Programmation I : Présentation générale

Jamila Sam

Laboratoire d'Intelligence Artificielle
Faculté I&C

Plan

- Présenter le cours :
  - Objectifs (« Quoi ? »)
  - Administration (« Comment ? »)
- Présenter l'environnement de travail
- Introduire la programmation : notion d'algorithme

Objectifs du cours

1. Apprendre à **programmer**
   savoir les bases et connaître correctement au moins un langage
   - ☞ pratique sur le langage C++
2. Savoir comment **utiliser** un **ordinateur** (sous Linux) dans le cadre du développement de programmes

Présentation générale du cours

Public : Cours obligatoire pour les étudiants du 1er semestre de la section des Sciences de la Vie. Connaissances supposées acquises : aucune

Langue : Français

Moyens :

Concepts théoriques introduits ou complémentés lors de **cours** magistraux ex-cathedra

☞ Mercredi 15^{15}–16^{00}/17^{00}

mis en pratique, de manière guidée, lors de **séances d'exercices** sur machines

☞ Jeudi 17^{15}–19^{00}

Compléments en lignes : **vidéos** et **quizzes** (disponibles pour 8 semaines du cours).

Présentation générale du cours

http://moodle.epfl.ch/course/view.php?id=5981

Horaires et Contenu : Un planning détaillant le contenu de chaque séance est disponible sur le site internet du cours.

Encadrement : Deux assistants et 12 assistants-étudiants (voir également le site internet du cours)

Interaction avec les enseignants

Plusieurs moyens pour contacter les enseignants, assistants et étudiants-assistants pour poser des questions sur le cours ou les exercices :

- Durant les séances d'exercices :
  - ☞ c'est le moyen le plus direct, et généralement le plus efficace.
- Par l'intermédiaire du forum (dans site Moodle)
  - ☞ moyen idéal pour diffuser la connaissance

  N'hésitez pas à en faire usage !
- par email aux assistants :
  - ☞ mais pour les cas généraux, préférez le forum.

Les contacts personnels avec l'enseignant (email, téléphone ou visites) devront être **strictement réservés aux cas personnels et/ou urgents !**

Support de cours

Tout le matériel est accessible via le **site Web** du cours :

- Vidéos et quizzes
  offerts par le cours massif en ligne (MOOC) associé au cours (voir plus loin)

- Transparents
  parfois enrichis de notes techniques (mini-références) détaillant certains concepts évoqués pendant le cours, en particulier les éléments du langage C++
  également parfois des références complémentaires (bibliographiques et/où hyperliens Internet)

- Énoncé des exercices
  disponibles sur le site Web en fin de semaine.

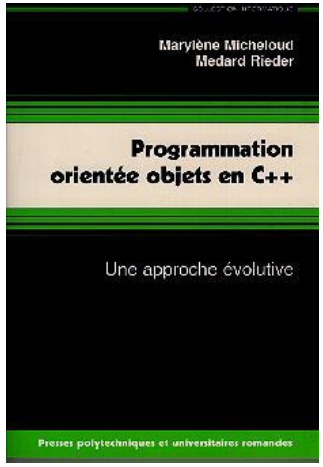
- Corrigé des exercices
  disponibles sur le site Web en fin de semaine suivante.

Support de cours

Ces éléments devraient constituer une **documentation suffisante** pour ce cours !

Si vous souhaitez la compléter, les ouvrages suivant sont également recommandés

Marylène Micheloud & Medard Rieder Programmation orientée objets en C++ – une approche évolutive, PPUR, 1997.



Il est disponible pour un prix avoisinant les 43 CHF.

Support de cours

Ces éléments devraient constituer une **documentation suffisante** pour ce cours !

Si vous souhaitez la compléter, les ouvrages suivant sont également recommandés

J.-C. Chappelier & F. Seydoux C++ par la pratique – recueil d'exercices corrigés et aide-mémoire, PPUR, 3ème édition. Empruntable en version électronique auprès de la bibliothèque

Il est disponible pour un prix avoisinant les 50 CHF.

Couplage au MOOC (1)

MOOC d'initiation à la programmation en C++ :

https://www.coursera.org/learn/init-prog-cpp/

Notre cours dispose de ses propres séries d'exercices et de transparents de complément

- ☞ Sur-ensemble du MOOC

Matériel MOOC utilisé :

1. Vidéos
2. Quizzes
3. Devoirs (mais ne comptent pas)
   - ☞ à utiliser pour se préparer aux tests

Couplage au MOOC (2)

- Avant le cours : visionner les vidéos, faire les quizzes et comprendre certains exercices de niveau 0
- Cours ex-cathedra : résumé et approfondissements
  - ☞ **seulement une heure**
- Exercices : mise en pratique
- Certificats (payants) : **en aucun cas obligatoires pour ce cours**

Couplage au MOOC (3)

Charge de travail :

- 1 heure de cours ex-cathedra : récapitulation et approfondissements ;
- 2 heures d'exercices en salle de TP : mise en pratique;
- 5 heures de travail à la maison:
  - 1:30-1:45 sur les vidéos de la semaine suivante
  - 0:15-0:30 sur les quizzes de la semaine suivante
  - 3 heures pour commencer à préparer la série d'exercices de la semaine en cours, finaliser celle de la semaine passée.

Notes et examens

Les épreuves de contrôle continu seront les suivantes :

- Série notée **individuel**, 1h15
- Examen théorique **individuel**, 1h45

Calcul de la note

- La note finale du semestre, N, est calculée comme suit :

$$N = \frac{(N_{serieNotee} + 2 * N_{examen})}{3}$$

- Les notes intermédiaires ne sont pas arrondies.

- Les cours Programmation I et II sont devenus indépendants. La moyenne arrondie de chaque cours est transmise au SAC à la fin de chaque semestre.

Notes et examens
Série notée

Objectif : vérifier la maîtrise des concepts du langage C++ exposés en cours.

Séance d'exercices, à l'issue de laquelle le travail réalisé est envoyé par courrier électronique aux assistants responsables.

Réalisée individuellement

La série notée aura lieu le :

Jeudi 16 Novembre

Notes et examens
Examen

Le semestre sera clôturé par un examen écrit portant sur le contenu du cours et les séances d'exercices.

Date :

Mercredi 20 Décembre

Aspects logiciels d'un ordinateur

Pour fonctionner, un ordinateur doit pouvoir interagir avec l'environnement :

- comprendre, c'est-à-dire ici traiter, les informations lui provenant (clic de souris, touche clavier, . . .)
- produire des sorties (sons, image écran, . . .)

