Algorithmique du traitement des données

USID07 – Master MEDAS

Cours 2

Auteur : Raphaël Fournier-S'niehotta (fournier@cnam.fr)

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Plan

Informatique

- Provient de la contraction de Information et Automatique
- Matériel (ordinateur) / Logiciel (programmes)
- Le rôle des humains ? Définir comment les informations vont être traitées (automatiquement) par des ordinateurs
- On écrit pour cela un ensemble de règles : un programme

E. Dijkstra

La science informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des téléscopes

Algorithme

- vient du nom latinisé du mathématicien perse Al-Khawarizmi
- un algorithme définit le comportement du programme : c'est la série d'instructions qui peut être exécutée
- un programme, pour être compréhensible par un ordinateur, doit être écrit dans un langage de programmation
- il existe de nombreux langages, selon les besoins (nous y reviendrons)
- un algorithme exprime la structure logique d'un programme, il est indépendant du langage de programmation
- on peut écrire les algorithmes dans un langage de descriptions des algorithmes, le pseudo-code

Les programmes

Programme méthode de résolution d'un problème,
sous la forme d'une *suite d'instructions*.

Instruction ordres à exécuter par la machine,
écrites en *langage de programmation*.

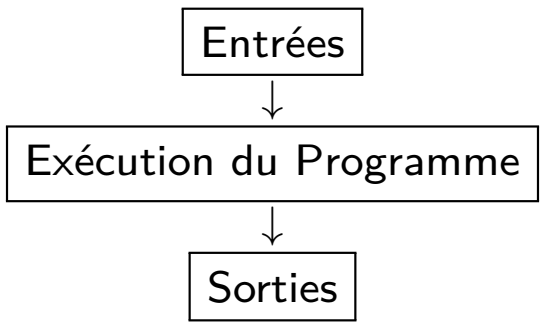
Exécution (d'un programme) application de chacune des instructions du programme.

Un programme traite des données

Un programme transforme ses données (entrées) pour aboutir à une solution (sorties).

Entrées données initiales du problème à résoudre.

Sorties données obtenues après exécution du programme.



Intermède (1) : le CPU

CPU (Unité centrale ou processeur) : exécute les instructions en **langage machine** (logées en mémoire centrale).

langage machine ensemble d'instructions très simples, en code binaire, propre à chaque type de machine, et que le CPU sait exécuter.

PC (Program Counter) C'est un registre du CPU qui contient l'adresse de la prochaine instruction à exécuter.

Intermède (2) : mémoire centrale (RAM)

RAM (Random Access Memory)

- Mémoire très rapide contenant programmes et données.
- composée **d'emplacements** identifiés chacun par une adresse.

