Physical Computing
Culture générale en Informatique

Jean-François Monin

Université Grenoble Alpes
Polytech'Grenoble

2013-2017

Physical Computing

Découverte des systèmes embarqués

1. Support matériel : électronique
2. Programmation : informatique
3. Illustration sur un cas d'école réduit mais complet

Informatique

1. Principes généraux (ce cours)
2. Activités pratiques sur architecture Arduino

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Physical Computing : le malaise

Physique : royaume du continu

- L'espace est continu
- Le temps est continu
- Évolution d'un système décrite par des équations différentielles
  – action infinitésimale $\rightarrow$ changement infinitésimal
  – trajectoires prédites par résolution analytiques : intégrales etc.

Physical Computing : le malaise

Physique : royaume du continu

- L'espace est continu
- Le temps est continu
- Évolution d'un système décrite par des équations différentielles
  – action infinitésimale $\rightarrow$ changement infinitésimal
  – trajectoires prédites par résolution analytiques : intégrales etc.

Informatique : royaume du discret

- L'espace est discret
- Le temps est discret
- Évolution d'un système décrite par palliers
  – à intervalles réguliers (horloge) ou non
  – trajectoires prédites par analyse logique ; raisonnement par cas, récurrence etc.

Physical Computing : point commun

Notion d'invariant

- Physique : conservation de l'énergie, de la quantité de mouvement, etc.
- Informatique : principe directeur pour la conception de programmes

Physical Computing : connexion

Discrétisation

- Toute information est représentée sous forme numérique chiffrée
  $\rightarrow$ approximation des valeurs physiques
- Échantillonage $\rightarrow$ valeurs physiques intermédiaires ignorées

Physical Computing : connexion

Discrétisation

- Toute information est représentée sous forme numérique chiffrée
  $\rightarrow$ approximation des valeurs physiques
- Échantillonage $\rightarrow$ valeurs physiques intermédiaires ignorées

Calculs

- Les calculs proprement dits sont exacts
- Mais leur pertinence doit être questionnée
  - problèmes de précision
  - problèmes d'arrondi

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Notion abstraite de calcul

Sur quoi porte un calcul ?

Notion abstraite de calcul

Sur quoi porte un calcul ?

- Sur des nombres

Notion abstraite de calcul

Sur quoi porte un calcul ?

- Sur des nombres
- Sur des textes

Notion abstraite de calcul

Sur quoi porte un calcul ?

- Sur des nombres
- Sur des textes
- Sur des symboles, ou des combinaisons de symboles

Qu'est-ce ce qu'un calcul ?

Notion abstraite de calcul

Sur quoi porte un calcul ?

- Sur des nombres
- Sur des textes
- Sur des symboles, ou des combinaisons de symboles

Qu'est-ce ce qu'un calcul ?

- Exprime un résultat (numérique, symbolique,...) en fonction d'entrées (numériques, symboliques)
- Effectué "machinalement" selon une "recette" précise, appelée un programme
- Effectué par une machine
  - numérique ou symbolique...,
  - physique ou virtuelle

Notion abstraite de calcul

Que représentent les symboles ?

Notion abstraite de calcul

Que représentent les symboles ?

- Des nombres (suites de chiffres) ?

Notion abstraite de calcul

Que représentent les symboles ?

- Des nombres (suites de chiffres) ?
- Des textes (suites de caractères) ?

Notion abstraite de calcul

Que représentent les symboles ?

- Des nombres (suites de chiffres) ?
- Des textes (suites de caractères) ?
- Des données représentant un aspect du monde environnant ?

Notion abstraite de calcul

Que représentent les symboles ?

- Des nombres (suites de chiffres) ?
- Des textes (suites de caractères) ?
- Des données représentant un aspect du monde environnant ?
- Autre chose ?

Notion abstraite de calcul

Que représentent les symboles ?

- Des nombres (suites de chiffres) ?
- Des textes (suites de caractères) ?
- Des données représentant un aspect du monde environnant ?
- Autre chose ?

C'est arbitraire !

Tout est affaire de convention, pour les besoins de la communication.

Notion abstraite de calcul

Que représentent les symboles ?

- Des nombres (suites de chiffres) ?
- Des textes (suites de caractères) ?
- Des données représentant un aspect du monde environnant ?
- Autre chose ?

C'est arbitraire !

Tout est affaire de convention, pour les besoins de la communication.

Ce qui reste à la fin : action de l'environnement (ex. mesure) et action sur l'environnement.

Schéma fondamental

Symboles élémentaires

- Chiffres binaires (0/1), ou bits (binary digit).

Schéma fondamental

Symboles élémentaires

- Chiffres binaires (0/1), ou bits (binary digit).
- Tout (ce qui est discret) peut être représenté (on dit : *codé*) par une suite de bits

Schéma fondamental

Symboles élémentaires

- Chiffres binaires (0/1), ou bits (binary digit).
- Tout (ce qui est discret) peut être représenté (on dit : *codé*) par une suite de bits
- Exemple : la mémoire d'un ordinateur est constituée d'une suite de blocs de n bits, avec n fixé pour chaque machine. Typiquement de nos jours, $n = 32$ ou $n = 64$

Schéma fondamental

Machine munie de capacités
Exemples : effectuer une addition, allumer une lampe, etc.
Chacune est désignée par un code symbolique

Cycle

- lecture d'un code,
- reconnaissance de ce code
- exécution de l'action correspondante
- positionnement à l'étape suivante (prochain code)

Machine = interpréteur

- Code d'un programme
- Code des données

Code et données

Définis par un langage symbolique plus ou moins expressif

Code et données

Définis par un langage symbolique plus ou moins expressif

Pas de signification intrinsèque à un code :
seule la machine qui l'interprète lui donne son sens.

Tension

On a une tension entre

- Les besoins du programmeur
  - données complexes
  - traitements complexes
  - langages riches et expressifs pour les gérer

Tension

On a une tension entre

- Les besoins du programmeur
  - données complexes
  - traitements complexes
  - langages riches et expressifs pour les gérer
- Les possibilités de la physique
  mécanismes élémentaires très simples
  - mécanique
  - électronique : transistors

Magie

Interpréteur programmé

- les données peuvent servir de programme à leur tour

Magie

Interpréteur programmé

- les données peuvent servir de programme à leur tour
- on obtient une machine virtuelle

Magie

Interpréteur programmé

- les données peuvent servir de programme à leur tour
- on obtient une machine virtuelle
- dans laquelle on peut programmer une machine encore plus sophistiquée

Magie

Interpréteur programmé

- les données peuvent servir de programme à leur tour
- on obtient une machine virtuelle
- dans laquelle on peut programmer une machine encore plus sophistiquée
- et ainsi de suite

Magie

Interpréteur programmé

- les données peuvent servir de programme à leur tour
- on obtient une machine virtuelle
- dans laquelle on peut programmer une machine encore plus sophistiquée
- et ainsi de suite

Compilateur

- Traduction d'un langage source
  interprétable sur une machine complexe
- vers un langage cible
  interprétable sur une machine plus simple