Cela se fait grâce à des **programmes** (ou « **logiciels** ») dont le plus fondamental, est le système d'exploitation.

Le système d'exploitation est responsable de la gestion des interactions entre l'unité centrale et ses périphériques,
Exemples : MacOS X, **Linux**, Solaris, Windows...

Catégories de Logiciels

logiciels d'application traitement de tâches spécifiques aux utilisateurs

traitements de textes, tableurs, logiciels de comptabilité, CAO, ...

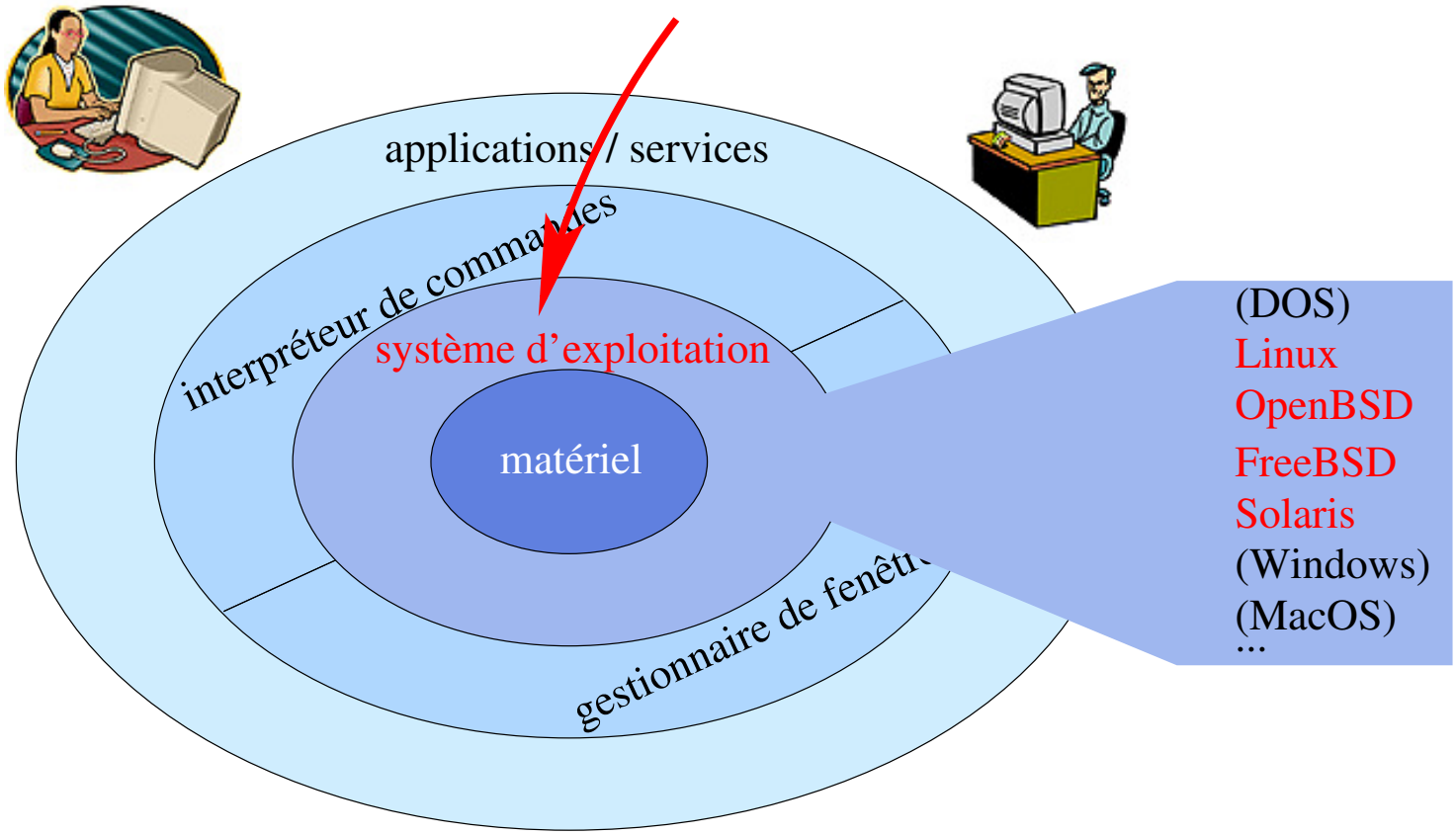
logiciels utilitaires servant au développement des applications

assembleurs, compilateurs, dévermineurs, gestionnaires de versions, gestionnaires de fenêtres, librairies d'outils, ...

logiciels systèmes regroupés dans le **système d'exploitation**

☞ présents au cœur de l'ordinateur, ces logiciels sont à la base de toute exploitation, coordonnant les tâches essentielles à la bonne marche du matériel.