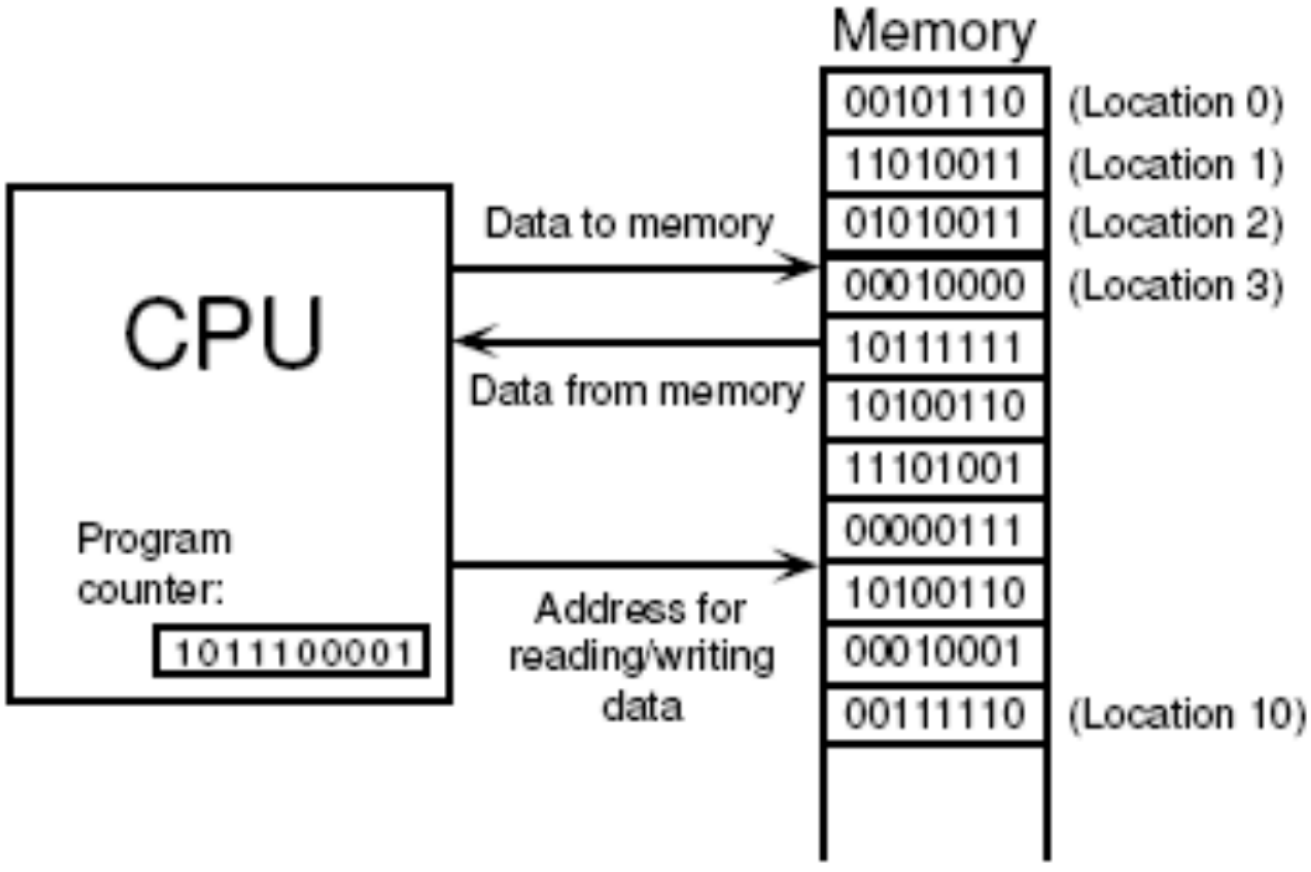
  Trouver instruction ou donnée ⇒ chercher à son adresse.

  Stocker une information ⇒ à l'adresse où elle doit aller.

Intermède (3) : exécution d'un programme

1. Instructions programme + ses données $\Rightarrow$ stockées en mémoire centrale.
2. Le CPU récupère l'instruction à l'adresse donnée par le PC, puis l'exécute.
3. Il répète cela pour l'instruction suivante (jusqu'à la fin).

Intermède (4) : exécution par le CPU



Langages de programmation de haut niveau

Langages de haut niveau (C, Ada, Pascal, Java, Python).
Langages de bas niveau (Intel/PC, PowerPC, Sparc/Sun)

- instructions sophistiqués qui facilitent l'écriture des programmes;
- lisibles pour un humain;
- impossibles à exécuter par le CPU.

$\Rightarrow$ traduire en langage machine avant exécution.

Eléments principaux de langages

syntaxe : règles d'écriture pour instructions et programmes.

Exemple : pour écrire l'énoncé mathématique

$$1 \leq x \leq 7$$

une syntaxe possible en Python :
$1 <= \mathrm{x} \text{ and } \mathrm{x} <= 7$.

types décrivent la nature des données.

Exemple : La donnée 1 avec type int de Python
$\Rightarrow$ nombre entier.

Eléments principaux de langages (suite)

sémantique : *(sens) comportement à l'exécution* pour les instructions.

Ex : a!=b signifie «tester si a est différent de b».

typage : *contraintes de cohérence* entre types.
Ex : "bonjour" / 2 :

- syntaxiquement correct,
- sémantiquement erroné :
  car / non applicable sur "bonjour".
- $\Rightarrow$ erreur de typage.

