

Slide 1

Les principes des langages de programmation

Concevoir des algorithmes : une activité ancienne (< -2500)

Slide 2

Transformée au milieu du XX^e siècle : ordinateurs

Écrire un texte lu par une machine : langage formel

Langage formel

Sous-ensemble de \mathcal{L}^* dont on connaît une définition mathématique

Slide 3

e.g. $\{a, aa, aaa, aaaa, \dots\}$,

ensemble des mots d'un nombre pair de lettres

OD : -1,25 (-0,50)180° OG : -1,00 (-0,25)180°

De nombreux langages formels :

<http://www.enseignement.polytechnique.fr/informatique/INF321/>

... pour aborder de nouveaux langages dans la suite de leur cursus à l'École et au delà.

</p>

Slide 4

<p>

Il permet d'accéder au cours

Algorithmes et

Programmation : du séquentiel au

distribué. </p>

<HR>...

Parmi lesquels : **les langages de programmation**

Slide 5

```
class Hypothenuse {
    static double hypothenuse (final double x,
                               final double y) {
        return Math.sqrt(x * x + y * y);}
    public static void main (String [] args) {
        System.out.println(hypothenuse(3,4));}}
```

Une double contrainte

Un programme doit être lisible par une **machine** (formel)

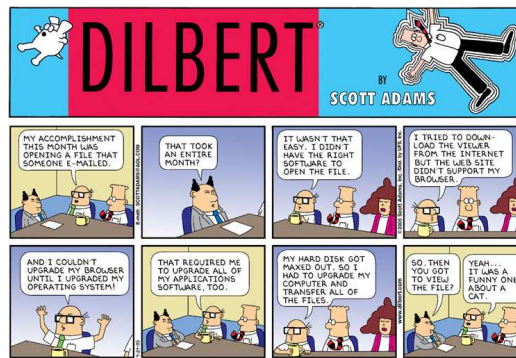
Slide 6

Un programme doit être lisible par un **être humain** (plus ou moins naturel)

De 6 lignes à 1 000 000 lignes

Slide 7 Un saut dans la **complexité** des objets industriels (v.s. locomotive à vapeur, appareil photo, ...)

Slide 8



Nouveaux problèmes : y a un bug, ...

Slide 9



En retour, conception d'autres objets complexes : circuits intégrés
1 000 000 000 transistors

Les langages de programmation

Plus de 2000 (de nouveaux langages avant 2013)

Principes communs : affectation, boucle, cellule, passage par référence

Slide 10

Langues naturelles : article, pronom, verbe, temps, mode, aspect, conjugaison, déclinaison, ...

Comprendre ces **principes** plus important que connaître les **idiotismes** de Java ou de Caml

Slide 11

Apprendre un langage ne suffit pas : comparer avec d'autres, pour dégager les principes universels

Objectifs

1. Apprendre à programmer en **Java**
2. Comprendre les **principes** (comparaison avec Caml et C)
3. Commencer à apprendre les outils qui permettent de décrire la **signification** des programmes (plus : année 3)
4. Apprendre les **algorithmes de base** sur les listes, les arbres et les tableaux (plus : année 2)

Organisation :

10 amphis

10 séances de TD (projets) (groupes 14, 20 en anglais)

Slide 13

tutorat qcm

Évaluation :

TD 7 (14 juin) noté + pâle HC (13 juillet)

Note finale = $(t + 2p)/3$

Slide 14

0. Les délégués

Slide 15

I. Le noyau impératif

Slide 16

Dans (presque) tous les langages, cinq constructions :

la déclaration, l'affectation, la séquence, le test et la boucle

L'affectation

```
x = t ;
```

```
x = 3 + y ;
```

x variable et t expression

Slide 17

variable : un mot d'une ou plusieurs lettres

expression : formée à partir des constantes et des variables avec des opérations (+, -, *, /, %, ...)

En Caml : $x := 3 + !y$

En C : comme en Java

Ce qu'il se passe quand on exécute $x = t ;$

Dans un recoin de la mémoire de l'ordinateur, une case appelée x

On met la valeur de t dans cette case

La valeur antérieure est oubliée

Slide 18

Expressions : $3, 5 + 3, y + 3$

Valeurs (nombre entier) : $3, 8, 10$

État : ensemble des valeurs des variables à un instant donné

Exemples

```
x = 4;
```

Slide 19

```
x = y + 2;
```

La déclaration

Avant de pouvoir utiliser la variable x il faut la déclarer

Associer le nom x à une des cases de la mémoire

```
{int x = t; p}
```

Slide 20

```
{int x = 5; x = 3 + y;}
```

Expression t : valeur initiale (optionnelle)

Instruction p : portée de la variable x

En Caml : `let x = ref 5 in p` En C : comme en Java

`int` : nombres entiers compris entre -2^{31} et $2^{31} - 1$

Outre `int` , trois autres intervalles `byte`, `short`, `long`

Autres types : `boolean` (`false`, `true`)