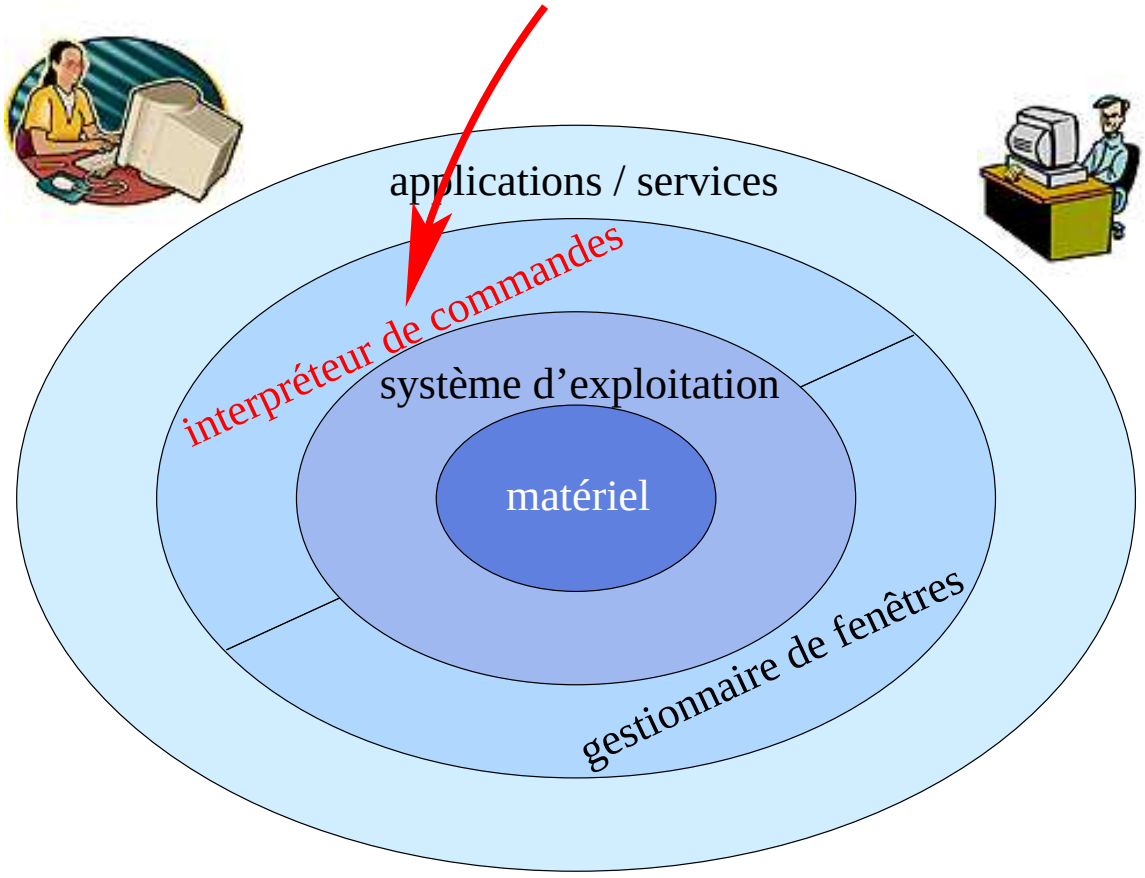
Système d'exploitation



Interaction avec Linux

Comment interagir avec votre système d'exploitation ?

☞ avec un **interpréteur de commande** (« **shell** »)



Parmi les shells Unix les plus utilisés, citons : Bourne [Again] shell (sh et bash), C shell (csh), Z shell (zsh), et celui présent par défaut sur les comptes du cours, l'Enhanced C shell (tcsh).

Interpréteur de commandes (1)

Pour interagir avec l'utilisateur, un système informatique doit disposer au minimum d'un **interpréteur de commandes** (« shell »)

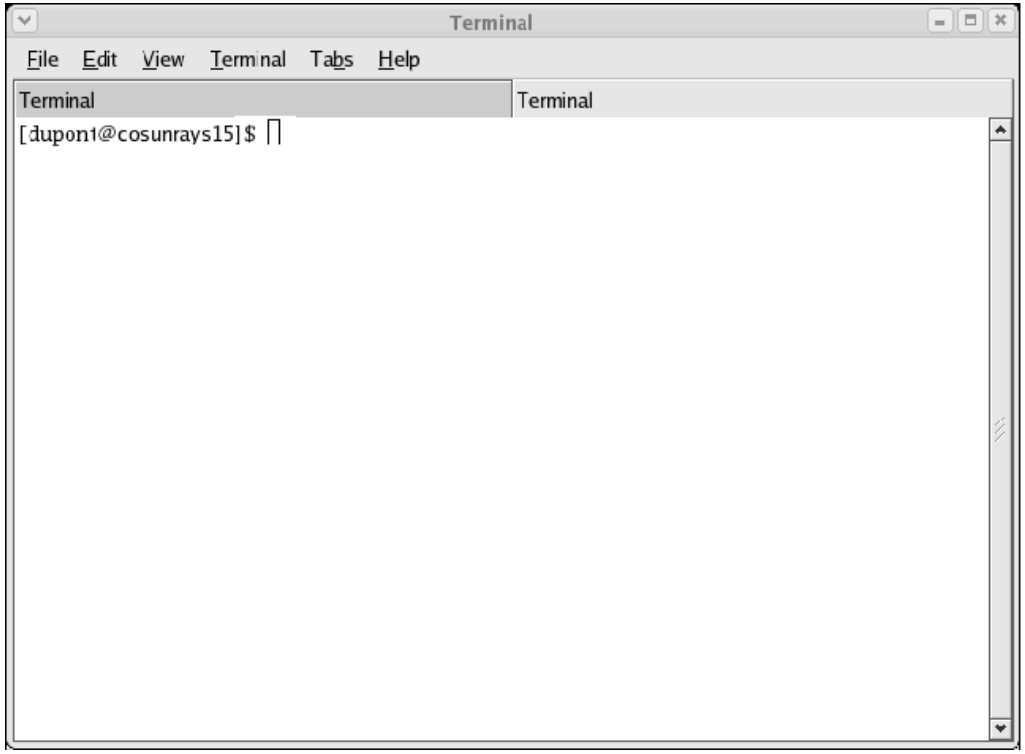
Contrairement à d'autres architectures moins modulaires, l'interpréteur de commandes (ainsi que le gestionnaire de fenêtres) des systèmes de type UNIX est un composant externe au SE.
Ne faisant pas directement partie du système, ils peuvent être changés à souhait.

Le shell attend les ordres que l'utilisateur transmet par le biais de l'interface, décode et décompose ces ordres en actions élémentaires, et finalement réalise ces actions en interagissant avec le système d'exploitation.

Parmi les shells Unix les plus utilisés, citons : Bourne [Again] shell (sh et bash), C shell (csh), Z shell (zsh), et celui présent par défaut sur les comptes du cours, l'Enhanced C shell (tcsh).

Interpréteur de commandes (2)

Comment se présentera concrètement votre interpréteur de commande ?
Dans une fenêtre terminal invoquable soit en cliquant sur une option du menu soit par l'usage de la commande `xterm`



Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Interpréteur de commandes (3)

Depuis un interpréteur de commandes vous aurez la possibilité de :

- lancer des programmes (par exemple un navigateur web, commande `firefox`)
- exécuter des commandes Unix (on peut aussi regrouper plusieurs commandes dans un fichier alors appelé script).
- définir des variables d'environnement,
- renommer ou définir de nouvelles commandes (`alias`), etc...

La plupart des interpréteurs offrent également des facilités d'édition comme le rappel des commandes précédentes (historique des commandes), la complétion (complète le nom du fichier lorsqu'il n'y a plus d'ambiguïté), la correction en cas de commande invalide, ...

