Introduction à l'informatique et la programmation IF1 2010-2011

Matthieu Picantin

LIAFA UMR 7089
CNRS & Université Paris 7 Denis Diderot

07/10/2010

Les fonctions du langage

- l'outil d'écriture des programmes
  - par des hommes ou des programmes
  - pour la machine ou les autres hommes
  - à différents niveaux d'abstraction
- le support d'interaction avec la pensée
  - les langages induisent des styles distincts impératif, fonctionnel, logique, temporel, etc...
- le support de guerres de religions
  - rares sont ceux qui aiment deux styles
  - et ceux qui font abstraction du NIH (Not Invented Here)

Le langage machine

- langage de plus bas niveau (initialement le seul)
- programmes **directement exécutables par la machine**
  - programmation fastidieuse
  - programmes non portables

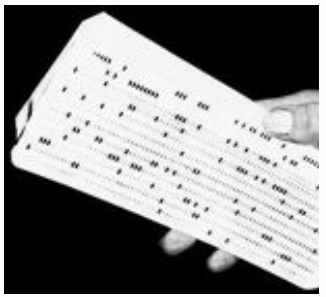
Le calcul de la factorielle, en code machine Intel

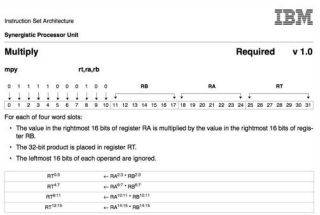
```
10111000  00000001  00000000  00000000  00000000
10111010  00000010  00000000  00000000  00000000
00111001  11011010
01111111  00000110
00001111  10101111  11000010
01000010
11101011  11110110
11000011
```


Le langage machine

- langage de plus bas niveau (**initialement le seul**)
- programmes directement exécutables par la machine
  - **programmation fastidieuse**
  - **programmes non portables**

Grace Hopper





Instruction Set Architecture — IBM

Synergistic Processor Unit

Multiply — Required — v 1.0

mpy rT,rA,rB

Le langage assembleur : représentation textuelle du langage machine

- utilisation de noms *mnémoniques* pour les instructions, les registres, etc
- utilisation de noms *symboliques* pour désigner les points dans le programme, les cases de la mémoire, etc
- possibilité de mettre des commentaires sur le code

Le calcul de la factorielle, en langage assembleur Intel

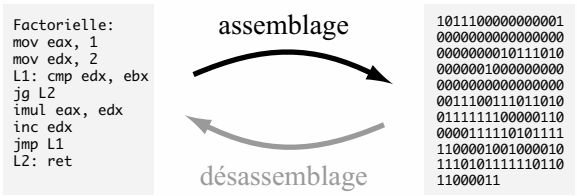
```
;; Calcul de la fonction factorielle
;; En entree:  l'argument N est dans le registre EBX
;; En sortie:  la factorielle de N est dans le registre EAX

Factorielle:
      mov eax, 1      ;; resultat initial = 1
      mov edx, 2      ;; indice de boucle = 2
L1:   cmp edx, ebx    ;; tant que indice <= N ...
      jg L2
      imul eax, edx   ;; multiplier le resultat par l'indice
      inc edx         ;; incrementer l'indice
      jmp L1          ;; fin tant que
L2:   ret             ;; fin de la fonction Factorielle
```


Le langage assembleur : représentation textuelle du langage machine

- utilisation de noms *mnémoniques* pour les instructions, les registres, etc
- utilisation de noms *symboliques* pour désigner les points dans le programme, les cases de la mémoire, etc
- possibilité de mettre des commentaires sur le code

Compilation du langage assembleur

- calcul des adresses mémoire correspondant aux noms symboliques
- encodage sous forme de nombres des mnémoniques d'instructions et de registres
- production d'un listing, d'un index de références croisées, etc, pour faciliter la relecture du code



Évaluation de formules mathématiques

$$x_1, x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

FORTRAN (FORmula TRANslator)