Historique

- -3000 Boulier, systèmes de numération...
- -1800 Babylone
- -500 “solfège” indien

Historique

- -3000 Boulier, systèmes de numération...
- -1800 Babylone
- -500 “solfège” indien
- 1640 Pascaline
- 1801 Métier à tisser programmable (Joseph-Marie Jacquard)
- 1832 Code Morse
- 1843 Calcul des nombres de Bernoulli (Ada Lovelace)

Historique

- -3000 Boulier, systèmes de numération...
- -1800 Babylone
- -500 “solfège” indien
- 1640 Pascaline
- 1801 Métier à tisser programmable (Joseph-Marie Jacquard)
- 1832 Code Morse
- 1843 Calcul des nombres de Bernoulli (Ada Lovelace)
- 1935 λ-calcul (Stephen Kleene, Alonzo Church)
- 1936 Machine de Turing
- 1948 ENIAC (John von Neumann)
- 1948 Théorie de l’information (Claude Shannon)

Historique

- -3000 Boulier, systèmes de numération...
- -1800 Babylone
- -500 “solfège” indien
- 1640 Pascaline
- 1801 Métier à tisser programmable (Joseph-Marie Jacquard)
- 1832 Code Morse
- 1843 Calcul des nombres de Bernoulli (Ada Lovelace)
- 1935 λ-calcul (Stephen Kleene, Alonzo Church)
- 1936 Machine de Turing
- 1948 ENIAC (John von Neumann)
- 1948 Théorie de l’information (Claude Shannon)
- 1952-54 Compilateur (G. Hopper) + FORTRAN (J. Backus)

Qu'est-ce qu'un algorithme ?

Qu'est-ce qu'un algorithme ?

"Processus systématique de résolution d'un problème permettant de décrire les étapes vers le résultat." (Wikipédia)

Qu'est-ce qu'un algorithme ?

"Processus systématique de résolution d'un problème permettant de décrire les étapes vers le résultat." (Wikipédia)

"Suite finie et non-ambiguë d'instructions permettant de donner la réponse à un problème." (Wikipédia)

Qu'est-ce qu'un algorithme ?

"Processus systématique de résolution d'un problème permettant de décrire les étapes vers le résultat." (Wikipédia)

"Suite finie et non-ambiguë d'instructions permettant de donner la réponse à un problème." (Wikipédia)

"Ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations." (Larousse)

Qu'est-ce qu'un algorithme ?

"Processus systématique de résolution d'un problème permettant de décrire les étapes vers le résultat." (Wikipédia)

"Suite finie et non-ambiguë d'instructions permettant de donner la réponse à un problème." (Wikipédia)

"Ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations." (Larousse)

En réalité, il est difficile de s'accorder sur une définition pleinement satisfaisante.
On peut retenir qu'un algorithme représente l'idée essentielle d'un programme.

Et formellement ?

La réponse dépend du *modèle* de calcul.

Et formellement ?

La réponse dépend du *modèle* de calcul.
Machine de Turing, λ-calcul, automate cellulaire, fonctions récursives... (équivalents du point de vue de l'expressivité)

Et formellement ?

La réponse dépend du *modèle* de calcul.
Machine de Turing, λ-calcul, automate cellulaire, fonctions récursives... (équivalents du point de vue de l'expressivité)

Organigrammes (grafcet...)

- proches des machines physiques

Et formellement ?

La réponse dépend du *modèle* de calcul.
Machine de Turing, λ-calcul, automate cellulaire, fonctions récursives... (équivalents du point de vue de l'expressivité)

Organigrammes (grafcet...)

- proches des machines physiques
- $\rightarrow$ « plats de nouilles »

Composition d'actions élémentaires

Et formellement ?

La réponse dépend du *modèle* de calcul.
Machine de Turing, λ-calcul, automate cellulaire, fonctions récursives... (équivalents du point de vue de l'expressivité)

Organigrammes (grafcet...)

- proches des machines physiques
- $\rightarrow$ « plats de nouilles »

Composition d'actions élémentaires

Dans la discipline impérative structurée

- La composition séquentielle
- La composition conditionnelle
- La composition itérative
- La composition récursive

Qu'est-ce que la programmation ?

Qu'est-ce que la programmation ?

Ensemble de techniques qui permettent de décrire un algorithme sous une forme "compréhensible" par la machine.

Qu'est-ce que la programmation ?

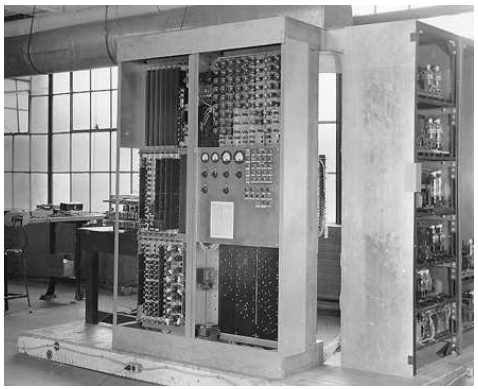
Ensemble de techniques qui permettent de décrire un algorithme sous une forme "compréhensible" par la machine.

- Un même algorithme donne des programmes très différents

Qu'est-ce que la programmation ?

Ensemble de techniques qui permettent de décrire un algorithme sous une forme "compréhensible" par la machine.

- Un même algorithme donne des programmes très différents
- Dépend de la machine MAIS



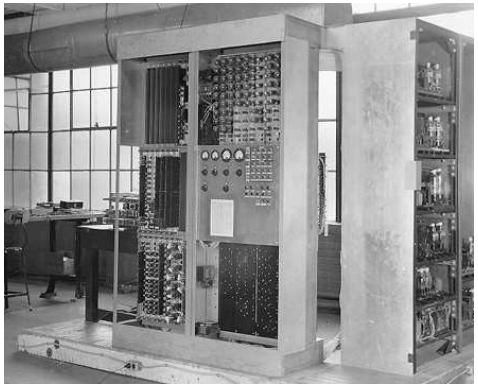
ENIAC 1948 (Von Neumann)

Même architecture !

Qu'est-ce que la programmation ?

Ensemble de techniques qui permettent de décrire un algorithme sous une forme "compréhensible" par la machine.

- Un même algorithme donne des programmes très différents
- Dépend de la machine MAIS



ENIAC 1948 (Von Neumann)

Même architecture !

- Langage de haut niveau : la traduction en instructions propres à la machine est assurée par un programme
- N'apprend rien de plus que l'algorithme mais le rend exécutable à grande échelle

Syntaxe et sémantique

Syntaxe :

Respect de la grammaire formelle d'un langage

Syntaxe et sémantique

Syntaxe :

Respect de la grammaire formelle d'un langage

Sémantique :

Donne du sens à un langage de programmation

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Les types de base :

$\{vrai, faux\}$: booléen	`bool`
$\mathbb{Z}$: entiers relatifs	`int`
$\mathbb{D}$: décimaux	`float`
Alphabet	`char`
⋮	⋮

munis de leurs opérations : + - * / ...