La commande `man`

`man` permet d'accèder à l'aide du sytème (« page de `man`uel »)

Utilisations :

`man` *nom*

`man` *section nom*

Exemples : `man tcsh` `man ls` `man man`

Les man-pages sont organisées en différentes **sections** :

1	commandes et programmes	5	formats de fichiers
2	appels systèmes (noyau)	6	jeux
3	bibliothèques logicielles	7	divers
4	fichiers spéciaux (`/dev`)	8	administration système

Comparer :

`man printf`	et	`man 3 printf`
`man time`	et	`man 2 time`
`man man`	et	`man 7 man`

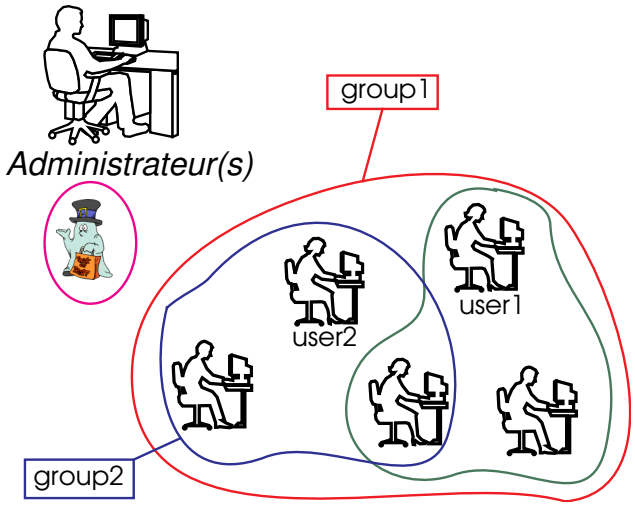
`man -a` *nom* pour avoir **toutes** les man-pages portant sur ce *nom*.
('`q`' pour quitter une manpage et passer à la suivante)

Utilisateurs et groupes

Architectures multi-utilisateurs

- identifier les personnes pouvant travailler avec le système
- assurer la confidentialité de leurs données
- parfois, leur facturer les ressources utilisées

☞ 2 entités : les **utilisateurs**, et les **groupes** d'utilisateurs



Les groupes permettent de définir des droits communs à un ensemble d'utilisateurs (quelles ressources utilisables, dans quelles limites, quels droits d'accès, ...)

☞ Chaque utilisateur appartient à au moins 1 groupe

(Essayez la commande `groups`)

Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Système de fichiers

Le concept de **fichiers** est une structure adaptée aux mémoires de masse permettant de regrouper des données.

Un fichier c'est une **collection ordonnée de données**, représentant une entité pour l'utilisateur.

Le **système d'exploitation** va donner corps au concept de fichiers, c'est-à-dire les gérer : les créer, détruire, modifier, lire, et offrir la possibilité de les désigner par des noms.

Dans le cas de systèmes multi-utilisateurs, il faut de plus **assurer la confidentialité** de ces fichiers, en protégant leur contenu du regard des autres utilisateurs.

Système de fichiers (2)

Pour assurer la gestion des fichiers, un système d'exploitation utilise un (voire plusieurs) système(s) de fichiers (« file system »). C'est le système de fichiers qui détermine les structures internes utilisées pour organiser les fichiers.

Parmi les nombreux systèmes de fichiers, citons :

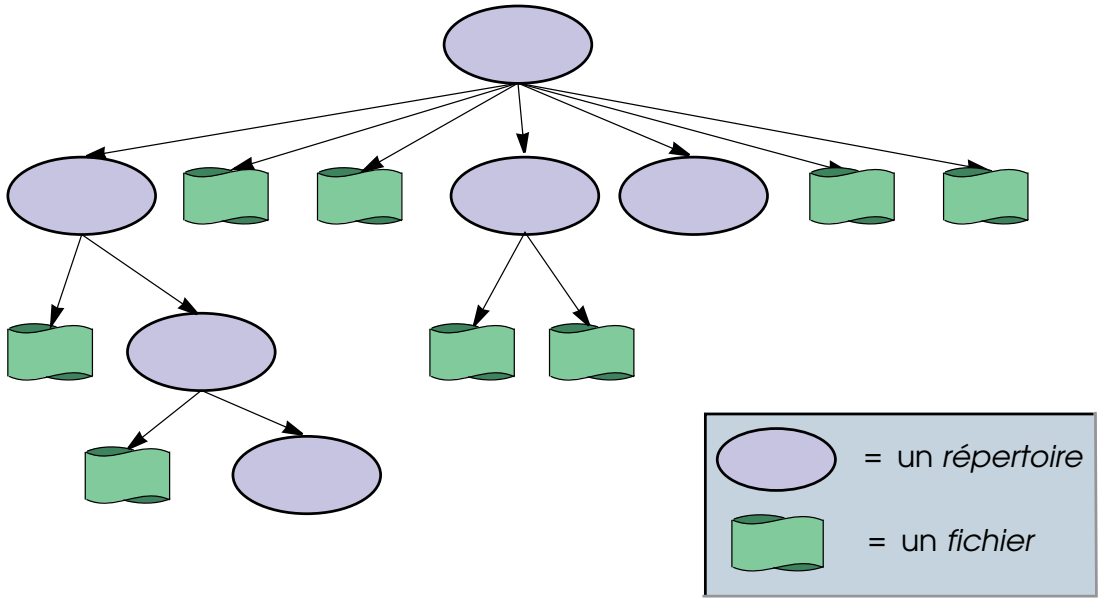
- FAT, VFat, HPFS, NTFS (Dos & Windows),
- Ext, Ext2, Ext3, Ext4, ReiserFS (Linux)
- ISO9660, UDF, UFS, Joliet, RockRidge (pour les CD),
- SystemV, VxFS, Spiralog (Solaris, VMS)
- NFS (Network File System : pour les réseaux)