Traducteur de programmes

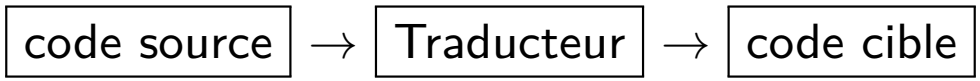
Programme qui traduit les instructions en langage de programmation de haut niveau vers instructions en langage machine.

code source (ou programme source) : fichier avec le programme à traduire.

code cible résultat de la traduction (binaire ou texte).

bytecode : (ou pseudo-code) instructions d'assez bas niveau, appelé aussi «code intermédiaire», non exécutable directement par le CPU, mais assez proches du langage machine.

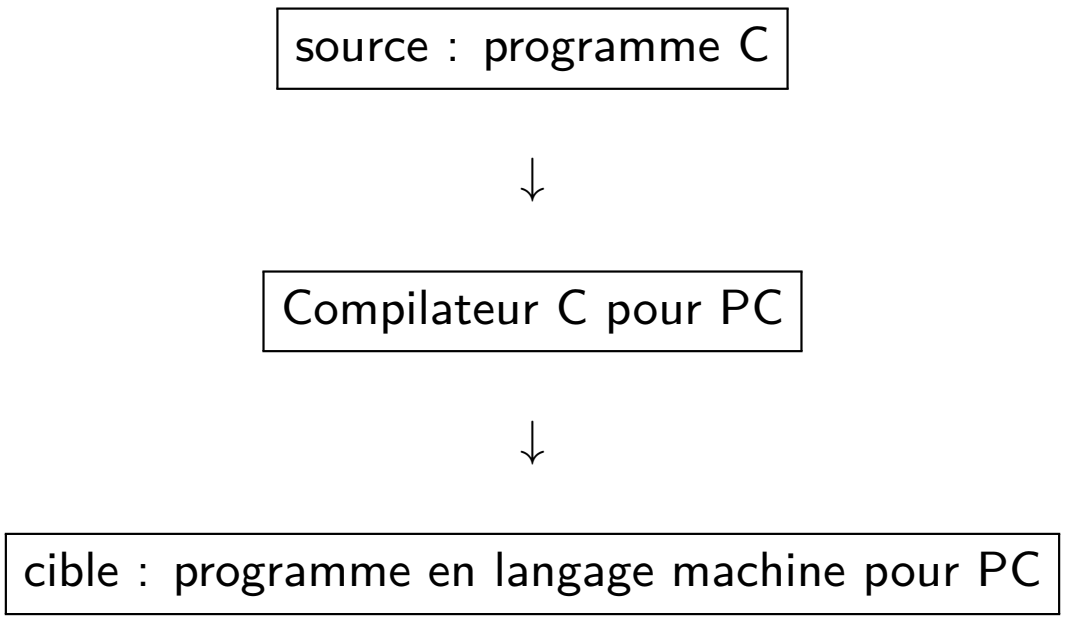
Deux sortes de traducteurs



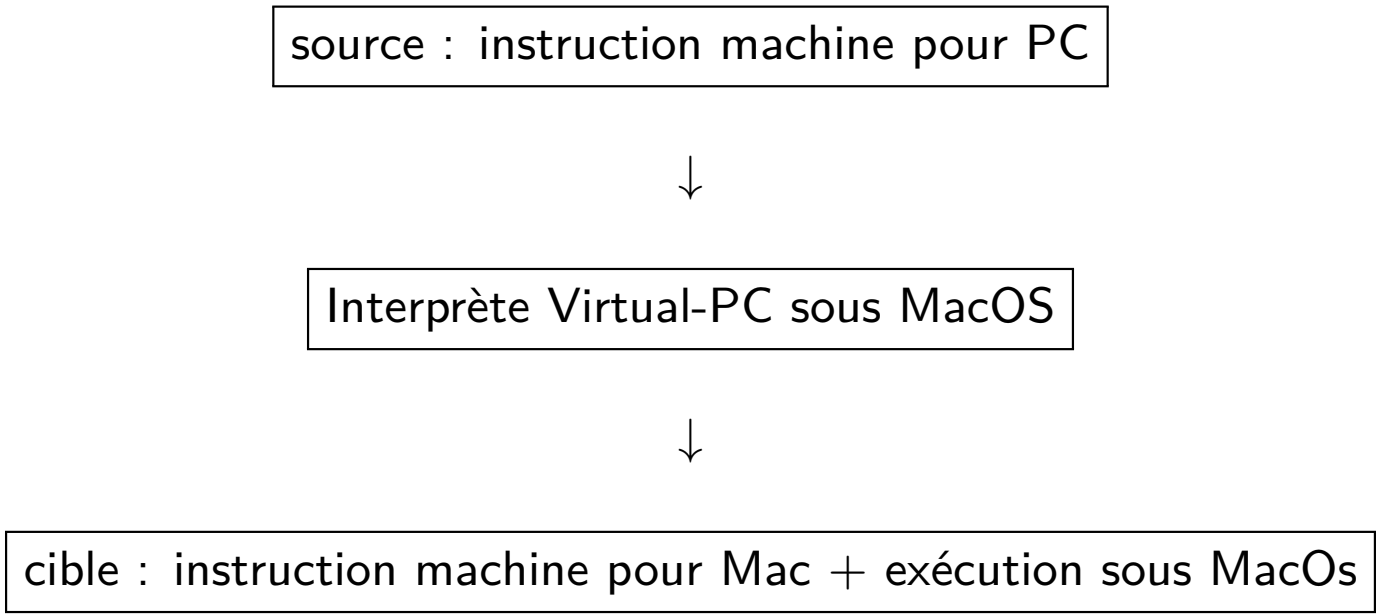
compilateur traite le fichier source dans sa totalité et produit en résultat un fichier avec du code cible. Le compilateur **n'exécute** pas le fichier cible obtenu en résultat.

interprète traduit une instruction source à la fois **et** l'exécute avant de passer à la suivante. Ne produit pas de fichier cible en résultat.

Exemple : compilateur C



Exemple : interprète Virtual PC sous MacOs



Traduction et exécution de programmes Python

Interprète Les instructions du programme Python sont interprétées dans un langage compréhensible par la machine.

Exécution

Le programme Bonjour

Ce programme affiche «Bonjour !!»

```
# Premier programme (dans fichier Bonjour.py)
print("Bonjour!!")
