *Etudiant*)
  - C'est le rôle des **structures** : rassembler au sein d'une seule entité un ensemble fini d'éléments de types éventuellement différents
    - Par exemple pour le type Etudiant : deux chaînes de caractères (le nom et le prénom) et un réel (la note)
  - C'est le deuxième type complexe disponible en algorithmique

# Définition d'une structure 1 / 2

- À la différence des tableaux, il n'existe pas par défaut de type structure
  - On ne peut pas déclarer une variable de type structure
  - On construit toujours un nouveau type basé sur une structure et on déclare des variables sur ce nouveau type

## Syntaxe

**Type** NomDuType = **Structure**
  $attribut_1$ : $TypeAttribut_1$
  $attribut_2$ : $TypeAttribut_2$
  ...
  $attribut_n$ : $TypeAttribut_n$
**finstructure**

## Remarque

un *attribut* est aussi appelé un *champ*

# Définition d'une structure 2 / 2

- Le type d'un attribut peut être :
  - un type simple
  - un type complexe
    - un tableau
    - un type basé sur une structure

# Exemple

**Type** Date = **Structure**
    leJour : 1..31
    leMois : 1..12
    lAnnee : **Naturel**
**finstructure**
**Type** Etudiant = **Structure**
    leNom : **Chaine de caracteres**
    lePrenom : **Chaine de caracteres**
    laNote : **Reel**
    laDateDeNaissance : Date
**finstructure**
**Type** Promotion = **Structure**
    lesEtudiants : **Tableau[**1..MAX**] d'**Etudiant
    nbEtudiants : **Naturel**
**finstructure**

...
unePromotion : Promotion

# Accès aux attributs

- On accède aux attributs d'une variable dont le type est basé sur une structure en suffixant le nom de la variable d'un point (".") suivi du nom de l'attribut
  - Par exemple, dans notre cas pour affecter le nom "Citacaisse" à notre premier étudiant, on a le code suivant :
    unePromotion.lesEtudiants[1].leNom $\leftarrow$ "Citacaisse"
- En fait pour bien comprendre il faut décomposer :

$$\underbrace{\underbrace{\underbrace{\underbrace{\underbrace{unePromotion}_{Promotion}.lesEtudiants[1]}_{Tableau[1..MAX]d'Etudiants}.laDateDeNaissance}_{Etudiant}.lAnnee}_{Date}}_{Naturel}$$

# Instructions et opérateurs

## Instructions

- L'affectation ( $\leftarrow$ )
  - Copie toute la structure
- Puisque les instructions *lire* et *écrire* n'acceptent que des paramètres de type simple, on ne peut pas *lire* ou *écrire* directement des structures

## Operations

- L'opérateur d'égalité ($=$)
  - est Vrai si tous les attributs sont égaux
- L'opérateur d'inégalité ($\neq$)
  - est Vrai si au moins un des attributs est différent
- Pas d'opérateur de comparaison d'ordre

# Un exemple

## Point2D

**Type** Point2D = **Structure**
    x : **Reel**
    y : **Reel**
**finstructure**
**Type** Vecteur = Point2D

## Exemple de fonctions sur Point2D

**fonction** point2DEnChaine **(**p :Point2D**) : Chaine de caracteres**
**debut**
    **retourner** "("+reelEnChaine(p.x)+","+reelEnChaine(p.y)+")"
**fin**
**procédure** calculerTranslation **(E** v : Vecteur,**E/S** p : Point2D**)**
...
**procédure** calculerRotation **(E** c : Point2D, angle : **Reel**,**E/S** p : Point2D**)**
...

Architecture des Systèmes d'Information

# Évolution

Que se passe-t-il si pour une raison on est obligé de modifier la structure ?