Structuration d'un système de fichiers

Grand nombre de fichiers

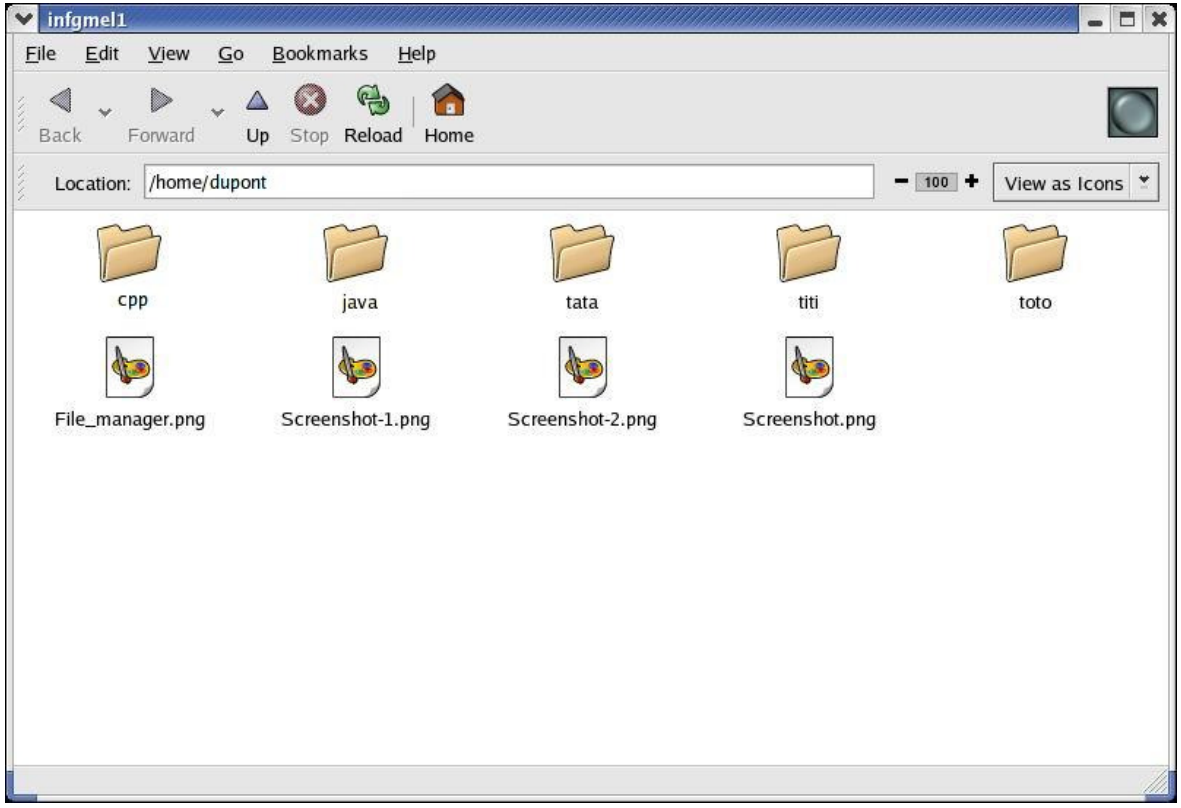
- ☞ fournir un moyen pour organiser ces fichiers
- ☞ concept de **répertoire** (directory)

Un répertoire est une collection (généralement non ordonnée) de fichiers ou de répertoires (alors appelés sous-répertoires).
Ils permettent d'organiser l'ensemble des fichiers dans une **structure arborescente**



Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Structuration d'un système de fichiers (2)



En plus de la notion de répertoire, la plupart des systèmes permettent également de définir des **liens symboliques** vers des fichiers ou des répertoires (« soft links » avec UNIX, ou « raccourcis » dans d'autres systèmes), qui permettent de définir des alias (i.e., autres noms)

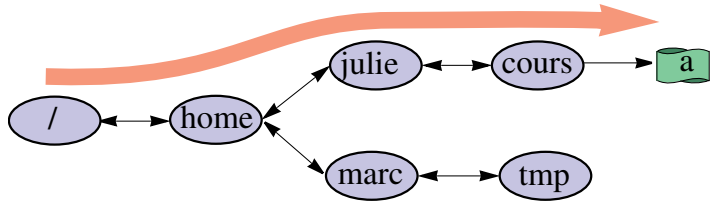
☞ permet d'assouplir la structure d'arbre

Nommage des fichiers : absolu et relatif

On appelle « **chemin** » la succession des répertoires conduisant à un fichier, à partir d'un endroit donné dans l'arborescence.

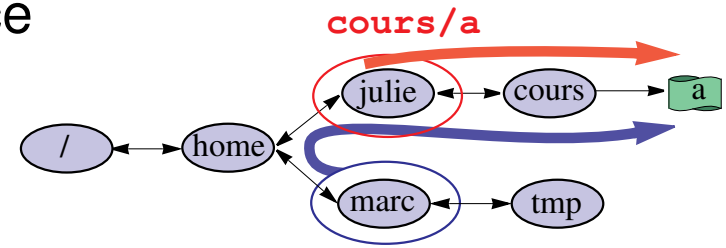
Pour désigner un fichier, il est possible de procéder de deux manières :

- à l'aide d'un **chemin absolu** : on prend comme convention un parcours de l'arbre partant de la racine
  Dans le cas de plusieurs arbres (« forêt »), le nom du lecteur est tout d'abord spécifié (i.e. on désigne la racine de l'arbre).



`/home/julie/cours/a`

- à l'aide d'un **chemin relatif** : c'est la succession des répertoires à traverser, à partir d'un autre répertoire de l'arborescence

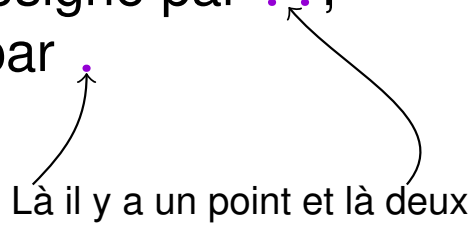


`../julie/cours/a`

Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Nommage des fichiers (2)

Le répertoire parent d'un sous-répertoire est désigné par `..`, tandis que le répertoire lui-même est désigné par `.`



Exemples de noms de fichiers (« chemins ») :

```
/home/prof/Work/cours/Info1/introduction2.tex
../images/paysages.gif
../../../toutlahaut.ps.gz
```

Sous UNIX/Linux, le délimiteur entre nom de répertoire et nom de fichier dans les chemins est la barre oblique « slash » : `/`

D'autres systèmes utilisent l'« antislash » ou « backslash » : `\`

```
D:\Users\Himher\Personnal Documents\introduction2.pdf
```


Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Système de fichiers UNIX/Linux

Chaque utilisateur possède un **répertoire personnel** (« home directory ») dans lequel il peut placer ses fichiers personnels. C'est la racine du sous-arbre réservé spécifiquement à un utilisateur

Les noms de fichiers possèdent généralement une extension, délimitée par un .

Là aussi il y a un point

Cette extension peut être utilisée pour indiquer la nature du fichier, c'est-à-dire l'application à laquelle il est associé. Contrairement à d'autres systèmes d'exploitation, sous UNIX/Linux les fichiers peuvent avoir 0, 1 ou plusieurs extension(s).

Exemples :

`serie1.cc` : fichier de code source C++

`cours-1.ps.gz`

1re extension indiquant un fichier Postscript

2e extension indiquant un fichier compressé avec `gzip`

Fichiers « cachés »

On distingue les fichiers/répertoires « cachés » au moyen d'une **convention** de nommage : ils sont préfixés par un `.`

Encore un point !