Deux notions de variable

$$A = 2\sin x + 3x$$

Deux notions de variable

$$A = 2\sin x + 3x$$

En Mathématiques

La valeur de A *dépend* de la valeur de x (paramètre formel, grandeurs liées).

- x prend ses valeurs dans un ensemble donné
- À *chaque* valeur de x correspond une valeur de A
- On l'explicite en écrivant $A(x) = 2\sin x + 3x$
- x, y, z, x_1, x_2, n, a, b...

Deux notions de variable

$$A = 2\sin x + 3x$$

En Mathématiques

La valeur de A *dépend* de la valeur de x (paramètre formel, grandeurs liées).

- x prend ses valeurs dans un ensemble donné
- À *chaque* valeur de x correspond une valeur de A
- On l'explicite en écrivant $A(x) = 2\sin x + 3x$
- x, y, z, x_1, x_2, n, a, b...

Parfois confondue avec une inconnue ($f(x) = k$)

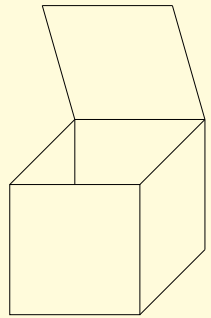
- 0, 1 ou plusieurs valeurs
- à déterminer par un raisonnement

Deux notions de variable

```
A = 2 sin(x) + 3 x
```

En informatique : actions

- `x` d
- Présent ou pas dans un *état* donné
- Contient une valeur



- Son contenu peut changer durant l'exécution d'un programme
- `A = 2 sin(x) + 3 x` utilise la valeur de `x` dans l'état courant pour calculer la valeur de `A` dans le nouvel état.
- `maximum`, `salaire`, `Tableau[i]`...

Deux notions de variable

```
A = 2 sin(x) + 3 x
```

En informatique : expressions

- 2 sin(x) + 3 * x

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Le signe =

En Mathématiques :

$f(x) = 2x + 1$
$1 = sin(x)^2 + cos(x)^2$
$2x + 1 = 0$
$a^{\varphi(n)} \equiv 1 \pmod{n}$

Le signe =

En Mathématiques :

$f(x) = 2x + 1$
$1 = sin(x)^2 + cos(x)^2$
$2x + 1 = 0$
$a^{\varphi(n)} \equiv 1 \pmod{n}$

En Informatique : deux utilisations au moins de =

- Test : $x == 3$
- Affectation : $x = 3$

Action élémentaire : l'affectation

Notion d'état

a	b	x	somme	...
2	10	0	128	...

Action élémentaire : l'affectation

Notion d'état

a	b	x	somme	...
2	10	0	128	...

L'affectation change l'état de la mémoire

Variable $\leftarrow$ expression

Action élémentaire : l'affectation

Notion d'état

a	b	x	somme	...
2	10	0	128	...

L'affectation change l'état de la mémoire

Variable $\leftarrow$ expression

Exemples : x $\leftarrow$ 1

$P \leftarrow a \times b$

En Python : `x = 1`

`P = a * b`

Action élémentaire : l'affectation

Notion d'état

a	b	x	somme	P	...
2	10	1	128	20	...

L'affectation change l'état de la mémoire

Variable $\leftarrow$ expression

Exemples : x $\leftarrow$ 1
$P \leftarrow a \times b$

En Python : `x = 1`
`P = a * b`

Autres notations

```
a := b
a = b
a prend la valeur b
b -> a
...
```

Exercices faciles

Premier exercice

a ← 2
b ← a * a - 2
a ← b - a

Exercices faciles

Premier exercice

	a	b
a ← 2	2	?
b ← a * a - 2	2	2*2 - 2 = 2
a ← b - a	2 - 2 = 0	2

Exercices faciles

Premier exercice

	a	b
a ← 2	2	?
b ← a * a - 2	2	2*2 - 2 = 2
a ← b - a	2 - 2 = 0	2

Échanger a et b ?

Exercices faciles

Premier exercice

	a	b
a ← 2	2	?
b ← a * a - 2	2	2*2 - 2 = 2
a ← b - a	2 - 2 = 0	2

Échanger a et b ?

tmp ← a
a ← b
b ← tmp

En Python

```
tmp = a
a = b
b = tmp
```

Exercices faciles

Premier exercice

	a	b
a ← 2	2	?
b ← a * a - 2	2	2*2 - 2 = 2
a ← b - a	2 - 2 = 0	2

Échanger a et b ?

tmp ← a
a ← b
b ← tmp

En Python

```
tmp = a
a = b
b = tmp
```

Coût en espace mémoire :
3 (entiers)

Échangeons

Échanger a et b en coût en espace mémoire = 2 (entiers) ?

Que fait cet algorithme ?

Échangeons

Échanger a et b en coût en espace mémoire = 2 (entiers) ?

a $\leftarrow$ a + b
b $\leftarrow$ a - b
a $\leftarrow$ a - b

Que fait cet algorithme ?

Échangeons

Échanger a et b en coût en espace mémoire = 2 (entiers) ?

	a	b
$a \leftarrow a + b$	$a_0 + b_0$	b_0
$b \leftarrow a - b$	$a_0 + b_0$	$(a_0 + b_0) - b_0 = a_0$
$a \leftarrow a - b$	$(a_0 + b_0) - a_0 = b_0$	a_0

Que fait cet algorithme ?

Échangeons

Échanger a et b en coût en espace mémoire = 2 (entiers) ?

	a	b
a ← a + b	$a_0 + b_0$	b_0
b ← a - b	$a_0 + b_0$	$(a_0 + b_0)$ - $b_0 = a_0$
a ← a - b	$(a_0 + b_0)$ - $a_0 = b_0$	a_0

Que fait cet algorithme ?

b ← a + b
a ← a + b
b ← a - b
a ← a - 2b

Échangeons

Échanger a et b en coût en espace mémoire = 2 (entiers) ?

	a	b
a ← a + b	$a_0 + b_0$	b_0
b ← a - b	$a_0 + b_0$	$(a_0 + b_0)$ - $b_0 = a_0$
a ← a - b	$(a_0 + b_0)$ - $a_0 = b_0$	a_0

Que fait cet algorithme ?

	a	b
b ← a + b	a_0	a_0+b_0
a ← a + b	$2a_0+b_0$	a_0+b_0
b ← a - b	$2a_0+b_0$	a_0
a ← a - 2b	b_0	a_0

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Composition conditionnelle

Si C *alors* A_1 *sinon* A_2

\+ Cas particulier *Si* C *alors* A_1

Composition conditionnelle

Si C alors A_1 *sinon* A_2

\+ Cas particulier *Si C alors* A_1

Exemple : maximum de A et B ?

Si A $<$ B
    alors MAX $\leftarrow$ B
    sinon MAX $\leftarrow$ A

```
if A < B:
    MAX = B
else:
    MAX = A
```

- Coût = Coût(test) + max (Coût (alors), Coût (sinon))
- Coût en espace mémoire = 3 (entiers)

Composition conditionnelle

Si C alors A_1 *sinon* A_2

\+ Cas particulier *Si C alors* A_1

Exemple : maximum de A et B ?