## Évolution possible

**Type** Point2D = **Structure**
    module : **Reel**
    argument : **Reel**
**finstructure**

⇒ on sera obligé de réécrire tous les algorithmes utilisant les Point2D

# Solution

## Bonnes pratiques

Séparer le fait de définir une structure et le fait de l'utiliser :
→ Création d'accesseurs : des fonctions/procédures qui permettent de manipuler la structure :

- pour **obtenir** l'information (*get*)
- pour **fixer** l'information (*set*)

Si la structure évolue, seuls les accesseurs changent

## Exemple de fonction utilisant les accesseurs :

```
fonction point2DEnChaine (p :Point2D) : Chaine de caracteres
debut
    retourner "("+reelEnChaine(obtenirAbscisse(p))
    +","+reelEnChaine(obtenirOrdonnée(p))+")"
fin
```

Architecture des Systèmes d'Information

# Solution dans le premier cas

## Accesseur

```
fonction obtenirAbscisse (p : Point2D) : Reel
debut
    retourner p.x
fin
fonction obtenirOrdonnee (p : Point2D) : Reel
debut
    retourner p.y
fin
procédure fixerAbscisse (E/S p : Point2D, E x : Reel)
debut
    p.x ← x
fin
procédure fixerOrdonnee (E/S p : Point2D, E y : Reel)
debut
    p.y ← y
fin
```

# Solution dans le deuxième cas 1 / 2

En supposant que l'on possède les fonctions mathématiques suivantes :
**fonction** cos **(**angle : **Reel) : Reel**
**fonction** sin **(**angle : **Reel) : Reel**
**fonction** arctan **(**a : **Reel) : Reel**
**fonction** sqrt **(**a : **Reel) : Reel**

## Accesseurs :

```
fonction obtenirAbscisse (p : Point2D) : Reel
debut
    retourner p.module*cos(p.argument)
fin
fonction obtenirOrdonnee (p : Point2D) : Reel
debut
    retourner p.module*sin(p.argument)
fin
```

# Solution dans le deuxième cas 2 / 2

## Accesseurs (suite) :

**procédure** fixerAbscisseOrdonnee **(E/S** p : Point2D, **E** x,y : **Reel)**
**debut**
    p.module ← sqrt(x*x+y*y)
    **si** x≠0 **alors**
        p.argument ← arctan(y/x)
    **sinon**
        ...
    **finsi**
**fin**
**procédure** fixerAbscisse **(E/S** p : Point2D, **E** x : **Reel)**
    **Déclaration** y : **Reel**
**debut**
    y ← obtenirOrdonnee(p)
    fixerAbscisseOrdonnee(p,x,y)
**fin**
**procédure** fixerOrdonnee **(E/S** p : Point2D, **E** y : **Reel)**
    **Déclaration** x : **Reel**
**debut**
    x ← obtenirAbscisse(p)
    fixerAbscisseOrdonnee(p,x,y)
**fin**

# Exemple

- Proposez un type Polynome avec les opérations de base :
  - création d'un polynome à partir d'un monome (un coefficient et un degré)
  - addition de deux polynomes
  - obtention du degré d'un polynome
  - obtention du coefficient d'un monome d'un certain degré

# Signature des opérations du type Polynome

## Conception préliminaire

**fonction** polynomeAUnSeulMonome **(**coef : **Reel**, deg : **Naturel) :** Polynome
**fonction** addition **(**p1,p2 : Polynome**) :** Polynome
**fonction** obtenirDegre **(**p : Polynome**) : Naturel**
**fonction** obtenirCoefficientDUnMonome **(**p : Polynome, degre : **Naturel) : Reel**

# Conception du type Polynome 1 / 7

## Conception détaillée

Il y a plusieurs possibilités

## Proposition 1

**Type** Polynome = **Structure**
    lesCoefficients : **Tableau[**0..MAX**] de Reel**
    degresMax : 0..MAX
**finstructure**

## Proposition 2

**Type** Polynome = **Structure**
    lesDegres : **Tableau[**1..MAX**] de Naturel**
    lesCoefficients : **Tableau[**1..MAX**] de Reel**
    nbMonomes : **NaturelNonNul**
**finstructure**