Exemple : `.cshrc`

Ce sont des fichiers/répertoires dont l'utilisateur n'a pas besoin explicitement (ou pas souvent), souvent des fichiers de configuration.

Pour voir les fichiers/répertoires « cachés », utilisez la commande

```
ls -a
```

Fichiers et shell

Un certain nombre des fonctions du shell sont relatives au système de fichiers :

- Navigation dans la structure des fichiers :
  - répertoire courant (`pwd`)
  - modification de ce répertoire (`cd` = change directory),
  - lister le contenu d'un répertoire (`ls`),
  - copier des fichiers (`cp`) et les déplacer (`mv`),
  - effacer des fichiers (`rm`),
  - créer des liens (`ln`), etc...

Toutes les commandes soumises au shell sont interprétées relativement au **répertoire courant**.

Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

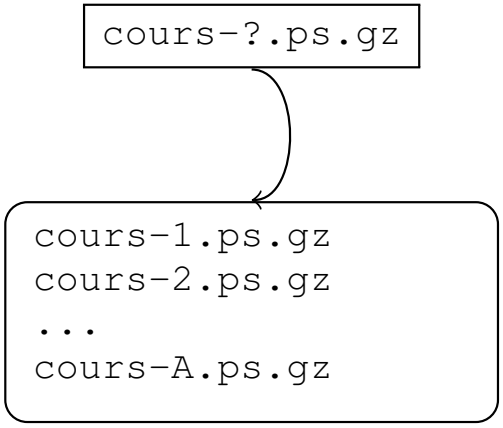
Fichiers et shell (2)

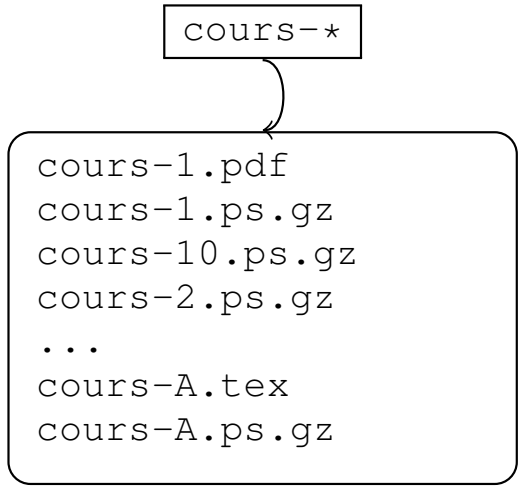
Les caractères de substitution (ou expressions régulières) permettent de spécifier plusieurs fichiers en une seule formule

`?` : remplace un seul caractère arbitraire

`*` : remplace une séquence quelconque de caractères

`[ ]` : remplace un seul caractère parmi ceux entre crochets.





Caractères de substitution
Exemples

`ls ???`	liste tous les fichiers de trois lettres du répertoire courant
`ls *.txt`	liste tous les fichiers du répertoire courant se terminant par `.txt`
`ls *.[ch]`	liste tous les fichiers du répertoire courant se terminant par `.c` ou par `.h`
`ls *.cc *.h`	liste tous les fichiers du répertoire courant se terminant par `.cc` ou par `.h`

Méta-caractères

`*`, `?`, `[`, `]` sont donc des **caractères réservés** (ou « méta-caractères »), en ce sens qu'ils ont un rôle particulier dans l'interpréteur de commandes.

Il en existe d'autres : `| \& ; ( ) < > $ \ " ' ~ ``

Si on souhaite les utiliser, il faut les **protéger** ce qui se fait avec le « backslash » : `\`

Exemple :

```
echo \\ abc \; \<
```

affiche `\ abc ; <`

Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Attributs de fichiers

Les attributs typiques d'un fichier sont :

- son nom
- sa taille
- la date et heure de création
- le propriétaire (créateur)
- les droits d'accès des autres utilisateurs
- modifiable, exécutable, caché,...
- système, possédant des alias,...
- la date et l'auteur de la dernière révision,
- ...

Attributs de fichiers (2)

Dans le cas du système Unix :

- A chaque fichier est associé un utilisateur propriétaire (le créateur du fichier) et un groupe propriétaire (l'un des groupes auxquels appartient l'utilisateur) : essayez la commande `ls -lF`.
- Les droits d'accès définissent trois attributs, et sont paramétrables pour 3 classes d'utilisateurs :