Si A $<$ B
alors MAX $\leftarrow$ B
sinon MAX $\leftarrow$ A

```
if A < B:
␣␣␣␣MAX = B
else:
␣␣␣␣MAX = A
```

- Coût = Coût(test) + max (Coût (alors), Coût (sinon))
- Coût en espace mémoire = 3 (entiers)

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Composition itérative

Selon une condition de terminaison C.

Tantque (*non C*) *faire A*

Exemple : Calculer la somme des n premiers entiers

Composition itérative

Selon une condition de terminaison C.

Tantque (non C) faire A

Exemple : Calculer la somme des *n* premiers entiers

```
i ← 0
somme ← 0
Tantque (i < n) :
    i ← i+1
    somme ← somme + i
```

```
i = 0
somme = 0
while i < n:
    i = i + 1
    somme = somme + i
```

Composition itérative

Selon une condition de terminaison C.

Tantque (*non C*) *faire A*

Exemple : Calculer la somme des *n* premiers entiers

i $\leftarrow$ 0	`i = 0`
somme $\leftarrow$ 0	`somme = 0`
Tantque (i $<$ n) :	`while i < n:`
$\quad$ i $\leftarrow$ i+1	`␣␣␣␣i = i + 1`
$\quad$ somme $\leftarrow$ somme + i	`␣␣␣␣somme = somme + i`

La complexité d'une boucle est la somme des coûts de toutes les itérations.

Composition itérative

Selon une condition de terminaison C.

Tantque (*non C*) *faire A*

Exemple : Calculer la somme des n premiers entiers

```
i ← 0                          i = 0
somme ← 0                      somme = 0
Tantque (i < n) :              while i < n:
    i ← i+1                    ␣␣␣␣i = i + 1
    somme ← somme + i          ␣␣␣␣somme = somme + i
```

La complexité d'une boucle est la somme des coûts de toutes les itérations.

- Coût = $n + 1$ tests + $2n$ additions
- Coût en espace mémoire = 3 (entiers)

Preuve de correction d'un algorithme

```
i ← 0
somme ← 0
Tantque (i < n) :
    i ← i+1
    somme ← somme + i
```

Invariant

Preuve de correction d'un algorithme

i $\leftarrow$ 0
somme $\leftarrow$ 0
Tantque (i $<$ n) :
    i $\leftarrow$ i+1
    somme $\leftarrow$ somme + i

Invariant

$$somme = \sum_{k=0}^{k=i} k$$

Preuve de correction d'un algorithme

i $\leftarrow$ 0
somme $\leftarrow$ 0 $\{somme = \sum_{k=0}^{k=i} k,\ i.e\ \ 0 = 0\}$
Tantque (i $<$ n) :
    i $\leftarrow$ i+1
    somme $\leftarrow$ somme + i

Invariant

$$somme = \sum_{k=0}^{k=i} k$$

Preuve de correction d'un algorithme

i $\leftarrow$ 0
somme $\leftarrow$ 0 $\{somme = \sum_{k=0}^{k=i} k,\ i.e\ \ 0 = 0\}$
Tantque (i $<$ n) : $\{somme = \sum_{k=0}^{k=i} k\}$
    i $\leftarrow$ i+1
    somme $\leftarrow$ somme + i

Invariant

$$somme = \sum_{k=0}^{k=i} k$$

Preuve de correction d'un algorithme

i $\leftarrow$ 0
somme $\leftarrow$ 0 $\{somme = \sum_{k=0}^{k=i} k,\ i.e\ \ 0 = 0\}$
Tantque (i $<$ n) : $\{somme = \sum_{k=0}^{k=i} k\}$
    i $\leftarrow$ i+1
    somme $\leftarrow$ somme + i
$\{somme' = somme + i' = \sum_{k=0}^{k=i} k + i + 1 = \sum_{k=0}^{k=i+1} k,\ i.e.\ \sum_{k=0}^{k=i'} k\}$

Invariant

$$somme = \sum_{k=0}^{k=i} k$$

Exercice : que calcule cet algorithme ?

```
# A et B sont des entiers > 0
R = A
Q = 0
while R >= B:
    R = R - B
    Q = Q + 1
# Résultat : le couple (Q,R)
```

Exercice : que calcule cet algorithme ?

```
# A et B sont des entiers > 0
R = A
Q = 0
while R >= B:
    R = R - B
    Q = Q + 1
# Résultat : le couple (Q,R)
```

Calcule le quotient et le reste de la division euclidienne de A par B.

Exercice : que calcule cet algorithme ?

```
# A et B sont des entiers > 0
R = A
Q = 0
while R >= B:
    R = R - B
    Q = Q + 1
# Résultat : le couple (Q,R)
```

Calcule le quotient et le reste de la division euclidienne de A par B.

Complexité

Exercice : que calcule cet algorithme ?

```
# A et B sont des entiers > 0
R = A
Q = 0
while R >= B:
    R = R - B
    Q = Q + 1
# Résultat : le couple (Q,R)
```

Calcule le quotient et le reste de la division euclidienne de A par B.

Complexité

- Coût en nombre d'opérations sur les entiers $2 * Q + 2$
- Coût en espace mémoire $= 4$ (entiers)

Prouver cet algorithme de division euclidienne ?

$R \leftarrow A$
$Q \leftarrow 0$
Tantque $(R \geq B)$:
    $R \leftarrow R$ - B
    $Q \leftarrow Q + 1$

Prouver cet algorithme de division euclidienne ?

$R \leftarrow A$
$Q \leftarrow 0$
Tantque $(R \geq B)$:
    $R \leftarrow R$ - B
    $Q \leftarrow Q + 1$

Invariant

Prouver cet algorithme de division euclidienne ?

R $\leftarrow$ A
Q $\leftarrow$ 0
Tantque (R $\geq$ B) :
    R $\leftarrow$ R - B
    Q $\leftarrow$ Q + 1

Invariant

$$B * Q + R = A \text{ et } R \geq 0$$

Prouver cet algorithme de division euclidienne ?

$\{$A et B deux entiers $> 0\}$
R $\leftarrow$ A
Q $\leftarrow$ 0 $\quad \{B * Q + R = B * 0 + A = A$, et $R \geq 0$ car $A > 0\}$
Tantque (R $\geq$ B) :
    R $\leftarrow$ R - B
    Q $\leftarrow$ Q + 1

Invariant

$$B * Q + R = A \text{ et } R \geq 0$$

Prouver cet algorithme de division euclidienne ?

$\{$A et B deux entiers $> 0\}$
R $\leftarrow$ A
Q $\leftarrow$ 0 $\quad \{B * Q + R = B * 0 + A = A$, et $R \geq 0$ car $A > 0\}$
Tantque (R $\geq$ B) : $\quad \{B * Q + R = A$ et $R \geq 0\}$
    R $\leftarrow$ R - B
    Q $\leftarrow$ Q + 1

Invariant

$$B * Q + R = A \text{ et } R \geq 0$$

Prouver cet algorithme de division euclidienne ?