```

- Le texte entre après # est un commentaire
- Ce programme ne manipule qu'une donnée : le message

  Bonjour !!

  affiché à l'écran lors de l'exécution.

Exécution de Bonjour

- se fait en appelant l'interprète
- avec la commande python
- suivie du nom du fichier contenant le programme (ici Bonjour.py).

On voit s'afficher le message attendu :

```
python bonjour.py
Bonjour !!
```

Erreurs de programmation

Introduction d'erreurs dans le nouveau fichier Bonjour2.py :

```
# Premier programme (dans fichier Bonjour.py)
prnit("Bonjour!!"/2)
```

- erreur de syntaxe : le mot-clé print a été changé en prnit.
- erreur de typage (sémantique) : "Bonjour" est une chaîne de caractères (type str) et ne peut être divisé par 2.

Erreurs de programmation

```
# Premier programme (dans fichier Bonjour.py)
prnit(”Bonjour!!”/2)
```

L'exécution s'arrête à la première erreur (ligne 1).

Nous corrigeons et relançons la compilation :

```
Traceback (most recent call last):
  File ”Bonjour2.py”, line 1, in <module>
    prnit(”Bonjour!”/2)
NameError: name 'prnit' is not defined
```

Ce message dit que à la ligne 1, l'opérateur prnit n'existe pas

Erreurs de programmation

Corrigeons :

```
# Premier programme (dans fichier Bonjour.py)
print(”Bonjour!!”/2)
```