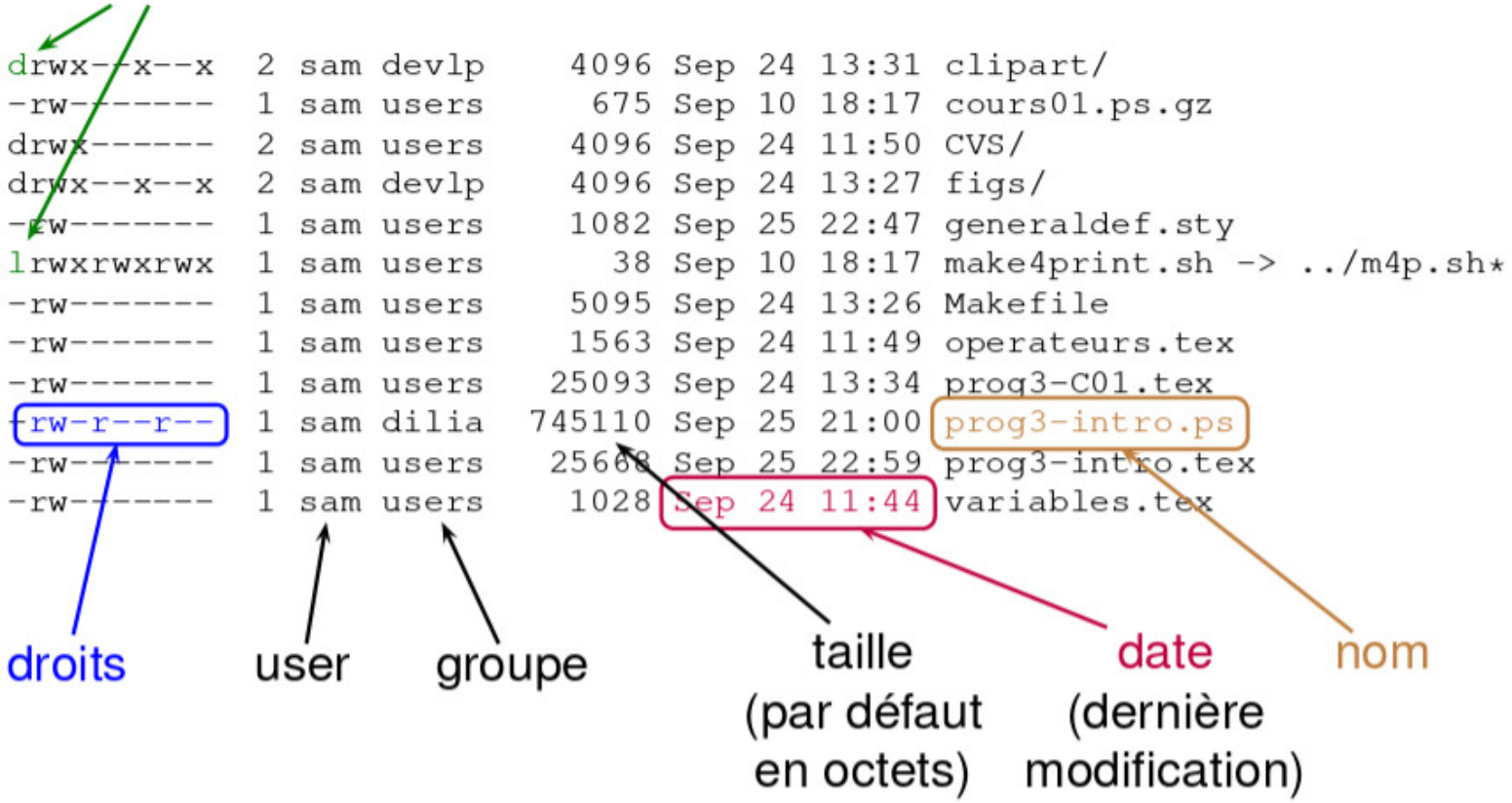
Attributs :	**Classes d'utilisateurs :**
Visibilité (lecture)	Propriétaire (owner ou user)
Modification (écriture, effacement)	Groupe (group)
Exécution	Autres (others)

Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Attributs de fichiers (3)

`ls -lF` (l : c'est un L minuscule, pas le chiffre un.)

Type de fichier : d = répertoire, l (L minuscule) = soft-link



Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Attributs de fichiers (4)

Syntaxe :

`chmod` `(u|g|o|a)*` `+|-` `(r|w|x)*`

- `(u|g|o|a)*` : **U**ser, **G**roup, **O**ther, **A**ll (`a=ugo`)
- `+|-` : `+` : ajouter, `-` : supprimer
- `(r|w|x)*` : **R**ead, **W**rite, e**X**ecute

(*) c'est-à-dire au moins un!

chmod o-r
chmod go+rx
chmod a+w

Exemples :

```
chmod a+x monscript.sh
chmod go-r perso
```

☞ tout le monde peut exécuter ce fichier
☞ les autres que moi n'ont pas le droit de lire ce fichier/repertoire

Jamila Sam
& Jean-Cédric Chappelier
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Editeurs de texte

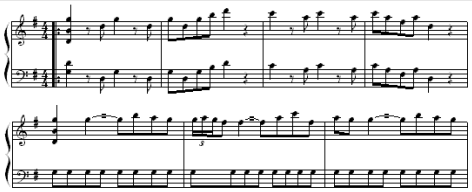
Pour écrire et modifier des fichiers le moyen le plus naturel est d'utiliser un un *éditeur de texte* tels que :

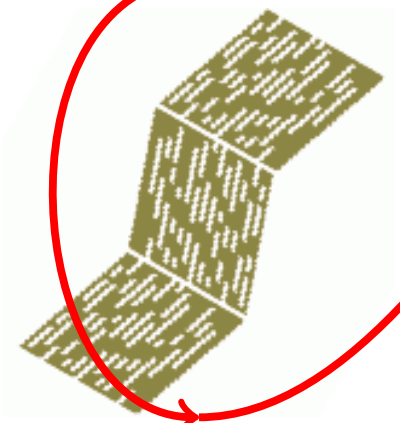
- Emacs
- Geany
- gedit
- Scite
- SublimeText
- notepad ou wordpad (Windows)
- WinEdt (Windows)
- jEdit (Windows, Mac OS X, Linux, ...)
- ...

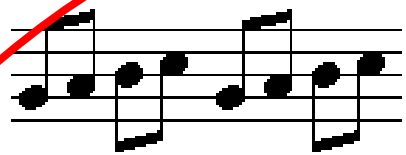
Connaître un/des éditeur(s) de texte est absolument indispensable!

☞ La mini-référence "Environnement Unix", disponible sur le site du cours dédie une de ses sections à Emacs et Geany. Vous aurez aussi l'occasion de les utiliser en TP.

Qu'est-ce que la programmation ?







«PROGRAMME» :
Conception : quelles notes enchainer ?
Réalisation : percer les trous aux bons endroits

Exécution : tourner la manivelle

Résultat : mélodie

Algorithme ?

☞ **solution calculatoire**/mécanique/itérative/... à un **problème**

"Spécification d'un schéma de calcul sous forme d'une suite d'opérations élémentaires obéissant à un enchaînement déterminé"

[Encyclopedia Universalis]

Algorithme

- suite finie de règles à appliquer,
- dans un ordre déterminé,
- à un nombre fini de données,
- se terminant (i.e., arriver, en un nombre fini d'étapes, à un résultat, et ce quelque soit les données traitées).

Algorithme

Un algorithme est un moyen pour un humain de représenter la résolution par calcul d'un problème à une autre personne physique (humain) ou virtuelle (calculateur); un algorithme est:

- séquentiel si ses opérations s'exécutent en séquence,
- parallèle si certaines de ses opérations s'exécutent en parallèle,
- réparti si certaines de ses opérations s'exécutent sur plusieurs machines.