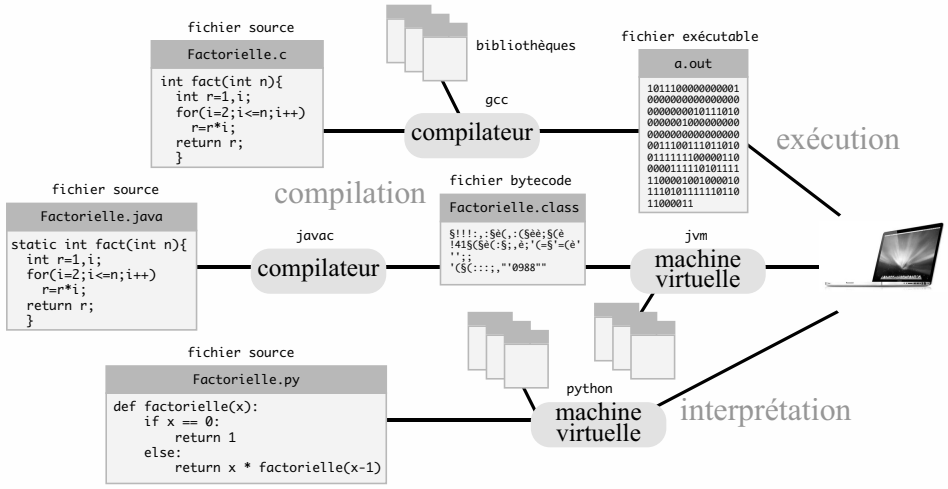
```
D = SQRT(B*B - 4*A*C)
X1 = (-B + D) / (2*A)
X2 = (-B - D) / (2*A)
```

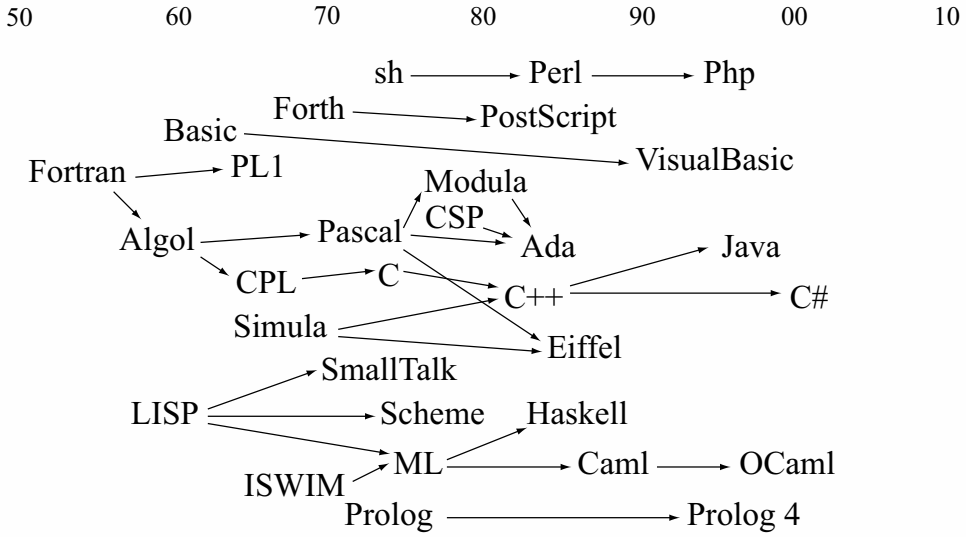
Assembleur

```
mul t1, b, b
mul t2, a, c
mul t2, t2, 4
sub t1, t1, t2
sqrt d, t1
mul t3, a, 2
sub x1, d, b
div x1, x1, t3
neg x2, b
sub x2, x2, d
div x2, x2, t3
```

Les langages évolués

- plus faciles à comprendre par l'humain
- indépendants des machines
- les opérations de *compilation et/ou interprétation* en permettent l'exécution par une machine





`http://www.oreilly.com/news/graphics/prog_lang_poster.pdf`
`http://www.digibarn.com/collections/posters/tongues/`

Pourquoi tant de langages ?

- beaucoup d'aspects à traiter ensemble
  - données, actions = programming in the small
  - architecture = programming in the large
- beaucoup d'innovations successives
  - fonctionnel, polymorphisme, modules, objets, parallélisme, etc
- énormément de compromis possibles
  - plusieurs grandes classes et beaucoup de dialectes

Raisons du succès (dans le temps)

1	S'adresser à un large public	C (associé à la popularité d'Unix)
2	Faire le job	Cobol (conçu pour l'écriture de rapports)
3	Offrir une nouvelle fonctionnalité	Java (écrit une fois, exécuté partout)
4	Combler un créneau	Mathematica (approprié aux calculs complexes)
5	Offrir un minimum d'élégance	Icon (doté d'une syntaxe familière)
6	Fédérer autour d'un leader charismatique	Perl (élaboré par Larry Wall)

Les composants d'un langage

- des principes de calcul
  - impératif, fonctionnel, logique, temporel, etc
- des principes d'architecture
  - modularité, héritage, polymorphisme, etc
- une syntaxe et un style syntaxique
  - détermine les programmes bien construits
- un systèmes de types
  - évite d'additionner des choux et des carottes
- une sémantique plus ou moins formelle
  - définit le sens des programmes
- un outillage
  - compilateurs, débogueurs, manuels, exemples, librairies, interfaces avec d'autres langages
- une communauté d'utilisateurs