Slide 21 Flottants : `float`, `double` (`6.02E23`)

Caractères `char` (`'g'`)

Huit types primitifs

Types composites (à suivre) : `String` (`"Bonjour"`)

`{T x = t; p}`

Les variables finales et les variables mutables

Déclaration d'une variable, engagement à ne jamais l'affecter

```
{final T x = 5; p}
```

```
{final int x = 5; y = x + 3;}
```

Slide 22 `{final int x = 5; x = 3;}` incorrecte

Variable non finale : `mutable`

En Caml `let x = 5 in p` et non `let x = ref 5 in p`

`y := x + 3` et non `y := !x + 3` quand `x` est finale

En C, `const` au lieu de `final`

La séquence

```
{p1 p2}  
{x = 1; y = x + 1;}
```

Slide 23 Quand on exécute cette instruction, on exécute p1 puis p2

En Caml : p1 ; p2

En C : comme en Java

Le test

```
if (b) p1 else p2  
if (x < 0) y = -1; else y = 1;
```

Quand on exécute cette instruction, on calcule la valeur de b

Slide 24 Si c'est true on exécute p1 si c'est false on exécute p2

En Caml : if b then p1 else p2

En C : comme en Java

while (b) q : notation finie pour une instruction infinie

```
if (b) {q if (b) {q if (b) {q if (b) ...
                                     else skip;}
      else skip;}
      else skip;}
      else skip;
```

Slide 27

avec une instruction fictive `skip ;` qui ne fait rien

En Caml : `while b do p ;` En C : comme en Java

```
{int x = 3; while (x <= 1000) x = 2;}
```

Slide 28 ne termine pas

Instruction infinie : potentialité de non terminaison

Slide 29

II. Le instructions d'entrée et sortie en Java (en un transparent)

```
System.out.print(x) ;
```

```
System.out.println() ;
```

Slide 30

```
System.out.println(x) ;
```

```
Ppl.readInt () ;
```

Slide 31

III. La fonction Σ

Ce qu'il se passe quand on exécute l'instruction i

$x = t ;$: « On met la valeur de t dans la case x »

Description en langue naturelle rapidement complexe et confuse
(déjà : `while`)

Slide 32 On finit par donner des exemples (... qui couvrent les cas simples)

Comment écrire un interpréteur / compilateur ?

Comment raisonner à propos d'un programme ?

Comment analyser automatiquement un programme ?

Slide 33 Remplacer cette phrase par une définition mathématique

L'état

Ensemble des valeurs des variables à un instant donné

Ensemble infini Var dont les éléments sont appelés *variables*

Ensemble $\text{Val} : [-2^{31}, 2^{31} - 1] \cup \{\text{false}, \text{true}\} \cup$

Slide 34 ...

Un *état* est une fonction d'une partie finie de Var dans Val

e.g. : $[x = 1, y = 4]$

Attention : pas $1 = x$

Exécuter une instruction i transforme l'état

Slide 35

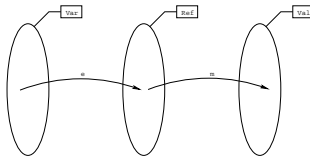
La signification des instructions d'un langage est définie par une fonction Σ

$$\Sigma(i, s) = s'$$

e.g. : $\Sigma(x=3 ; , [x = 1, y = 4]) = [x = 3, y = 4]$

La décomposition de l'état

Utile pour plus tard : $s = m \circ e$, l'environnement et la mémoire



Slide 36

Ensemble intermédiaire **Ref** des références

$$\Sigma(i, e, m) = m'$$

$\Sigma(x=3 ; , [x=r1, y=r2], [r1=1, r2=4]) = [r1=3, r2=4]$

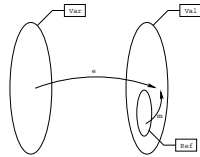
Le cas des variables finales

`final int x = 3; p`

au lieu d'associer `x` à `r` dans `e` et `r` à `3` dans `m`

on associe `x` à `3` dans `e` (la mémoire est ce qui peut changer)

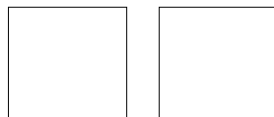
Slide 37 $Ref \subseteq Val$



La représentation graphique des états

Une référence : une case

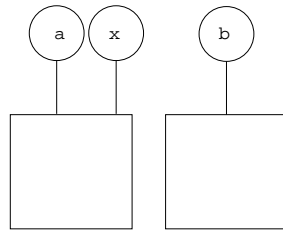
Slide 38



La représentation graphique des états

L'environnement

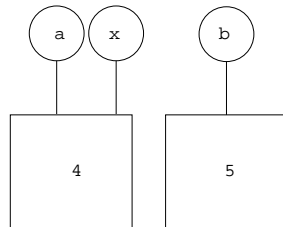
Slide 39



La représentation graphique des états

La mémoire

Slide 40



Variable associée directement à une valeur dans l'environnement

Slide 41



La valeur d'une expression

$x = t ;$

« On met la valeur de t dans la case x »