Toute une théorie

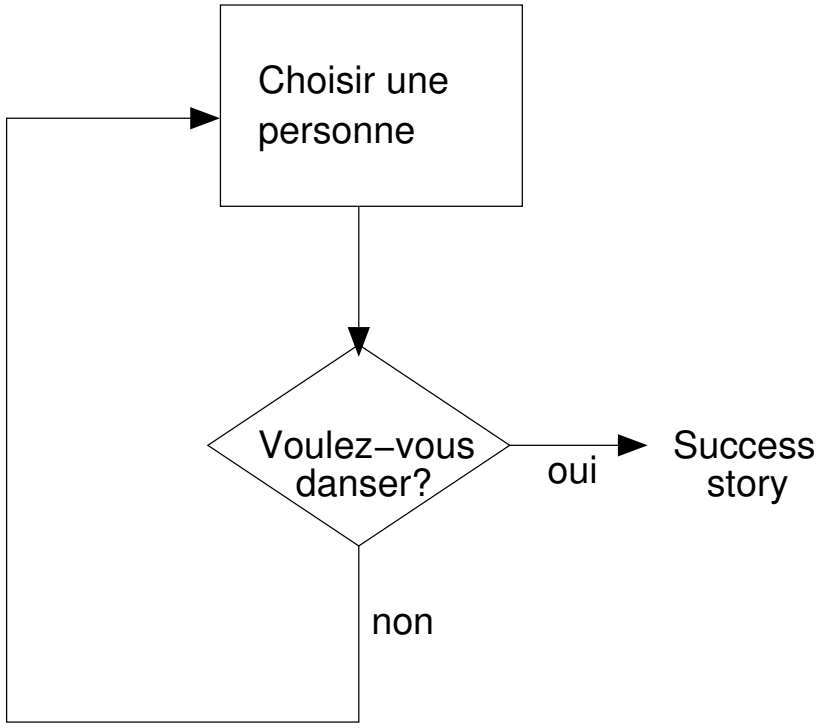
Formalisation : dans les années (19)30 par des mathématiciens :
Gödel, **Turing**, **Church**, **Post**, ...

☞ *fonctions "calculables"* et *machines de Turing* : abstraction mathématique des notions de **traitement** (suite d'opérations élémentaires), de **problème** et d'**algorithme**.

☞ le cours ICC a pour but de vous en donner un aperçu !

Algorithmes et données

« Voulez-vous danser ? » : premier algorithme :



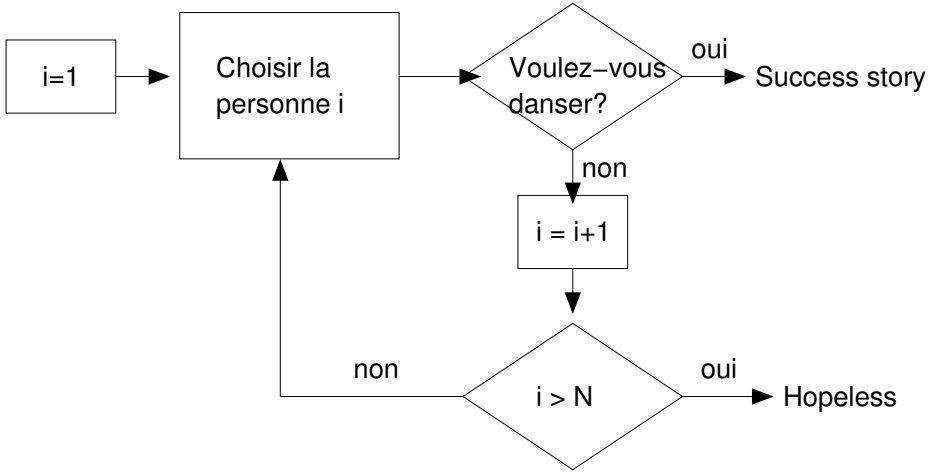
Données :

- Personne
- Ensemble de N personnes

☞ Il n'est pas garanti que l'algorithme puisse se terminer !

Algorithmes et données

« Voulez-vous danser ? » : deuxième algorithme :



Données :

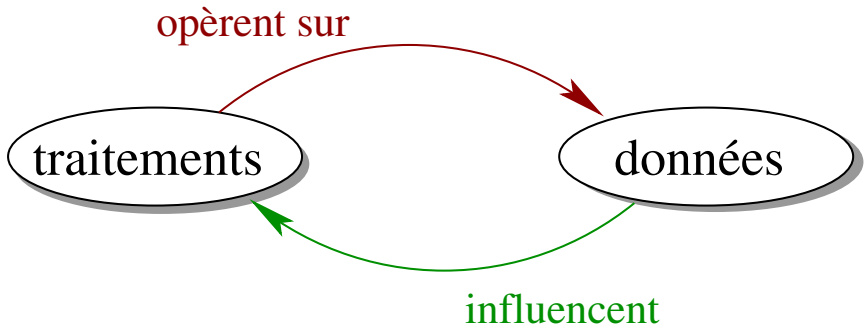
- Personne
- Ensemble **ordonné** de *N* personnes

- les données sont *structurées*
- l'algorithme se **termine** nécessairement (au pire *N* essais successifs)
- . . . mais il n'est pas sûr qu'il soit le plus *efficace* possible !

Conception d'un programme

Concrètement, concevoir un programme c'est décomposer la **tâche** à automatiser sous la forme :

- d'une séquence d'instructions (**traitements**)
- et de **données** permettant une mise en oeuvre efficace et correcte des traitements.



Formalisation des traitements : **algorithmes**

☞ distinguer formellement les bons traitements des mauvais

Formalisation des données : **structures de données (abstraites)**

☞ distinguer formellement les bonnes structures de données des mauvaises

Algorithmes – objectifs

On attend d'un algorithme qu'il :

- se termine,
- produise un résultat correct,
- pour toute donnée d'entrée valable.

☞ Il n'y a (mal)heureusement aucune recette pour produire un algorithme!

Difficulté de l'Informatique

Un programme doit être valable pour toute une gamme d'entrées!

- ☞ Impossible de vérifier par des essais: on ne pourra jamais tester tous les cas.

Vérification par preuves mathématiques.

Importance du travail soigneux et mûrement réfléchi!

Algorithme ≠ Programme

Un algorithme est **indépendant du langage de programmation** dans lequel on va l'exprimer **et de l'ordinateur** utilisé pour l'exécuter

C'est une description abstraite des étapes conduisant à la solution d'un problème.

Algorithme = partie conceptuelle d'un **programme** (indépendante du langage)

Programme = implémentation (i.e., réalisation) de l'**algorithme**, dans un langage de programmation et sur un système particulier.

☞ Premier programme au cours prochain

Pour préparer le prochain cours

- Vidéos et quiz du MOOC semaine 1 :
  - Introduction [09:55]
  - Variables [18:08]
  - Variables : lecture/écriture [13:43]
  - Expressions [14:50]

  ☞ A échelonner sur plusieurs jours pour éviter l'« overdose »

- Le prochain cours :
  - de 15h15 à 16h (résumé et compléments)