L'exécution s'arrête à la première erreur (ligne 1).

```
Traceback (most recent call last):
  File ”Bonjour2.py”, line 1, in <module>
    print(”Bonjour!”/2)
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Ce message dit que à la ligne 1, l'opérateur / ne peut marcher sur une chaîne de caractère et un entier

Les algorithmes

problème énoncé simple d'un problème à résoudre.

algorithme description d'une méthode de résolution
via le calcul d'une solution.

langage français, pseudo-code, ou pseudo-mathématique.

données entrées initiales de l'algorithme, sorties ou solutions .

Les algorithmes (suite)

entrées ces sont les *informations nécessaires* à l'algorithme pour résoudre le problème.

sorties c'est ce que l'on veut calculer.

formulation décrire comment

- calculer les sorties
- en partant des entrées,
- par une suite de pas à exécuter *mécaniquement* par un calculateur (humain ou machine)

Exemple : calcul de l'impôt

La déclaration d'impôts sur le revenu vient avec une notice décrivant les opérations à réaliser pour calculer le montant de l'impôt :

- problème : calculer le montant de votre impôt
- entrées : montants des salaires, charges, abattements, etc.
- algorithme : la suite des calculs à réaliser sur les entrées.
- résultat ou sortie : le montant de impôt calculé.

Exemple : recette de cuisine

Problème : Préparation d'une omelette

Entrées : Ce sont les ingrédients,

- 2 oeufs,
- sel,
- un peu de matière grasse

Sorties : Une omelette pour 1 personne.

Algorithme : C'est la méthode de préparation,

1. Casser les oeufs dans un bol,
2. Y ajouter du sel, puis les battre,
3. Faire chauffer la matière grasse dans une pôele,
4. Verser le mélange des oeufs dans la pôele et faire cuire doucement jusqu'à la consistance souhaitée.

Exemple : conversion euros/dollars

Problème : Calculer et afficher la conversion en dollars d'une somme en euros saisie au clavier.

Les données :

Entrées : un nombre réel x saisit au clavier,
Sorties : un réel z tel que $z = x * 1.10$

Algorithme :

1. lire la valeur saisie pour x,
2. calculer $z = x * 1.10$
4. afficher le résultat final z

Importance d'un bon algorithme

Compréhension et modélisation du problème

- L'algorithme modélise le comportement du programme

Correction du résultat

- Algorithme faux : résultat du calcul faux

Efficacité du calcul

- Coût d'un calcul = nombre d'opérations et quantité d'information manipulée
- Ces quantités sont définies par l'algorithme
- Algorithme efficace : calcul efficace (et inversement)

L'étude algorithmique d'un problème est indispensable

La bonne conception d'un algorithme est fondamentale et absolument nécessaire préalablement à l'écriture d'un programme informatique

Produire des programmes

1. **Analyse** : Énoncé détaillé du problème $\Rightarrow$ on obtient un cahier des charges.
2. **Conception** : Choix de représentation des données (nombres entiers, réels, tableaux, bases des données?), puis conception d'une méthode de résolution
   on obtient $\Rightarrow$ Données + Algorithme.
3. **Codage** : L'algorithme est traduit dans un langage de programmation $\Rightarrow$ on obtient le programme source.
4. **Mise au point** : Compilation, tests, correction d'erreurs.
5. **Évolution** On fait évoluer le programme par des améliorations, corrections ou extensions.

Petite méthodologie de conception

Pour écrire un programme il faut suivre les étapes suivantes :

1. Déterminer les *informations nécessaires* pour résoudre le problème. Ce sont les *données* ou encore les *entrées* du programme.
2. Déterminer ce que l'on veut calculer. Ce sont les *résultats* ou *sorties* du programme
3. *Etre capable soi même* de calculer les sorties à partir d'exemples d'entrées. Ceci est une étape de *compréhension* du problème.
4. Décrire comment on calcule les sorties par rapport aux entrées. Ceci est *l'algorithme* de résolution.
5. Coder cet algorithme en Python.

Tester un programme

1. Élaborer un jeu de tests représentatif de tous les cas possibles (univers) des entrées.

   Un *jeu de tests* consiste en une suite de "valeurs typiques" pour chacune des entrées, accompagnées des valeurs attendus comme résultat dans ce cas.

   Par exemple, si on veut tester un programme qui calcule le carre d'un nombre entier, avec entrée x (entier) et sortie z (entier); un jeu de tests (non exhaustif) peut être $\{(x \leftarrow 0, z \mapsto 0); (x \leftarrow 1, z \mapsto 1); (x \leftarrow 3, z \mapsto 9)\}$

2. Confronter le fonctionnement de l'algorithme avec le jeu de tests proposé

Analyse pour la conversion euros/dollars

1. *Énoncé du problème* : Calculer et afficher la conversion en dollars d'une somme en euros saisie au clavier.

2. *Données + Algorithme*
   1. entrées : un nombre réel x saisit
   2. sorties : un réel z tel que $z = x * 1.10$
   3. l'algorithme
      1. lire la valeur saisie pour x,
      2. calculer $z = x * 1.10$
      4. afficher le résultat final z

3. un jeu de tests :
   $(x \leftarrow 0.0, z \mapsto 0.0); (x \leftarrow 15.0, z \mapsto 17.05);$
   $(x \leftarrow -3.5, z \mapsto -3.85); (x \leftarrow 10, z \mapsto 11);$

Codage en Python

- **Données** :
  entrées : x nombre réel
  sortie : z nombre réel

- **Algorithme** :
  1. lire x
  2. calculer $z = x * 1.10$
  3. afficher le résultat se trouvant dans z