$\{$A et B deux entiers $> 0\}$
R $\leftarrow$ A
Q $\leftarrow$ 0 $\quad \{B * Q + R = B * 0 + A = A$, et $R \geq 0$ car $A > 0\ \}$
Tantque (R $\geq$ B) : $\quad \{B * Q + R = A$ et $R \geq 0\ \}$
    R $\leftarrow$ R - B
    Q $\leftarrow$ Q + 1
    $\{B * Q' + R' = B * (Q + 1) + (R - B) = B * Q + R = A$
        et $R' = R - B \geq 0$ car $R \geq B$ *(condition du Tantque)*$\}$

Invariant

$$B * Q + R = A \text{ et } R \geq 0$$

Prouver cet algorithme de division euclidienne ?

$\{$A et B deux entiers $> 0\}$
R $\leftarrow$ A
Q $\leftarrow$ 0 $\quad \{B * Q + R = B * 0 + A = A$, et $R \geq 0$ car $A > 0\ \}$
Tantque (R $\geq$ B) : $\quad \{B * Q + R = A$ et $R \geq 0\ \}$
    R $\leftarrow$ R - B
    Q $\leftarrow$ Q + 1
    $\{B * Q' + R' = B * (Q + 1) + (R - B) = B * Q + R = A$
        et $R' = R - B \geq 0$ car $R \geq B$ *(condition du Tantque)*$\}$
$\{A = B * Q + R$ et $0 \leq R < B\}$

Invariant

$$B * Q + R = A \text{ et } R \geq 0$$

Composition itérative

Pour $i = 1$ à n

Exemple : Calculer la somme des n premiers entiers

```
Somme ← 0
Pour i = 1 à n :
    Somme ← Somme + i
```

- Coût =
- Coût en espace mémoire =

Composition itérative

Pour $i = 1$ à n

Exemple : Calculer la somme des n premiers entiers

Somme $\leftarrow$ 0
Pour i = 1 à n :
    Somme $\leftarrow$ Somme + i

- Coût = $2n$ additions + n tests
- Coût en espace mémoire = 3 (entiers)

Peut-on faire mieux ?

Composition itérative

Pour $i = 1$ à n

Exemple : Calculer la somme des n premiers entiers

```
Somme ← 0
Pour i = 1 à n :
    Somme ← Somme + i
```

- Coût $= 2n$ additions $+$ n tests
- Coût en espace mémoire $= 3$ (entiers)

Peut-on faire mieux ? Gauss : $n(n+1)/2$

Composition itérative

Pour $i = 1$ à n

Exemple : Calculer la somme des n premiers entiers

```
Somme ← 0
Pour i = 1 à n :
    Somme ← Somme + i
```

- Coût $= 2n$ additions $+$ n tests
- Coût en espace mémoire $= 3$ (entiers)

Peut-on faire mieux ? Gauss : $n(n+1)/2$

Exercices possibles : somme des n premiers entiers, x^n, k-ième élément de u_n, intérêts composés ...

Composition itérative

Pour $i = 1$ à n

Exemple : Calculer la somme des n premiers entiers

Somme $\leftarrow$ 0	`somme = 0`
Pour i = 1 à n :	`for i in range(n+1):`
Somme $\leftarrow$ Somme + i	`somme = somme + i`

- Coût = $2n$ additions + n tests
- Coût en espace mémoire = 3 (entiers)

Peut-on faire mieux ? Gauss : $n(n+1)/2$

Exercices possibles : somme des n premiers entiers, x^n, k-ième élément de u_n, intérêts composés ...

Digression : les listes Python

Listes ou plutôt tableaux

Un ensemble de N éléments numérotés de 0 à N-1

```
# Liste de 5 entiers
L = [15,42,1,23,-10]

# Accès à chacun des éléments de la liste
somme = L[0] + L[1] + L[2] + L[3] + L[4]
L[4] = 0
```

Une liste particulière :

l'instruction `range (N)`

Elle permet de créer la liste des N premiers entiers de 0 à $N-1$.

```
>>> L = range(10)
>>> L[0]
0
>>> L[5]
5
```

Une liste particulière :

l'instruction `range (N)`

Elle permet de créer la liste des N premiers entiers de 0 à $N-1$.

```
>>> L = range(10)
>>> L[0]
0
>>> L[5]
5
```

Variantes :

`range(M, N)` va de M à $N-1$.

`range(M, N, P)` va de M à $N-1$ avec un pas P.

for ... in ...

En Python l'itération s'effectue par parcours d'une liste :

```
L = [15,42,1,23,-10]

for i in L:
    j = i*i
    print "Un nombre",i,"et son carré",j
```

for ... in ...

En Python l'itération s'effectue par parcours d'une liste :

```
L = [15,42,1,23,-10]

for i in L:
    j = i*i
    print "Un nombre",i,"et son carré",j
```

On utilisera en général la forme

```
for i in range(N):
    ...
    ...
```

Pour i allant de 0 à N-1 :

...

...

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Notion de fonction en informatique

Motivation

- Organiser le code
- Isoler des parties réutilisables
- Faciliter le développement et la maintenance

Notion de fonction en informatique

Motivation

- Organiser le code
- Isoler des parties réutilisables
- Faciliter le développement et la maintenance

En Python :

```
def nomdelafonction(arguments):
␣␣␣␣...
␣␣␣␣return valeur
```

Notion de fonction en informatique

Motivation

- Organiser le code
- Isoler des parties réutilisables
- Faciliter le développement et la maintenance

En Python :

```
def nomdelafonction(arguments):
␣␣␣␣...
␣␣␣␣return valeur
```

Exemple :

```
def abs(x):
    if x < 0:
        return -x
    else:
        return x
```

Exercice : maximum

- Écrire un algorithme qui calcule le maximum de 2 entiers.

Exercice : maximum

- Écrire un algorithme qui calcule le maximum de 2 entiers.

Une solution

Si A $>$ B alors Max $\leftarrow$ A sinon Max $\leftarrow$ B

Exercice : maximum

- Écrire un algorithme qui calcule le maximum de 2 entiers.

Une solution

Si A $>$ B alors Max $\leftarrow$ A sinon Max $\leftarrow$ B

- Écrire une fonction qui calcule le maximum de deux entiers ?

Exercice : maximum

- Écrire un algorithme qui calcule le maximum de 2 entiers.

Une solution

Si A $>$ B alors Max $\leftarrow$ A sinon Max $\leftarrow$ B

- Écrire une fonction qui calcule le maximum de deux entiers ?

Une solution en Python

```
def max2 (x,y):
    if x > y:
        return x
    else:
        return y
```

Maximum de A, B, C et D.

Écrire un algorithme qui calcule le maximum de 4 entiers.

Maximum de A, B, C et D.

Écrire un algorithme qui calcule le maximum de 4 entiers.

Solution 1

```
Si A>B et A>C et A>D Alors Max ← A
Sinon
    Si B>A et B>C et B>D alors Max ←B
    Sinon
        Si C>A et C>B et C>D alors Max ← C
        Sinon Max ← D
```

Solution 2

```
Si A>B et A>C et A>D Alors Max ← A
Sinon
    Si B>C et B>D alors Max ←B
    Sinon
        Si C>D alors Max ← C
        Sinon Max ← D
```

Maximum de A, B, C et D.