La programmation impérative

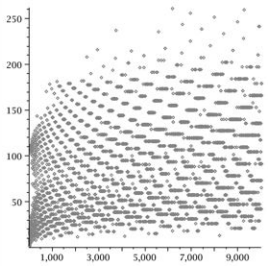
- basé sur l'effet de bord permettant de remplacer en mémoire une valeur par une autre (opération appelée affectation)
- utilisation de noms symboliques pour les adresses
- un programme est une suite de traitements exécutés séquentiellement
- possibilité de
  - réaliser des tests
  - itérer des traitements
  - modulariser le traitement (sous-programmes, procédures, fonctions)

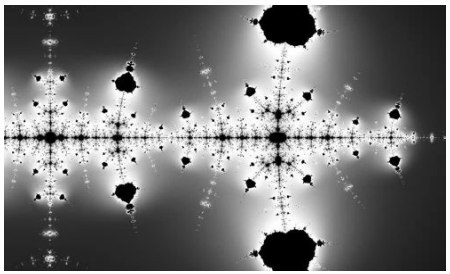
Définition itérative de factorielle (en C)

```
int fact (int n) {
  // factorielle de n
  int r = 1, i;
  for (i = 2; i <= n; i++)
    r = r * i;
  return r;
}
```

Longueur & hauteur dans le problème de Collatz (en Java)

```
static int[] collatz (int n) {
  // longueur et hauteur
  int[] t = new int[2]; t[0] = 0; t[1] = n;
  while (n>1) {
    if (n % 2 == 0) n = n / 2;
    else n = 3 * n + 1;
    t[0]++;                          //maj longueur
    if (n > t[1]) t[1] = n;          //maj hauteur
    }
  return t;
}
```





La programmation fonctionnelle

- utilise l'évaluation de fonctions comme opération de base
- se fonde sur la théorie du λ-calcul (branche de la logique mathématique étudiant la notion générale de fonction indépendamment de ses domaines)
- privilégie la notion de **calcul** et interdit ou limite les effets de bords
- s'appuie sur la notion d'**environnement** (ensemble de couples [nom,valeur]), dans lequel une paire ne peut pas être modifiée une fois créée

Définition récursive de factorielle (en Caml)

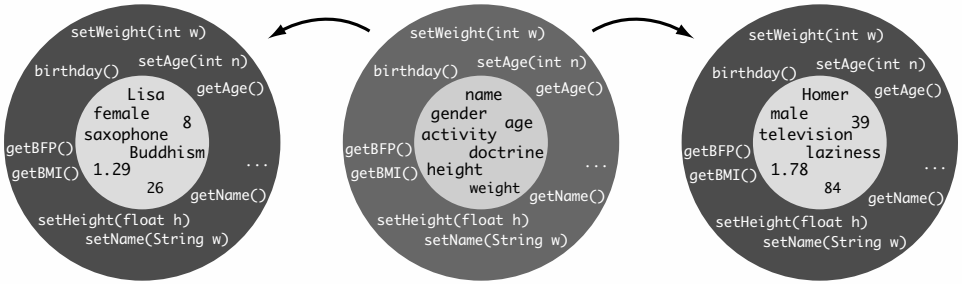
```
let rec fact n =
  if n < 2 then 1 else fact(n - 1) * n ;;
```

La fonction d'Ackerman-Péter (en Python)

```
def ackermann_peter(m, n):
    if m == 0:
        return n + 1
    elif n == 0:
        return ackermann_peter(m - 1, 1)
    else:
        return ackermann_peter(m - 1, ackermann_peter(m, n - 1))
```


La programmation orientée objet

- permet d'améliorer la gestion et la qualité des grands projets logiciels
- consiste à créer une modélisation des éléments du monde réel comme *objet*
  - un identifiant qui permet de définir une référence unique vers un objet
  - ses **attributs** qui contiennent des données qui caractérisent l'état de l'objet
  - ses **méthodes** qui représentent les actions possibles pour l'objet



La programmation logique

- utilise l'expressivité de la logique
- s'appuie sur les concepts d'unification, de récursivité et de retour sur trace
- demande la construction d'une base de connaissances (prédicats et règles)
- permet la formulation de requêtes à la base

Exemple en Prolog

```
Pere (Homer, Bartholomew).
Pere (Abraham, Homer).
Pere (x,y), Pere (y,z) :- GrandPere (x,z).

?- GrandPere (GP, Bartholomew).

> GP = Abraham
```