# Conception du type Polynome 2 / 7

## Proposition 3

**Type** Polynome = **Structure**
    lesMonomes : **Tableau**[1..MAX] **de** Monome
    nbMonomes : **NaturelNonNul**
**finstructure**

avec
**fonction** monome (coef : **Reel**, deg : **Naturel**) : Monome
**fonction** obtenirDegre (m : Monome) : **Naturel**
**fonction** obtenirCoefficient (m : Monome) : **Reel**

# Conception du type Polynome 3 / 7

## Proposition 4

**Type** Polynome = **Structure**
    lesMonomes : **Tableau[**1..MAX**] de** Monome
    nbMonomes : **NaturelNonNul**
    degre : **Naturel**
**finstructure**

Nous allons retenir la quatrième proposition

# Conception du type Polynome 4 / 7

*polynomeAUnSeulMonome*

**fonction** polynomeAUnSeulMonome **(**coef : **Reel**, deg : **Naturel) :** Polynome

**Déclaration** resultat : Polynome

**debut**

resultat.nbMonomes ← 1

resultat.lesMonomes[1] ← monome(coef,deg)

resultat.degre ← deg

**retourner** resultat

**fin**

# Conception du type Polynome 5 / 7

## *addition*

```
fonction addition (p1,p2 : Polynome) : Polynome
    Déclaration   resultat : Polynome
                  degre : Naturel
                  rang : Naturel
debut
    resultat ← p1
    pour degre ←0 à obtenirDegre(p2) faire
        si obtenirCoefficient(resultat,degre)=0 alors
            si obtenirCoefficient(p2,degre)≠0 alors
                resultat.nbMonomes ← resultat.nbMonomes+1
                resultat.lesMonomes[resultat.nbMonomes] ← monome(obtenirCoefficient(p2,degre),degre)
            finsi
        sinon
            rang ← obtenirRang(resultat,degre)
            resultat.lesMonomes[rang] ← monome(obtenirCoefficient(resultat.lesMonomes[rang])
                                                +obtenirCoefficientDUnMonome(p2,degre),degre)
        finsi
    finpour
    resultat.degre ← max(obtenirDegre(p1),obtenirDegre(p2))
    retourner resultat
fin
```

# Conception du type Polynome 6 / 7

## *obtenirRang (fonction privée)*

```
fonction obtenirRang (p : Polynome, degre : Naturel) : Naturel
    Déclaration   i : Naturel
                  trouve : Booleen
debut
    trouve ← FAUX
    i ← 1
    tant que i≤p.nbMonomes et non trouve faire
        si obtenirDegre(p.lesMonomes[i])=degre alors
            trouve ← VRAI
        sinon
            i ← i+1
        finsi
    fintantque
    si trouve alors
        retourner i
    sinon
        retourner 0
    finsi
fin
```

# Conception du type Polynome 7 / 7

## *obtenirCoefficientDUnMonome*

```
fonction obtenirCoefficientDUnMonome (p : Polynome, degre : Naturel) : Reel
    Déclaration   i : Naturel
debut
    i ← obtenirRang(p,degre)
    si i=0 alors
        retourner 0
    sinon
        retourner obtenirCoefficient(p.lesMonomes[i])
    finsi
fin
```

## *obtenirDegre*

```
fonction obtenirDegre (p : Polynome) : Naturel
debut
    retourner p.degre
fin
```

## Exercices

- Faite la conception détaillée du type Monome
- Comment s'arranger pour que la complexité de obtenirCoefficientDUnMonome soit inférieur à $0(n)$

# Conclusion

- Les structures sont la pierre angulaire du développement informatique. Elles permettent de représenter les « objets » du « monde »
- Il est très important de séparer les rôles « utilisateur » et « concepteur », c'est-à-dire appliquer le **principe d'encapsulation**