- Écrire un algorithme qui calcule le maximum de 4 entiers ?

Maximum de A, B, C et D.

- Écrire un algorithme qui calcule le maximum de 4 entiers ?

Solution 3 :

```
Max = A
if B > Max:     Max = B
if C > Max:     Max = C
if D > Max:     Max = D
```

Maximum de A, B, C et D.

- Écrire un algorithme qui calcule le maximum de 4 entiers ?

Solution 3 :

```
Max = A
if B > Max:     Max = B
if C > Max:     Max = C
if D > Max:     Max = D
```

Solution 4 : En utilisant max2

M1 = max2(A,B)
M2 = max2(C,D)
Max = max2(M1,M2)

Maximum de A, B, C et D.

- Écrire un algorithme qui calcule le maximum de 4 entiers ?

Solution 3 :

```
Max = A
if B > Max:     Max = B
if C > Max:     Max = C
if D > Max:     Max = D
```

Solution 4 : En utilisant max2

M1 = max2(A,B)
M2 = max2(C,D)
Max = max2(M1,M2)

ou encore max2(max2 (A, B), max2 (C, D))

Composition récursive

Une action A est récursive si elle est exprimée par une composition d'actions dont elle fait elle-même partie.

Factorielle : $n!$

- $0! = 1$
- $n! = n * (n-1)!$

Composition récursive

Une action A est récursive si elle est exprimée par une composition d'actions dont elle fait elle-même partie.

Factorielle : $n!$

- $0! = 1$
- $n! = n * (n-1)!$

En Python :

```
def fact (n) :
    if n == 0:
        return 1
    else:
        return n * fact (n-1)
```

Composition récursive

Une action A est récursive si elle est exprimée par une composition d'actions dont elle fait elle-même partie.

Factorielle : $n!$

- $0! = 1$
- $n! = n * (n-1)!$

En Python :

```
def fact (n) :
    if n == 0:
        return 1
    else:
        return n * fact (n-1)
```

Exemple :
fact(2) = 2*fact(1) = 2*1*fact(0)= 2*1*1= 2

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Factorielle

Écrire un algorithme qui calcule factorielle avec une boucle tant que.

Quizz

Quel programme calcule factorielle n dans la variable F ?

A

```
i ← 0 ;
F ← 1 ;
Tantque (i <= n) :
    i ← i +1 ;
    F ← F * i ;
```

B

```
i ← 0 ;
F ← 1 ;
Tantque (i < n) :
    i ← i +1 ;
    F ← F * i ;
```

C

```
i ← 1 ;
F ← 1 ;
Tantque (i <= n) :
    F ← F * i ;
    i ← i +1 ;
```

D

```
i ← 0 ;
F ← 1 ;
Tantque (i < n) :
    F ← F * i ;
    i ← i +1 ;
```

Programme A

```
i ← 0 ;
F ← 1 ;
Tantque (i <= n) :
    i ← i +1 ;
    F ← F * i ;
```

Invariant

Programme A

```
i ← 0 ;
F ← 1 ;
Tantque (i <= n) :
    i ← i +1 ;
    F ← F * i ;
```

Invariant

$$F = i!$$

Programme A

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = i!, \textit{ i.e. } F = 1 = 0! = i! \}$
Tantque (i <= n) :
    i $\leftarrow$ i +1 ;
    F $\leftarrow$ F * i ;

Invariant

$$F = i!$$

Programme A

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = i!, \textit{ i.e. } F = 1 = 0! = i!\}$
Tantque (i <= n) : $\{F = i!\}$
    i $\leftarrow$ i +1 ;
    F $\leftarrow$ F * i ;

Invariant

$$F = i!$$

Programme A

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = i!, \textit{ i.e. } F = 1 = 0! = i!\ \}$
Tantque (i <= n) : $\{F = i!\}$
    i $\leftarrow$ i +1 ;
    F $\leftarrow$ F * i ;
    $\{F' = F * i' = F * (i+1) = i! * (i+1) = (i+1)!\ \textit{i.e. } F' = i'!\}$

Invariant

$$F = i!$$

Programme A

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = i!, \textit{ i.e. } F = 1 = 0! = i!\}$
Tantque (i <= n) : $\{F = i!\}$
    i $\leftarrow$ i +1 ;
    F $\leftarrow$ F * i ;
    $\{F' = F * i' = F * (i+1) = i! * (i+1) = (i+1)! \textit{ i.e. } F' = i'!\}$
MAIS $\{F = i! \text{ et } i = n+1 \textit{ i.e. } F = (n+1)!\}$

Invariant

$$F = i!$$

Programme A

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = i!, \textit{ i.e. } F = 1 = 0! = i!\}$
Tantque (i <= n) : $\{F = i!\}$
    i $\leftarrow$ i +1 ;
    F $\leftarrow$ F * i ;
    $\{F' = F * i' = F * (i+1) = i! * (i+1) = (i+1)! \textit{ i.e. } F' = i'!\}$
MAIS $\{F = i! \text{ et } i = n+1 \textit{ i.e. } F = (n+1)!\}$

Calcule $(n+1)!$ et non $n!$, exemple : si $n = 1$ alors $F = 1 * 2$

Invariant

$$F = i!$$

Programme $B!$

```
i ← 0 ;
F ← 1 ;
Tantque (i < n) :
    i ← i +1 ;
    F ← F * i ;
```

Invariant

Programme $B!$

```
i ← 0 ;
F ← 1 ;
Tantque (i < n) :
    i ← i +1 ;
    F ← F * i ;
```

Invariant

$$F = i!$$

Programme $B!$

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\quad \{F = (i-1)!$ *i.e.* $F = 1 = 0! = i!\}$
Tantque (i $<$ n) :
    i $\leftarrow$ i +1 ;
    F $\leftarrow$ F * i ;

Invariant

$$F = i!$$

Programme $B!$

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = (i-1)!$ *i.e.* $F = 1 = 0! = i!\}$
Tantque (i $<$ n) : $\{F = i!\}$
    i $\leftarrow$ i +1 ;
    F $\leftarrow$ F * i ;

Invariant

$$F = i!$$

Programme $B!$

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = (i-1)!$ *i.e.* $F = 1 = 0! = i!\}$
Tantque (i $<$ n) : $\{F = i!\}$
    i $\leftarrow$ i +1 ;
    F $\leftarrow$ F * i ; $\{F' = F * i' = i! * (i+1) = (i+1)!$ *i.e.* $F' = i'!\}$

Invariant

$$F = i!$$

Programme $B!$

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = (i-1)!$ *i.e.* $F = 1 = 0! = i!\}$
Tantque (i $<$ n) : $\{F = i!\}$
    i $\leftarrow$ i +1 ;
    F $\leftarrow$ F * i ; $\{F' = F * i' = i! * (i+1) = (i+1)!$ *i.e.* $F' = i'!\}$
DONC $\{F = i!$ et $i = n$ *i.e.* $F = n!\}$