```
x = input("Somme en euros? ")
z = float(x) * 1.10
print("En dollars : "+str(z))
```

Suite et fin de l'exemple

Le code est mis dans le fichier Conversion.py.

```
python Conversion.py
Somme en euros? 10
En dollars 11
```

Tests : Nous répetons plusieurs fois l'exécution avec différents valeurs des test.

```
python Conversion.py
Somme en euros? -3
En dollars -3.3000000003
python Conversion.py
Somme en euros? 15.5
En dollars 17.05
```

Aspects fondamentaux de la programmation

Programmes = Données + Instructions

- Données ⇒ comprendre variables + types.
- Instructions ⇒ comprendre structures de contrôle + sous-programmes.

Aspects fondamentaux de la programmation (2)

Savoir écrire des programmes $\Rightarrow$ comprendre:

- syntaxe des instructions:
  $\Rightarrow$ *comment agencer mot-clés et symboles*.

- sémantique des instructions:
  $\Rightarrow$ *comportement à l'exécution*.

Variables et types: un aperçu

```
int x = 7;
```

Variables: C'est quoi? (sémantique):

- un nom dans le programme (ici x),
- associé à un emplacement de la mémoire,
- qui contient une donnée (ici 7),
- et qui a un type associé (ici int);

et un type? décrit la nature d'une donnée. Le type int signifie «nombre entier».

Instructions: un aperçu

Ex: une instruction «d'affectation»

```
x = 10 + 3;
```

sémantique Ordre à exécuter.

Ici: «*calculer 10+3, puis, enregistrer le résultat dans l'emplacement associé à la variable x*».

structure de contrôle: un aperçu

Ex: une boucle

```
for i in range(10):
  print(i)
```

sémantique Instruction qui change le «flux d'exécution», c.a.d., le fait que les instruction du programme s'exécutent séquentiellement.

Ex: la boucle «for» permet de «répeter» des instructions.

boucle Instruction de répétition.

Sous-programmes: un aperçu

Ex: utilisation du sous-programme d'affichage d'un entier

```
Math.sqrt(x);
```

Un **sous-programme** est:

- une suite d'instructions (non montrée ici)
- que l'on regroupe sous un nom (ici: `Math.sqrt()`).
- correspondant à une tâche à exécuter(ici, calculer la racine carré de x passé en argument),
- Quand ce nom apparaît dans le programme, les instructions du sous-programme sont exécutées.

Résumé du chapitre

- Les ordinateurs sont composés d'un processeur (CPU), d'une mémoire vive (RAM), de périphériques
- Le CPU exécute les programmes, à l'aide d'instructions qui sont progressivement stockées et lues dans la mémoire vive.
- Un programme est une méthode mécanique de résolution de problème.
- Il travaille sur des données en entrées et produit une/des sorties.
- Un programme se compose de suite d'instructions, exprimées à l'aide d'une syntaxe, propre à chaque langage
- les langages de haut niveau doivent être traduits d'un langage compréhensible par des humains au langage machine
- Un algorithme pour la résolution d'un problème est un énoncé détaillé sous forme de pas à suivre pour calculer les solutions ou sorties pour le problème à partir de ses données d'entrée.

Résumé du chapitre (2/2)

- en Python, les intructions sont interprêtées sur la machine d'exécution
- Il y a plusieurs phases pour concevoir des programmes :
  - Analyse et conception
  - Codage
  - Mise au point