$\Theta(t, e, m)$

Slide 42

– $\Theta(n, e, m) = n$

– $\Theta(x, e, m) = m(e(x))$ si x est mutable

$\Theta(x, e, m) = e(x)$ si x est finale

– $\Theta(t + u, e, m) = \Theta(t, e, m) + \Theta(u, e, m)$

– idem pour les autres opérations

La valeur d'une expression en Caml et en C

Caml :

$$- \Theta(x, e, m) = e(x)$$

que x soit finale ou mutable

Slide 43

$$- \Theta(!t, e, m) = m(\Theta(t, e, m))$$

$x + 1$ en Java $!x + 1$ en Caml

C : comme Java

L'exécution d'une instruction : la déclaration et l'affectation

La $f^n m + (x = v)$ coïncide avec m sauf en x où elle vaut v

$$- \Sigma(\{T \ x = t ; q\}, e, m) = \Sigma(q, e+(x=r), m+(r=v))$$

r référence quelconque qui n'apparaît pas dans e et m

Slide 44

$$v = \Theta(t, e, m)$$

$$\Sigma(\{\text{final } T \ x = t ; q\}, e, m) =$$

$$\Sigma(q, (e+(x=v)), m)$$

$$v = \Theta(t, e, m)$$

$$- \Sigma(x = t ; e, m) = (m+(e(x)=v))$$

$$v = \Theta(t, e, m)$$

L'exécution d'une instruction : la séquence et le test

- $\Sigma(\{p_1 p_2\}, e, m) = \Sigma(p_2, e, \Sigma(p_1, e, m))$
- $\Sigma(\text{if } (b) p_1 \text{ else } p_2, e, m) = \Sigma(p_1, e, m)$
si $\Theta(b, e, m) = \text{true}$
 $\Sigma(\text{if } (b) p_1 \text{ else } p_2, e, m) = \Sigma(p_2, e, m)$
si $\Theta(b, e, m) = \text{false}$

Slide 45

L'exécution d'une instruction : la boucle

`while (b) q`

Abréviation pour l'instruction infinie

```
if (b) {q if (b) {q if (b) {q if (b) ...  
                                     else skip;}  
                                     else skip;}  
                                     else skip;}  
else skip;
```

Slide 46

avec $\Sigma(\text{skip } ;, e, m) = m$

Approximations finies

```
p0 = if (b) giveup; else skip;  
p1 = if (b) {q if (b) giveup; else skip;} else skip;  
... pn+1 = if (b) {q pn} else skip;
```

Σ jamais définie en (giveup; , e, m)

Slide 47

Si $\Sigma(p_n, e, m)$ définie, alors pour tout $n' > n$, $\Sigma(p_{n'}, e, m)$ définie et $\Sigma(p_{n'}, e, m) = \Sigma(p_n, e, m)$

Suite $\Sigma(p_n, e, m)$ jamais définie ou définie à partir d'un certain rang et constante sur son domaine : **limite**

- $\Sigma(\text{while } (b) \ q, e, m) = \lim_n \Sigma(p_n, e, m)$

Pour résumer

- $\Sigma(\{T \ x = t; \ q\}, e, m) = \Sigma(q, e+(x=r), m+(r=v))$
 $\Sigma(\{\text{final } T \ x = t; \ q\}, e, m) = \Sigma(q, e+(x=v), m)$
r n'apparaît pas dans e et m et $v = \Theta(t, e, m)$

- $\Sigma(x = t; , e, m) = m+(e(x)=v) [v = \Theta(t, e, m)]$

Slide 48

- $\Sigma(\{p_1 \ p_2\}, e, m) = \Sigma(p_2, e, \Sigma(p_1, e, m))$

- $\Sigma(\text{if } (b) \ p_1 \ \text{else } p_2, e, m) = \Sigma(p_1, e, m)$

si $\Theta(b, e, m) = \text{true}$

$\Sigma(\text{if } (b) \ p_1 \ \text{else } p_2, e, m) = \Sigma(p_2, e, m)$

si $\Theta(b, e, m) = \text{false}$

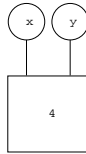
- $\Sigma(\text{while } (b) \ q, e, m) = \lim_n \Sigma(p_n, e, m) \ p_n = \dots$

Exercices

1.5

1.9: Dans $[x = r, y = r], [r = 4]$

Slide 49



On exécute $x = 5$, combien vaut y ?

Et dans $[x = r1, y = r2], [r1 = 4, r2 = 4]$?

Slide 50 [La prochaine fois : les fonctions](#)