Invariant

$$F = i!$$

Programme $C!$

```
i ← 1 ;
F ← 1 ;
Tantque (i <= n) :
    F ← F * i ;
    i ← i +1 ;
```

Invariant

Programme $C!$

```
i ← 1 ;
F ← 1 ;
Tantque (i <= n) :
    F ← F * i ;
    i ← i +1 ;
```

Invariant

$$F = (i - 1)!$$

Programme $C!$

i $\leftarrow$ 1 ;
F $\leftarrow$ 1 ; $\{F = i!$ *i.e.* $F = 1 = 0! = (1-1)! = (i-1)!\}$
Tantque (i <= n) :
    F $\leftarrow$ F * i ;
    i $\leftarrow$ i +1 ;

Invariant

$$F = (i-1)!$$

Programme $C!$

i $\leftarrow$ 1 ;
F $\leftarrow$ 1 ; $\{F = i!$ *i.e.* $F = 1 = 0! = (1-1)! = (i-1)!\}$
Tantque (i <= n) : $\{F = (i-1)!\}$
    F $\leftarrow$ F * i ;
    i $\leftarrow$ i +1 ;

Invariant

$$F = (i-1)!$$

Programme $C!$

i $\leftarrow$ 1 ;
F $\leftarrow$ 1 ; $\{F = i!$ *i.e.* $F = 1 = 0! = (1-1)! = (i-1)!\}$
Tantque (i <= n) : $\{F = (i-1)!\}$
    F $\leftarrow$ F * i ;
    i $\leftarrow$ i +1 ; $\{F' = F * i = (i-1)! * i = i!$ *i.e.*
    $F' = (i+1-1)! = (i'-1)!\}$

Invariant

$$F = (i-1)!$$

Programme $C!$

i $\leftarrow$ 1 ;
F $\leftarrow$ 1 ; $\{F = i!$ *i.e.* $F = 1 = 0! = (1-1)! = (i-1)!\}$
Tantque (i <= n) : $\{F = (i-1)!\}$
F $\leftarrow$ F * i ;
i $\leftarrow$ i +1 ; $\{F' = F * i = (i-1)! * i = i!$ *i.e.*
$F' = (i+1-1)! = (i'-1)!\}$
À la sortie de la boucle nous avons F= (i-1) ! et i = n+1 DONC F= n !

Invariant

$$F = (i-1)!$$

Programme *D*

```
i ← 0 ;
F ← 1 ;
Tantque (i < n) :
    F ← F * i ;
    i ← i +1 ;
```

Invariant

Programme D

```
i ← 0 ;
F ← 1 ;
Tantque (i < n) :
    F ← F * i ;
    i ← i +1 ;
```

Invariant

$$F = i!$$

Programme D

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = i!$ *i.e.* $F = 1 = 0! = i!\}$
Tantque (i $<$ n) :
    F $\leftarrow$ F * i ;
    i $\leftarrow$ i +1 ;

Invariant

$$F = i!$$

Programme D

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = i!$ *i.e.* $F = 1 = 0! = i!\}$
Tantque (i $<$ n) : $\{F = i!\}$
    F $\leftarrow$ F * i ;
    i $\leftarrow$ i +1 ;

Invariant

$$F = i!$$

Programme D

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = i!$ *i.e.* $F = 1 = 0! = i!\}$
Tantque (i $<$ n) : $\{F = i!\}$
    F $\leftarrow$ F * i ;
    i $\leftarrow$ i +1 ; $\{F' = F * i = i! * i$ *i.e.* $F' \neq i'!\}$

Invariant

$$F = i!$$

Programme D

i $\leftarrow$ 0 ;
F $\leftarrow$ 1 ; $\{F = i!$ *i.e.* $F = 1 = 0! = i!\}$
Tantque (i $<$ n) : $\{F = i!\}$
    F $\leftarrow$ F * i ;
    i $\leftarrow$ i +1 ; $\{F' = F * i = i! * i$ *i.e.* $F' \neq i'!\}$

On ne peut pas terminer la preuve car cet algorithme calcule $i * i!$ avec i inititalisé à 0 (cad 0) et non $n!$
Exemple : si $n = 1$ alors $F = 0 * 1 = 0$

Invariant

$$F = i!$$

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Moyenne

Proposer un algorithme qui calcule la moyenne d'un tableau *Tab* de taille $n > 0$.

Moyenne

Proposer un algorithme qui calcule la moyenne d'un tableau *Tab* de taille $n > 0$.

Algorithme

moy $\leftarrow$ 0 ;
Pour $i = 0$ jusqu'à $n - 1$:
    moy $\leftarrow$ moy + Tab[i]/n ;

Moyenne

Proposer un algorithme qui calcule la moyenne d'un tableau *Tab* de taille $n > 0$.

Algorithme

moy $\leftarrow$ 0 ;
Pour $i = 0$ jusqu'à $n - 1$:
    moy $\leftarrow$ moy + Tab[i]/n ;

Attention aux arrondis

Médiane

Calculer la médiane d'un tableau *Tab* de taille n ?

Médiane

Calculer la médiane d'un tableau *Tab* de taille n ?

Algorithme naïf :

Médiane

Calculer la médiane d'un tableau *Tab* de taille n ?

Algorithme naïf :

- Tant que le tableau a une taille plus grande que 3, oter le maximum et le minimum.
- Ensuite calculer la médiane d'un tableau d'un ou deux éléments.

Médiane

Calculer la médiane d'un tableau *Tab* de taille *n* ?

Algorithme naïf :

- Tant que le tableau a une taille plus grande que 3, oter le maximum et le minimum.
- Ensuite calculer la médiane d'un tableau d'un ou deux éléments.

Algorithme

Tab $\leftarrow$ trier(Tab)
Si n % 2 = 1
    Alors med $\leftarrow$ Tab[n/2]
    Sinon med $\leftarrow$ (Tab[n/2-1]+Tab[n/2]) /2

Médiane

Calculer la médiane d'un tableau *Tab* de taille *n* ?

Algorithme naïf :

- Tant que le tableau a une taille plus grande que 3, oter le maximum et le minimum.
- Ensuite calculer la médiane d'un tableau d'un ou deux éléments.

Algorithme

```
Tab ← trier(Tab)
Si n % 2 = 1
        Alors med ← Tab[n/2]
        Sinon med ← (Tab[n/2-1]+Tab[n/2]) /2
```

Il existe un algorithme linéaire.

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Puissance rapide

Idée mathématique

$$\text{puis}(x,\, n) = \begin{cases} x, & \text{si } n = 1 \\ \text{puis}(x^2,\, n/2), & \text{si } n \text{ est pair} \\ x \times \text{puis}(x^2,\, (n-1)/2), & \text{si } n > 2 \text{ est impair} \end{cases}$$

Puissance rapide

Idée mathématique

$$\text{puis}(x, n) = \begin{cases} x, & \text{si } n = 1 \\ \text{puis}(x^2, n/2), & \text{si } n \text{ est pair} \\ x \times \text{puis}(x^2, (n-1)/2), & \text{si } n > 2 \text{ est impair} \end{cases}$$

```
def puisrapide(x,n):
    if (n==1) :
        return x
    else :
        if (n%2==0) :
            return puisrapide(x*x,n/2)
        else :
            return x * puisrapide(x*x, (n-1)/2)
```

Complexité logarithmique $O(log_2(n))$.

Puissance rapide

Version itérative

```
def puissrapide(x,n):
        res = 1
        while n != 0 :
            if (n%2 ==0) :
                x = x*x
                n = n/2
            else :
                res = res*x
                n= n-1
        return res
```

Un outil universel

- Pour le raisonnement
- Expressions booléennes utilisées dans les structures de contrôle (it-then-else, boucle while)
- Circuits électroniques pour les opérations de base au niveau du matériel
- en particulier, les calculs arithmétiques se ramènent à du calcul booléen !

La logique pour le raisonnement

Un raisonnement permet de déduire une conclusion à partir d'hypothèses
On s'intéresse uniquement à la forme d'un raisonnement. S'il est correct, la conclusion tient dès que les hypothèses sont vérifiées.
S'assurer de la pertinence des hypothèses ne relève pas le logique (mais de la physique, de la chimie, de l'observation...)

Exemple

H1) La nationale 90 est enneigée
H2) Toute route enneigée ou verglassée est glissante
H3) Toute route glissante est dangereuse
C) La nationale 90 est dangereuse

Exemple

H1) La nationale 90 est enneigée
H2) Toute route enneigée ou verglassée est glissante
H3) Toute route glissante est dangereuse
C) La nationale 90 est dangereuse

Preuve : De H1 on déduit que la nationale 90 est enneigée ou verglassée. Par H2, il en résulte qu'elle est glissante. Pas H3, il en résulte qu'elle est dangereuse.

Exemple

H1) La nationale 90 est enneigée
H2) Toute route enneigée ou verglassée est glissante
H3) Toute route glissante est dangereuse
C) La nationale 90 est dangereuse

Preuve : De H1 on déduit que la nationale 90 est enneigée ou verglassée. Par H2, il en résulte qu'elle est glissante. Pas H3, il en résulte qu'elle est dangereuse.

Dans la suite, on se limite à la logique propositionnelle (pas de quantificateurs, "tout", "il existe").

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Syntaxe des formules logique

On dénote des propositions élémentaires par des lettres p, q, etc.
On combine des propositions existantes au moyen de

connecteurs logiques

- la conjonction $\wedge$ (et)
- la disjonction $\vee$ (ou)
- la négation $\neg$ (non)
- l'implication $\Rightarrow$ (implique, entraîne)
- l'équivalence $\Leftrightarrow$ (équivaut à)

Exemples

- $p \wedge \neg q \Rightarrow \neg(p \Rightarrow q)$
- $p \vee q \Rightarrow p$

Raisonnement correct

Soient $H_1, \ldots H_n$ et C des formules logiques
Un raisonnement est une déduction de C en supposant $H_1, \ldots H_n$
Notation : $H_1, \ldots H_n, \vdash C$
Variante : démontrer $\vdash (H_1 \wedge \ldots \wedge H_n) \Rightarrow C$

Cela peut se faire par différentes techniques

- règles de déduction logique
  Exemple : de $A \wedge B$, on peut déduire A
- règles de simplification
  Exemple $\neg(A \vee B)$ est interchangeable avec $\neg A \wedge \neg B$
- tables de vérité

Plan

Introduction
- Le malaise
- La magie

Algorithmique structurée
- Les objets
- Affectation et état
- Conditionnelle
- Itérations et invariant
- Fonctions et récursivité

Exemples d'algorithmes
- Factorielle
- Moyenne, Médiane etc ...
- Complexité

Logique
- Raisonnement
- Algèbre booléenne

Tables de vérité

On ne s'intéresse pas à la signification des propositions, mais uniquement à leur *valeur de vérité*, qui peut être *faux* (dénoté par 0) ou *vrai* (dénoté par 1).
Une valeur de vérité est appelée un *booléen* (ou *valeur booléenne*).

Tables de vérité

On ne s'intéresse pas à la signification des propositions, mais uniquement à leur *valeur de vérité*, qui peut être *faux* (dénoté par 0) ou *vrai* (dénoté par 1).
Une valeur de vérité est appelée un *booléen* (ou *valeur booléenne*).

La valeur de vérité d'une formule dépend de la valeur de vérité des variables propositionnelles qui la constituent.

Tables de vérité

On ne s'intéresse pas à la signification des propositions, mais uniquement à leur *valeur de vérité*, qui peut être *faux* (dénoté par 0) ou *vrai* (dénoté par 1).
Une valeur de vérité est appelée un *booléen* (ou *valeur booléenne*).

La valeur de vérité d'une formule dépend de la valeur de vérité des variables propositionnelles qui la constituent.

p	q	$\neg p$	$p \vee q$	$p \wedge q$	$p \Rightarrow q$	$p \Leftrightarrow q$
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

Exercice

Trouver les valeurs de vérité de

- $p \vee q \Rightarrow p$
- $p \wedge \neg q \Rightarrow \neg(p \Rightarrow q)$

en fonction des valeurs de vérité de p et q

Lorsque la valeur de vérité d'une formule A est 1 dans tous les cas, on dit que A est une *tautologie*

Équivalences remarquables

- associativité $x \vee (y \vee z) \equiv (x \vee y) \vee z$, $x \wedge (y \wedge z) \equiv (x \wedge y) \wedge z$
- commutativité $x \vee y \equiv y \vee x$, $x \wedge y \equiv y \wedge x$
- idempotence $x \vee x \equiv x$, $x \wedge x \equiv x$
- distributivité $x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z)$,
  $x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z)$
- neutralité $0 \vee x \equiv x$, $1 \wedge x \equiv x$
- absorbtion $1 \vee x \equiv 1$, $0 \wedge x \equiv 0$
- $x \wedge \neg x \equiv 0$, $x \vee \neg x \equiv 1$

Équivalences remarquables

- $x \vee \neg x \equiv 1$ (tiers exclu), $x \wedge \neg x \equiv 0$
- $\neg\neg x \equiv x$
- $\neg(x \wedge y) \equiv \neg x \vee \neg y$
- $\neg(x \vee y) \equiv \neg x \wedge \neg y$
- $x \vee (x \wedge y) \equiv x$
- $x \wedge (x \vee y) \equiv x$
- $x \vee (\neg x \wedge y) \equiv x \vee y$

Problème inverse

Théorème

Toute fonction booléenne peut être représentée en utilisant uniquement $\wedge$, $\vee$ et $\neg$

Corollaire

Toute fonction booléenne peut être représentée en utilisant uniquement le *nand* : $\neg(x \wedge y)$; ou le *nor* : $\neg(x \vee y)$

Application importante

Pour les circuits logiques, on peut se contenter d'un jeu de portes limité :

- $\wedge$, $\vee$ et $\neg$
- *nand*
- *nor*